A Probabilistic Approach to Estimating Computer System Reliability

Robert Apthorpe – Excite@Home, Inc.

ABSTRACT

Probabilistic Risk Assessment (PRA) is a method of estimating system reliability by combining logic models of the ways systems can fail with numerical failure rates. One postulates a failure state and systematically decomposes this state into a combination of more basic events through a process known as Fault Tree Analysis (FTA). Failure rates are derived from vendor specifications, historical trends, on-call reports, and many other sources. FTA has been used for decades in the defense, aerospace, and nuclear power industries to manage risk and increase reliability of complex engineering systems. Combining FTA with event tree analysis (ETA), one can associate failure probabilities with consequences to clearly communicate risk both pictorially and numerically. Basic PRA techniques can help increase the reliability and security of computer systems.

Introduction

As system administrators, our primary responsibility is to maximize the availability of the systems in our charge. This responsibility is a constant challenge, exacerbated by economic and competitive pressure, the continuing rapid growth of the Internet, and institutions' increasing reliance on network services. As systems become more ubiquitous and more important to an organization, more emphasis is placed on reliability [1].

Often we desire a quantitative measure of reliability. We are inundated with diagnostic information and in recent years have focused on data analysis. Unfortunately, this data is only a historical record; it tells us little about underlying system behavior, dependencies, or latent vulnerabilities, and its predictive value is limited.

The computer industry lags behind other industries in terms of risk assessment methodologies. We rarely assess risk formally, usually only when designing a system. System administrators lack the most basic means of comprehensively analyzing and quantifying risks to the systems they maintain.

We seek a formal approach to risk assessment that offers a qualitative understanding of system behavior and vulnerability. The approach should be systematic and comprehensive, producing quantitative measures of risk and component importance consistent with observed historical data. We want a technique that produces lasting benefits for the effort spent and that can be understood and applied by the average system designer or administrator. Above all, we desire a method with an established record of success.

One such technique is Probabilistic Risk Assessment (PRA), a systematic method of enumerating the ways a system can fail. PRA considers both probability and consequences of individual events, affords a scalable level of detail, and can model human reliability as well as software and hardware reliability. Developed in the 1960s, PRA has seen widespread use in the aerospace, defense, and nuclear power industries [11].

Intent

This paper serves as a brief tutorial on event tree analysis (ETA) and fault tree analysis (FTA). I will model a typical mail receipt system using ETA, then decompose one event into a fault tree. I will quantify the fault tree and show metrics that help analysts estimate the importance of components. Finally, I will discuss limitations of the method and suggest areas for future study and research.

The techniques I describe have developed over the last 40 years and combine such fields as probability theory, graph theory, systems engineering, operations research, and statistics [19] [20]. Despite the age of technique, introductory material on PRA is scarce. Often PRA techniques are only taught at the graduate level making the subject even less accessible, especially to practicing technologists.

Reliability analysts must thoroughly understand the systems they model from the high-level systemic scale down to the component level. They also need broad knowledge of the interrelationships between systems and the tenacity to ferret out subtle dependencies. System administrators already have many of these skills and all that is needed is an introduction to the technique. PRA reduces risk assessment to a rational, mostly mechanical process.

Preliminary Analysis

Most formal risk or reliability assessments start with these three steps:

1. Identify the hazard
2. Identify relevant systems, components, and individuals
3. Bound the analysis

While this may seem simplistic and obvious, it's vitally important to know at the outset which events one is concerned with, which systems are to be analyzed, and the level of detail one will consider.

Example: Inbound Mail Transport

An organization (wazmo.org) considers it vital that no inbound mail is ever irrevocably lost. Reviewing the mail transport process [14, 15, 16, 17] we see the following relevant systems, components, and individuals:

- Remote sender
- Remote user host
- Remote MUA (mail user agent, e.g., client software)
- Remote MTA (mail transport agent, e.g., mail server software)
- Remote network
- The Domain Name System (DNS)
- The Internet
- wazmo.org's ISP
- Local network
- Local MTAs
- Local user host
- Local MUA
- Local recipient

Expanding our scope, we can consider the following:

- Power transmission and distribution network
- On-site environment (temperature, humidity, etc.)
- Off-site environment (likelihood of fire, flood, ice storm, earthquake, etc.)
- Human activity (inept or malicious)
- Extraterrestrial activity (solar flares, magnetic storms, etc.)
- Political climate (civil unrest, war)

As esoteric as some of these items seem, there's ample anecdotal and documentary evidence of systems being challenged by each of them [12] [13]. Emergency response organizations often rely on pagers which in turn rely on satellite systems; the reliability of pager networks is frequently taken for granted despite occasional outages due to satellite misconfiguration and solar flares. The same is true for systems such as the national electric power grid and the public switched telephone network.

To simplify our analysis we will only model portions of wazmo.org's local site, their ISP (mail servers, name servers), and the Internet (the connection from wazmo.org to cynistar.net, its ISP.)

The mail servers are listed in DNS (db.wazmo.org) in decreasing preference as follows:

```
wazmo.org 1D IN MX 10 shaft.mx.wazmo.org.
      1D IN MX 20 dolomite.mx.wazmo.org.
      1D IN MX 30 mta00.cynistar.net.
```

This shows two internal mail servers (shaft and dolomite) and an external backup mail server (mta00.) Shaft is the primary with a preference of 10, dolomite is secondary with a preference of 20, and mta00 is tertiary with a preference of 30.

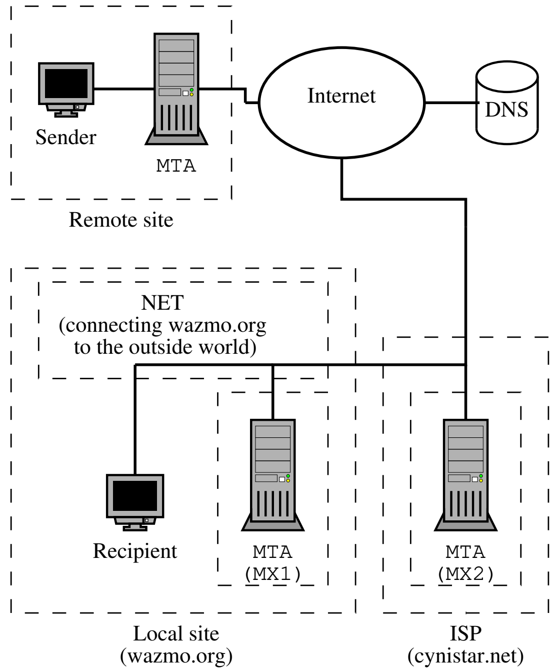

Figure 1: Simple mail transport model.

Fundamentals of Event Tree Analysis

An event tree is a diagram of all the events that can occur in a system, a graphical form of truth table. Event tree analysis is an inductive approach, in that it answers the question "What if event X happens?" One starts with an initiating event (root node) on the left side of the diagram proceeding though a series of branch points that represent the occurrence and magnitude of particular events. Each of the branches has an associated probability or split fraction. The end states (leaf nodes) represent the combination of events leading to a particular consequence. The result is a tree structure that clearly shows the events leading to particular consequences and allows a quantitative measure of risk to be derived without much effort. This technique is best illustrated with an example.

Event Tree Construction

Start by defining the consequence of interest and systems or functions relating to that consequence. In this case, we define the consequence as "loss of inbound mail" and we choose the systems to be:

- **DNS**: can the domain name system resolve the appropriate destination for inbound mail (specifically MX records)?
- **NET**: can remote hosts reach local mail servers? Can local users reach local mail servers?
- **MX1**: are local (primary) mail servers accepting mail?
- **MX2**: are remote (secondary) mail servers accepting mail?

Our initiating event is "mail sent to wazmo.org." This is a high-probability event for all but the smallest domains. Combining these events, we construct a *basic* event tree, one containing all possible branches. Some states make no sense and can be eliminated, leading to a *reduced* event tree. For example, if DNS cannot resolve the address of the receiving MTA (either MX1 or MX2), the states of the local network, and primary and secondary mail servers are irrelevant. If DNS, NET, and MX1 are available, the state of MX2 is irrelevant since mail will be delivered to MX1 and there will be no demand for MX2. Finally, if the local network (NET) is unavailable, the state of MX1 is irrelevant; once the connection attempt to MX1 fails the sending MTA will try MX2 next. The reduced event tree tersely and clearly explains a fairly complex system.

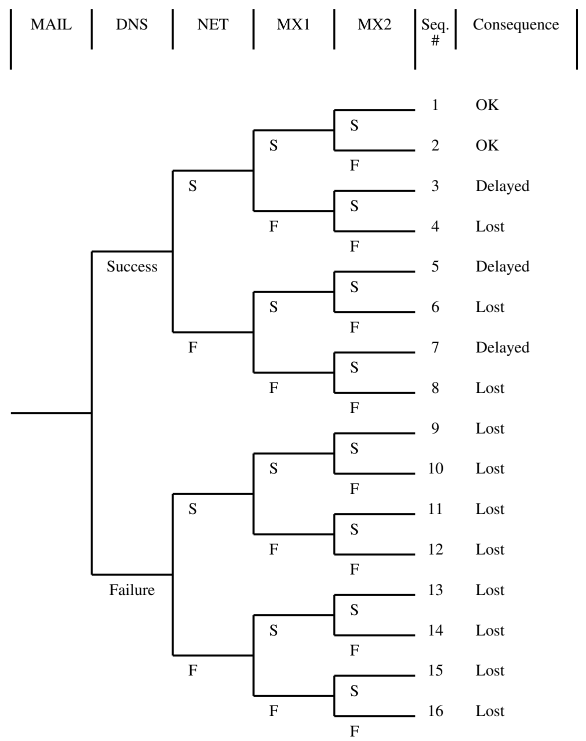

Figure 2: Basic event tree.

We can estimate reliability with an event tree by associating probabilities with each branch point. For example, we could send test messages to MX1 and MX2 and record the number of failures to estimate the reliability of the mail exchangers. If the reliability of DNS is given as P_{DNS}, reliability of the site's network and internet connectivity (NET) is P_{NET}, and reliability of MX1 and MX2 are given by P_{MX1} and P_{MX2} respectively, the probability of prompt mail delivery (sequence #1 in Figure 3) is

$$\begin{aligned} P_{ok} &= P_1 \\ &= P_{MAIL} \times P_{DNS} \times P_{NET} \times P_{MX1} \times \\ &\qquad (P_{MX2} + (1 - P_{MX2})) \\ &= P_{MAIL} \times P_{DNS} \times P_{NET} \times P_{MX1} \end{aligned}$$

where P_{MAIL} is the probability that inbound mail will arrive during the interval of interest. The probability of mail being delayed (assuming a delay in transport from MX2 to MX1 when it is eventually repaired) is the sum of the probability of sequences #2 and #4 from Figure 3:

$$P_2 = P_{MAIL} \times P_{DNS} \times P_{NET} \times (1 - P_{MX1}) \times P_{MX2}$$

$$P_4 = P_{MAIL} \times P_{DNS} \times (1 - P_{NET}) \times P_{MX2}$$

$$\begin{aligned} P_{delayed} &= P_2 + P_4 \\ &= P_{MAIL} \times P_{DNS} \times P_{MX2} \times (1 - P_{NET} \times P_{MX1}) \end{aligned}$$

And finally,

$$\begin{aligned} P_{LOST} &= P_3 + P_5 + P_6 \\ &= (1 - (P_1 + P_2 + P_4)) \\ &= (1 - P_{ok} - P_{delayed}) \end{aligned}$$

In this simple case each branch point has only two states {Success, Failure}; it is possible to have more than two states per event. For example, we could add a third state to MX1 and MX2 – {Success, Degraded, Failure} – and model another situation in which inbound mail is delayed. The sum of each state's probability at a branch point must add up to unity to encompass all possible events (e.g., DNS either works or it doesn't; MX1 is either fully-functional, degraded, or failed; there are no other states than the ones we consider.) Also, the probabilities do not need to be consistent along different paths, that is, P_{MX2} in sequences #2 and #3 need not have the same value as P_{MX2} in sequences #4 and #5. However P_{MX2} must be consistent in sequences #2 and #3 since by definition $P_{MX2_{success}} + P_{MX2_{failure}} \equiv 1$ at each branch point.

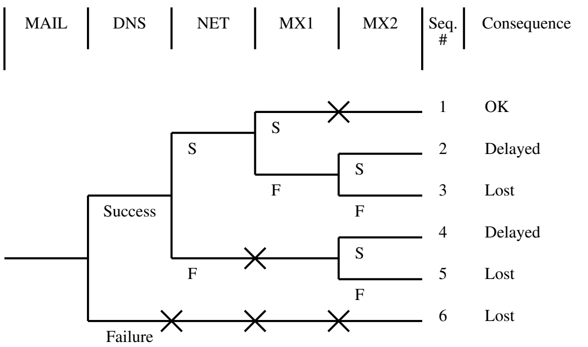

Figure 3: Reduced event tree.

The next task is to derive numerical values for each of these failure probabilities. One may estimate probabilities or use observed data but in more sophisticated models these values are derived from fault tree analyses.

Fundamentals of Fault Tree Analysis

The most common PRA technique is fault tree analysis (FTA). A fault tree is a logical model of the various parallel and sequential combinations of faults that will result in the occurrence of a predefined undesired event. This event (known as a *top event* due to its position in the tree) is linked to more basic events through a number of intermediate events and logic gates detailed below. Fault tree analysis is a deductive technique which asks ''How can event X occur?''

Fault trees provide a detailed graphical explanation of system behavior. In this respect, fault trees are a communication tool that can effectively explain complex systems to those not intimately familiar with the details of a system's design.

Concepts and Definitions

Most of this information is summarized from NUREG-0492 [2] and [19, 20]. It has been reordered somewhat for clarity.

Faults vs. Failures

For modeling purposes, we distinguish between failures (basic abnormal occurrences) and faults (higher-order undesirable events.) Component failures are a subset of a larger class of events known as faults. This helps us distinguish between more concrete basic events and more abstract intermediate events. The reason for this distinction will become clear later, especially in the case of command faults.

State-of *Fault Categories*

Faults may be categorized as *state-of-component* faults or *state-of-system* faults. State-of-component faults suggest the fault should be decomposed into primary, secondary, and command faults. State-of-system faults are often caused by action outside a component and suggest that further decomposition of the fault is necessary. The decomposition process for state-of-system faults is not as mechanical as that for state-of-component faults.

Component Fault Categories

Component faults can be classified as primary, secondary, and command faults.

Primary faults are faults of a component that occur when the component is operating within its design specification, e.g., a web server rated at 50 TPS (transactions per second) fails at a load of 30 TPS.

Secondary faults are faults of a component that occur when the component is operating outside its design specification, e.g., a web server rated at 50 TPS (transactions per second) fails at a load of 90 TPS.

Command faults result when a component properly performs its intended function but at the wrong time or place, e.g., the fire suppression system in a data center discharges due to a spurious or premature signal from a detector or controller.

Events

Events can be categorized as *top*, *intermediate*, or *primary*, depending on their position in the fault tree. In computational tree terminology, primary events are leaf nodes and top events are root nodes. Primary events are further subdivided into *basic*, *undeveloped*, *conditioning*, and *external* events. Basic events are are considered atomic and are not expanded further. These are usually primary, secondary, or command faults. Undeveloped events are just that; they indicate events that are not considered atomic but for reasons of uncertainty or simplicity are not expanded further. Conditioning events show any conditions or restrictions on that apply to any logic gate (usually inhibit or priority-and gates.) External events (sometimes known as *house* events) signify events that are normally expected to occur; they are not faults by themselves. They are often used for sensitivity analysis. Conditioning and external events are rarely used in practice. See Figure 4 for a summary of event types and their symbols.

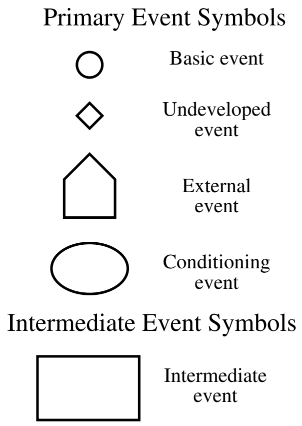

Figure 4: Event types.

Gates

The logic gates used in fault tree analysis include the familiar and, or, and exclusive-or gates. transfer-in and transfer-out gates are conveniences that allow a large fault tree to be broken into sections that fit on the printed page. The inhibit gate is a special form of the and gate composed of a single input event and a conditioning event. The priority-and gate is a special form of and gate where the input events must occur in a specified order (by convention input events must occur in order from left to right in the diagram.) The m-of-n gate is true when at least *m* of the *n* input events occur; this gate is used to model voting logic or multiply-redundant systems. Many of these gates are for convenience purposes. In practice, one can effectively model systems using only and and or gates, using transfer gates to break up the tree into printable sections. Figure 5 summarizes these gates and their symbols.

Cutsets

A *cutset* is a set of basic events which, if they occur, guarantees the top event will occur. A *minimal cutset* is a cutset which, if any event is removed, will no longer be a cutset (e.g., no longer guarantees the top event will occur.) The primary purpose of fault tree analysis is to identify minimal cutsets.

For the curious, ''cutset'' is a term from graph theory. Fault tree analysis works in *failure space* looking for sets of events that prevent successful operation. It is possible to convert a fault tree to a connected graph of success events (events that must occur to ensure successful operation.) The set of events that disconnect (cut) a connected graph is a cutset [21]; this implies that fault tree analysis may have its origin in success path analysis (in graph theory parlance, the fault tree is the dual of a graph of success paths.) While it's far beyond the scope of this paper to explore etymology, history, and graph theory, it may be helpful to review the underlying mathematics and history for additional insight.

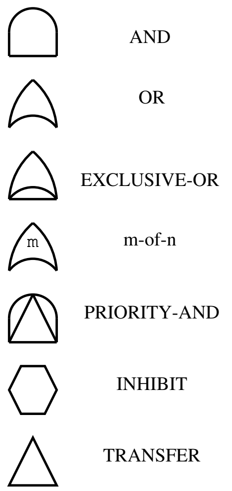

Figure 5: Gate types.

Fault Tree Construction

The basic steps in performing a fault tree analysis are

1. Define failures of interest (top events), usually by an inductive process such as event tree analysis. The combination of fault- and event-tree analysis is sometimes known as cause-consequence analysis (CCA).
2. Define the systems to be analyzed and the limit of resolution of the analysis.
3. Build logical models (fault trees) of the events that lead to each of the top events defined in Step 1.
4. Evaluate the models to determine the sets of basic events (minimal cutsets) that lead to each top event.
5. Optionally, quantify the likelihood of each minimal cutset using component failure probabilities.

To explore this technique, we return to our mail system analysis. For simplicity, we will only model the MX1 event show in Figure 3.

Fault Tree Analysis of Inbound Mail Transport System

We begin the analysis by defining our top event as "Local MTA fails to accept mail" This fault is clear, direct, and unambiguous. It is best to phrase events tersely and directly, usually in a single sentence containing no more than fifteen to twenty words. Remember, the top event states what happens; the fault tree explains how.

Now we define our system. We have two mail servers and a single dedicated switch in common as shown in Figure 6.

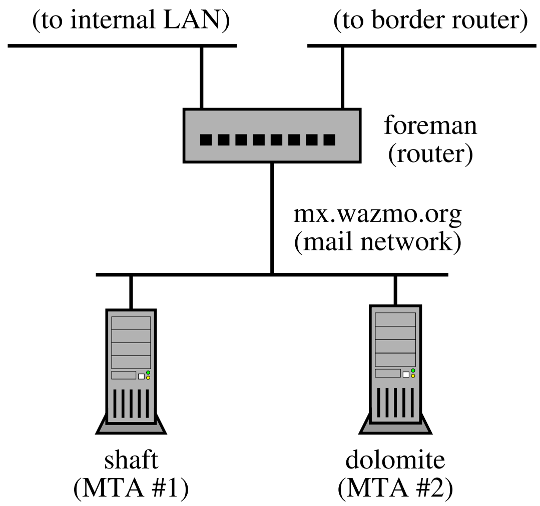

Figure 6: Physical description of local Mail exchangers (MX1).

The system operates as follows:

1. Incoming mail traffic is routed from the site's border routers to the incoming mail network
2. According to DNS, MTA #1 (shaft.mx.wazmo.org) is listed with a preference of 10, MTA #2 (dolomite.mx.wazmo.org) has a preference is 20, and the off-site mail server (mta00.cynistar.net) is listed with a preference of 30.
3. Remote mail servers will first attempt delivery to MTA #1. If MTA #1 will not accept incoming mail, the remote server will try to deliver mail to MTA #2. Failing that, it will then deliver mail to the off-site mail server. Undelivered mail is silently discarded.

Note that we assume no failures in the border routers and internal networking (NET), the off-site mail server (MX2) or in DNS. We only concentrate on the system we've defined as MX1; failures in these other systems will be detected when these systems are analyzed separately. One benefit of fault tree analysis is that we can decompose our analysis into smaller, more manageable pieces.

We can simplify the analysis by making some assumptions about the components and the system:

1. The mail servers are identical and are capable of handling all traffic directed to them under normal circumstances. That is, only one mail server needs to be available for the system to perform within its design specification.

2. Failures are permanent. No automated or manual recovery actions are assumed.
3. Components are independent of each other. This is a very important assumption and will be discussed later.
4. The router and both mail servers are served by the same source of electrical power.
5. We only consider faults in the components under analysis. We do not consider faults due to problems with cabling or room cooling, nor do we consider events such as theft, fire, flood, seismicity, etc.

The definition of a system includes modeling assumptions and design basis assumptions, the "design basis" being the range of conditions within which a system is designed to operate.

Next, we construct a fault tree using the rules described in the Fault Tree Handbook [2]:

- Ground Rule I: Write the statements that are entered in the event boxes as faults; state precisely what the fault is and when it occurs.
- Ground Rule II: If the answer to the question "Can this event consist of a component failure?" is "Yes," classify the event as a "state-of-component fault." If the answer is "No," classify the event as a "state-of-system fault." If the fault event is classified as "state-of-component," add an or-gate below the event and look for primary, secondary and command modes. If the fault event is classified as "state-of-system," look for the minimum necessary and sufficient immediate cause or causes. A "state-of-system" fault event may require an and-gate, an or-gate, and inhibit-gate, or possibly no gate at all. As a general rule, when energy originates from a point outside the component, the event may be classified as "state-of-system."
- No Miracles Rule: If the normal functioning of a component propagates a fault sequence, then it is assumed that the component functions normally.
- Complete-the-Gate Rule: All inputs to a particular gate should be completely defined before further analysis of any one of them is undertaken.[1]
- No Gate-to-Gate Rule: Gate inputs should be properly defined fault trees, and gates should not be connected directly to other gates.

The phrase "immediate, necessary, and sufficient" from Ground Rule II requires some clarification. We must clearly describe the current state of the system. Example: a computer is powered through an uninterruptible power supply (UPS) that holds 20 minutes of reserve power. If AC power is lost and does not recover before the UPS battery drains, the device fails, and the failure event is written as "Loss of AC power," not "Loss of power for more than 20 minutes." The *immediate cause* of the event is loss of power. The loss of power is *sufficient* to cause the event. If the computer has multiple redundant power supplies, an immediate cause of failure would still be "Loss of AC power,"; when that event is expanded, the *necessary* and *immediate* cause is "Loss of power from UPS 1 and 2."

[1] More commonly known as a breadth-first (*vs.* depth-first) expansion.

Some analysts have compiled an additional list of heuristics to simplify fault tree construction [19].

- Replace an abstract event by a less abstract event.
- Classify an event into more elementary events.
- Identify distinct causes for an event.
- Couple trigger event with "no protective action."
- Find cooperative causes for an event.
- Pinpoint a component failure event.

We start by expanding the top event E1 "Local MTA fails to accept mail." E1 is a *state-of-system* fault so we look for its necessary, immediate, and sufficient causes. We define event E2 as "Router unavailable" and E3 as "Inbound mail service unavailable" and combine them into and or-gate. Our tree is now E1 = (E2 or E3). See Figure 7.

Note transfer gates 1 and 2, leading to Figures 8 and 9, respectively. This leads to a bit of jumping between figures as we follow the Complete-the-Gate Rule.

Event E2 "Router unavailable" is a *state-of-system* fault so we again look for necessary, immediate, and sufficient causes. We expand E2 into two events, E4 "Router failed" and E5 "Router OOS[2] for maintenance." Note we are modeling human action and operations procedure as well as random failure. See Figure 8.

Event E3 "Inbound mail service unavailable" is a *state-of-system* fault; we model this as E6 "No MTA available" and E7 "Common-cause failure of all MTAs" both feeding into an or-gate. Common-cause failure is a special class of failure which will be discussed later; this leads from our assumption of component independence. Note that E7 is marked with a small diamond to indicate an undeveloped event. See Figure 9.

Returning to Figure 8, we see that event E4 "Router failed" is a *state-of-component* fault so we expand it into primary, secondary, and command faults all feeding into an or-gate. We find two primary faults E8 "Router hardware failure" and E9 "Router software failure," two secondary faults E10 "Router overloaded" and E11 "Router loses AC power," and one command fault E12 "Router misconfigured."

We mark E8 and E9 as basic events and events E10, E11, and E12 as undeveloped events. The distinction is somewhat arbitrary since we can choose to expand any of these events later. In this case, we decide that we may want to create more detailed

[2] OOS: Out Of Service

models of power failure, router misconfiguration, and router overload later and that we're satisfied with the rather gross categorizations of router hardware and software failure. Also, the router hardware and software are determined by the router vendor; we ostensibly control AC power, network traffic, and router configuration. Regardless of our categorization of events as basic or undeveloped, we have finished modeling intermediate event E2 as a combination of primary events.

Event E6 "No MTA available" is a *state-of-system* fault. We model E6 as E13 "MTA #1 failed" and E14 "MTA #2 failed" both feeding into an and-gate. Note that if we added a third identical MTA, we could simply duplicate the tree structure beneath E13 under a new event "MTA #3 failed." The common-cause failure event E7 should not change. Caution must be used when copying parts of the tree to ensure that nothing has been missed. The value of the Complete-the-Gate Rule is obvious here.

Following transfer gate 3 from Figure 9 to Figure 10, we can now expand event E13 "MTA #1 unavailable." We can see by our physical diagram and our assumption that the mail servers are identical that the expansion of events E13 and E14 will be similar. E13 is similar to E2, in that E13 is a *state-of-system* fault, composed of a "system failed" and a "system OOS" event combined into an or-gate (events E15 "MTA #1 failed" and E16 "MTA #1 OOS for maintenance.") Similarly, we see event E14 "MTA #2 unavailable" expands into E17 "MTA #2 failed" and E18 "MTA #2 OOS for maintenance," also combined into an or-gate in Figure 11.

Event E15 "MTA #1 failed" is a *state-of-component* fault so we expand it into primary, secondary, and command faults all feeding into an or-gate. We find two primary faults, E19 "MTA #1 hardware failure" and E20 "MTA #1 software failure," two secondary faults, E21 "MTA #1 out of resources" and E22 "MTA #1 loses AC power," and one command fault E23 "MTA #1 misconfigured." This is similar in structure to E4 "Router failed" but we've used "system out of resources" instead of the more specific "system overloaded" to model events where the mail server software does not have the resources to accept all the mail it's receiving (i.e., there's too much incoming mail or there aren't adequate resources available.) At some point, we may wish to specify resources and the effects of their depletion but in this simple model we will not expand these events further.

Similarly, Event E17 "MTA #2 failed" expands into two primary faults, E24 "MTA #2 hardware failure" and E25 "MTA #2 software failure," two secondary faults, E26 "MTA #2 out of resources" and E27 "MTA #2 loses AC power," and one command fault E28 "MTA #2 misconfigured." Our model is complete now that all the leaf nodes of the tree are primary events (either basic or undeveloped events.) We can now generate the cutsets for this tree.

Discussion of Tree Construction

Events E11, E22, and E27 all involve the loss of AC power and may be considered identical according to our assumption that all devices are on the same electrical supply. In a more detailed analysis, one would probably break out electric power into its own fault tree, analyzing off-site power, backup generators, uninterruptible power supplies (UPS), switchgear, and power distribution units.

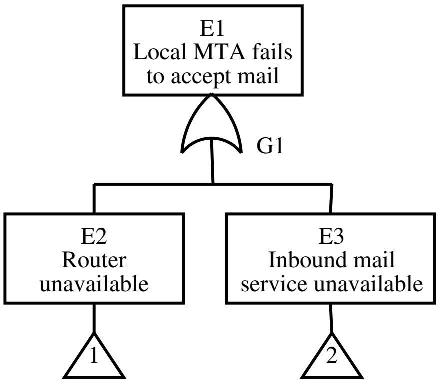

Figure 7: MX1 tree 1 (of 5).

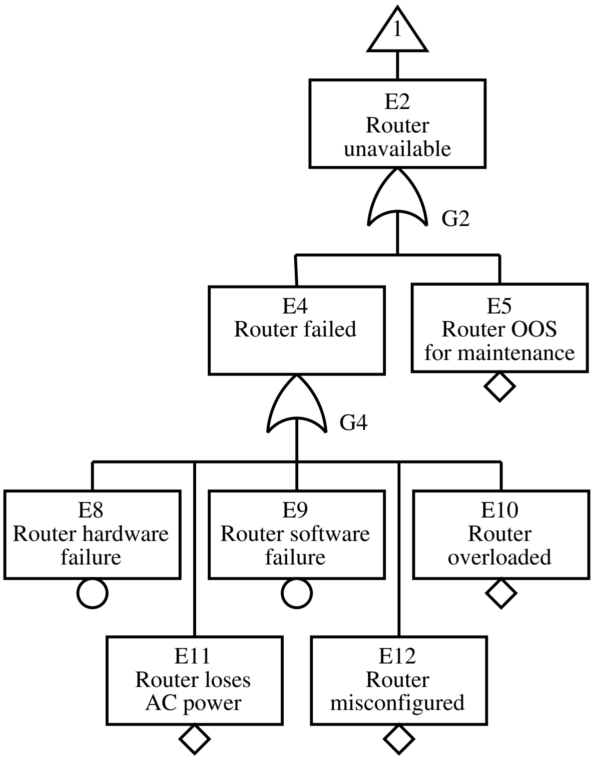

Figure 8: MX1 tree 2 (of 5).

Finding Minimal Cutsets

Having developed a fault tree for MX1, we now can find the minimal cutsets of this tree. In a more complex analysis, one would use a computer code to find the minimal cutsets. In this case we will manually derive them from logical expressions.

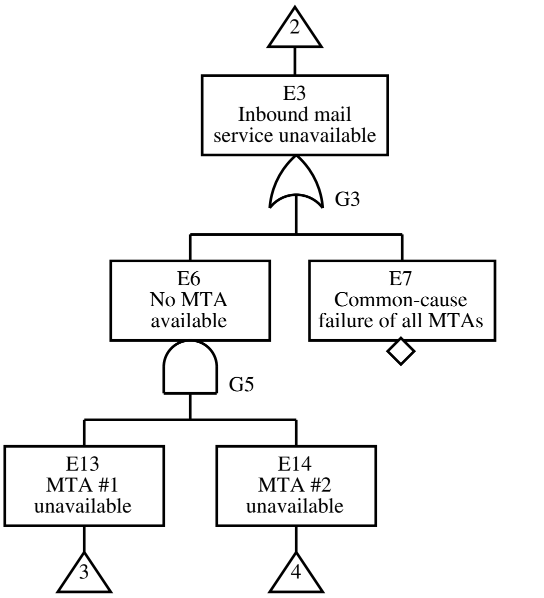

Figure 9: MX1 tree 3 (of 5).

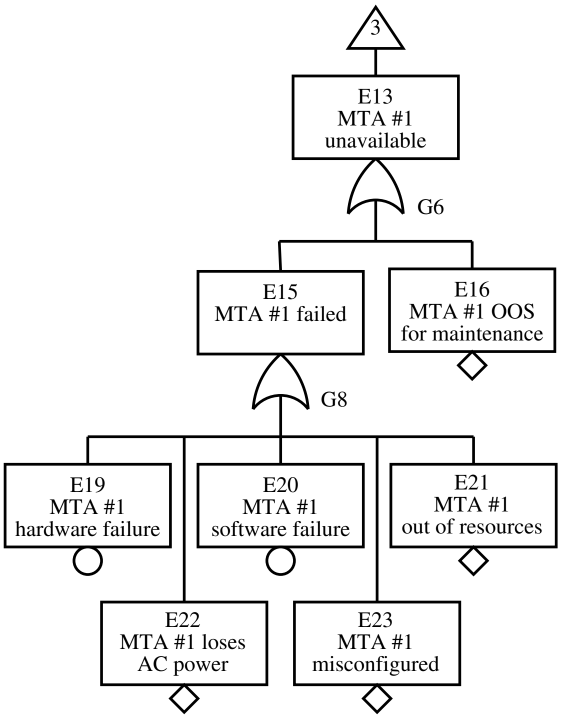

Figure 10: MX1 tree 4 (of 5).

The MX1 fault tree reduces to the following set of equations, where E_i represents the ith intermediate event, e_i represents the ith primary (basic or undeveloped) event and i is the event number in the fault tree, e.g., e_5 represents event E5 "Router OOS for maintenance."

$$
\begin{aligned}
E_1 &= or(E_2, E_3) \\
E_2 &= or(E_4, e_5) \\
E_3 &= or(E_6, e_7) \\
E_4 &= or(e_8, e_9, e_{10}, e_{11}, e_{12}) \\
E_6 &= and(E_{13}, E_{14}) \\
E_{13} &= or(E_{15}, e_{16}) \\
E_{14} &= or(E_{17}, e_{18}) \\
E_{15} &= or(e_{19}, e_{20}, e_{21}, e_{22}, e_{23}) \\
E_{17} &= or(e_{24}, e_{25}, e_{26}, e_{27}, e_{28})
\end{aligned}
$$

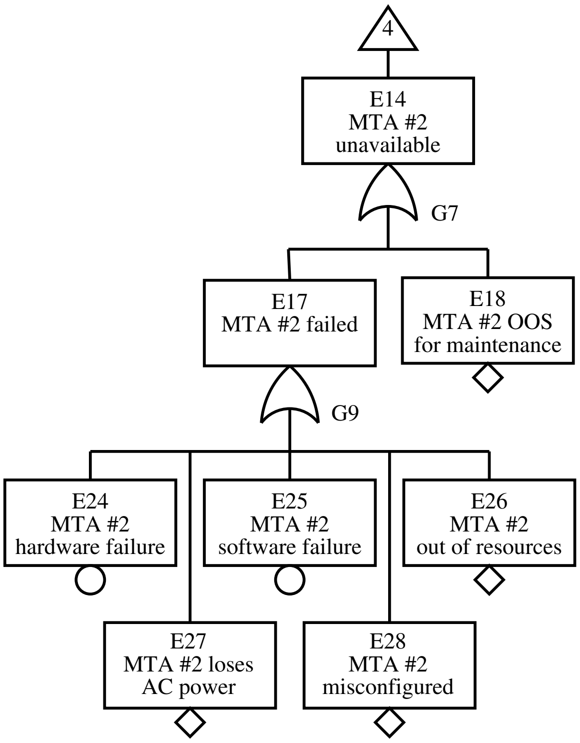

Figure 11: MX1 tree 5 (of 5).

We write top level event E_1 in terms of primary events, using $and(event_1, \ldots, event_i)$ and $or(event_1, \ldots, event_i)$ to represent the logical operations and and or.

$$
\begin{aligned}
E_1 &= or(E_2, E_3) \\
&= or(or(E_4, e_5), or(E_6, e_7)) \\
&= or(or(e_8, e_9, e_{10}, e_{11}, e_{12}), e_5, and(E_{13}, E_{14}), e_7) \\
&= or(e_5, e_7, e_8, e_9, e_{10}, e_{11}, e_{12}, \\
&\qquad and(or(E_{15}, e_{16}), or(E_{17}, e_{18}))
\end{aligned}
$$

$$
\begin{aligned}
E_1 = {} & or(e_5, e_7, e_8, e_9, e_{10}, e_{11}, e_{12}, \\
& and(or(e_{16}, e_{19}, e_{20}, e_{21}, e_{22}, e_{23}), \\
& or(e_{18}, e_{24}, e_{25}, e_{26}, e_{27}, e_{28}))
\end{aligned}
$$

Note that $and(or(A, B), or(C, D)) = or(and(A, C), and(A, D), and(B, C), and(B, D))$. In set notation, E_1 in the following cutsets (note: $e_1 e_2 = and(e_1, e_2)$):

$$
\begin{aligned}
E_1 = {} & or(e_5, e_7, e_8, e_9, e_{10}, e_{11}, e_{12}, \\
& e_{16}e_{18}, e_{16}e_{24}, e_{16}e_{25}, e_{16}e_{26}, e_{16}e_{27}, e_{16}e_{28} \\
& e_{19}e_{18}, e_{19}e_{24}, e_{19}e_{25}, e_{19}e_{26}, e_{19}e_{27}, e_{19}e_{28} \\
& e_{20}e_{18}, e_{20}e_{24}, e_{20}e_{25}, e_{20}e_{26}, e_{20}e_{27}, e_{20}e_{28} \\
& e_{21}e_{18}, e_{21}e_{24}, e_{21}e_{25}, e_{21}e_{26}, e_{21}e_{27}, e_{21}e_{28} \\
& e_{22}e_{18}, e_{22}e_{24}, e_{22}e_{25}, e_{22}e_{26}, e_{22}e_{27}, e_{22}e_{28} \\
& e_{23}e_{18}, e_{23}e_{24}, e_{23}e_{25}, e_{23}e_{26}, e_{23}e_{27}, e_{23}e_{28})
\end{aligned}
$$

Each term in the *or*() clause is a cutset implied by the top event E1. This is neither the set of all cutsets nor the set of minimal cutsets. To generate the set of minimal cutsets we need to remove all cutsets that contain other cutsets. Our assumption about a common power supply allows us to consider events E11, E22, and E27 to be identical. This reduces E_1 to:

$$\begin{aligned} E_1 \;=\; & or(e_5, e_7, e_8, e_9, e_{10}, e_{11}, e_{12}, \\ & e_{16}e_{18}, e_{16}e_{24}, e_{16}e_{25}, e_{16}e_{26}, e_{16}e_{28} \\ & e_{19}e_{18}, e_{19}e_{24}, e_{19}e_{25}, e_{19}e_{26}, e_{19}e_{28} \\ & e_{20}e_{18}, e_{20}e_{24}, e_{20}e_{25}, e_{20}e_{26}, e_{20}e_{28} \\ & e_{21}e_{18}, e_{21}e_{24}, e_{21}e_{25}, e_{21}e_{26}, e_{21}e_{28} \\ & e_{23}e_{18}, e_{23}e_{24}, e_{23}e_{25}, e_{23}e_{26}, e_{23}e_{28}) \end{aligned}$$

This is the set of minimal cutsets for tree MX1.

Discussion of Cutsets

We've identified seven single points of failure (SPOF) and 25 two-event cutsets. Six SPOFs relate specifically to the router and one to the electrical power system. The two-event cutsets are combinations of events that fail both mail servers.

While simple inspection of Figure 6 shows the router as a SPOF, our systematic analysis picks out specific faults (hardware, software, configuration) which we might use to drive policy decisions. For example, we may have stricter controls on router configuration than on mail server configuration. We might also require that any mail server configuration changes be applied only to the primary server and only be applied to the secondary server after some period of live testing or 'burn-in' to reduce the risk of common-cause failures due to server misconfiguration. Usually configuration changes are made to increase functionality, security, or reliability so we must weigh the benefits of standardization with the risks of increased common-cause failure. There is no easy answer to this policy question, though fault tree analysis has helped to make this question more apparent. Although we have no quantitative measure of risk at this point in the analysis, we have at least enumerated the risks to the system within the bounds of our analysis.

Note also that we consider unavailability due to planned maintenance activities. The cutset $e_{16}e_{18}$ ("MTA #1 OOS for maintenance," "MTA #2 OOS for maintenance") may be prohibited by procedure; if it isn't, it should be. One must balance the benefits of periodic scheduled maintenance with the risk posed by these activities. Also, it is inappropriate to discard this cutset simply because the combination of events is forbidden by procedure. A proper analysis will consider human error and failure to follow procedure. Errors of omission (not taking the appropriate action) and errors of commission (taking an inappropriate action, or performing the appropriate action at the wrong place or time) and other facets of human reliability analysis (HRA) are beyond the scope of this paper but must be mentioned for completeness. Depending on the thoroughness of the analysis, one may need to evaluate procedures, operational documentation, and user interfaces for potential human reliability pitfalls.

Common-cause failure

While we often assume that primary (component) failures are independent of each other, this is not always the case. Sometimes a system may fail from multiple basic failures attributable to a common root cause. Two examples: 1) components share a common source of electric power, and 2) a set of components produced by a given manufacturer contain an endemic flaw. We generally cannot find common cause failures just by evaluating the fault tree. We must investigate minimal cutsets individually. It is helpful to compare each cutset to a list of common cause categories such as:

- Manufacturer
- Location
- Regional environmental conditions such as susceptibility to flood, seismic activity, tornados, and ice storms
- Local environmental conditions such as temperature, vibration, humidity, dirt, dust, smoke, fire, and R/F interference
- Human interactions (users and operators)
- Degradation due to test or maintenance activities

For each component, we list applicable characteristics in each category, then group components according to similar characteristics. We then identify each minimal cutset susceptible to common cause failures in each group of components and judge if it warrants further analysis. For example, if the router and both mail servers were located in the same cabinet in a data center, they are susceptible to failures stemming from cabinet wiring faults, physical shock or damage to the cabinet, loss of cooling, mistaken identity (i.e., MTA #1 is mistaken for MTA #2, especially if a cryptic machine naming convention is used or the machines are physically similar.) Common-cause analysis is time-consuming and difficult, though [19] provides a more systematic process for identifying common-cause failures.

Obtaining Failure Rate Data

One can estimate failure rates by reviewing operator shift reports, monitoring system logs, using vendor-supplied MTBF estimates, or using engineering judgment. It is important to use measured, installation-specific data whenever practical. Often generic data and engineering judgment are used until one obtains enough information to build a local reliability database.

Note that this sort of data analysis is a fairly involved topic on its own. Probability distribution modeling occupies an entire chapter in the Fault Tree Handbook, and features prominently in [7, 19, 20, and [9].

Estimating Availability

For the purposes of our example, we will make some reasonable assumptions about system operation and use them to estimate component availability. Assume the mail servers operate continuously except for a one hour outage every three months to apply kernel patches, a three hour outage every year to upgrade the mail server software, and 10 minutes of unavailability a month to update configuration files. The probability of a mail server being unavailable due to maintenance is

$$\begin{aligned}
t^{MTA}_{maint} &= 4\ [events/yr] \cdot 60\ [min/event] \\
&\quad + 1\ [events/yr] \cdot 180\ [min/event] \\
&\quad + 12\ [events/yr] \cdot 10\ [min/event] \\
&= 540\ [min/year] \\
t^{MTA}_{demanded} &= 60\ [min/hr] \cdot 24\ [hr/day] \cdot 365\ [day/yr] \\
&= 525600\ [min/yr] \\
P_{maint}(MTA) &= \frac{t^{MTA}_{maint}}{t^{MTA}_{demanded}} \\
&= \frac{540}{525600} \\
&= 1.03 \times 10^{-3} \ .
\end{aligned}$$

For the router, we assume an annual three hour outage to upgrade firmware and a quarterly five minute outage for configuration changes, giving an “OOS for maintenance” probability of

$$\begin{aligned}
t^{router}_{maint} &= 1\ [events/yr] \cdot 180\ [min/event] \\
&\quad + 4\ [events/yr] \cdot 5\ [min/event] \\
&= 200\ [min/yr] \\
t^{router}_{demanded} &= 525600\ [min/yr] \\
P_{maint}(router) &= \frac{t^{router}_{maint}}{t^{router}_{demanded}} \\
&= \frac{200}{525600} \\
&= 3.81 \times 10^{-4} \ .
\end{aligned}$$

Performance Metrics and Observed Failure Data

Estimated failure probabilities are useful for scoping studies or design-phase analyses but they are no substitute for observed, site-specific data. Much work has been done in the field of failure rate estimation; references [19, 20] provide a good introduction to common data analysis techniques.

Components rarely have a constant failure rate as shown in the “OOS for maintenance” analysis above. Failure rate generally varies with time. For physical components, the plot of failure rate versus time often takes on a characteristic “bathtub” shape. Three regions of interest are shown in Figure 12 – Region I is known as the “infant-mortality” or “burn-in” region, Region II is a region of nearly constant failure rate, and Region III is the “wear-out” region in which the failure rate increases as components degrade due to wear.

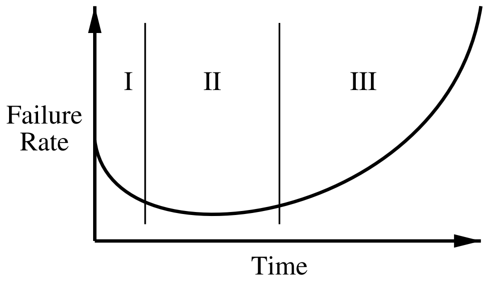

Figure 12: Failure rate variation with time (“Bathtub Curve”).

When modeling component failure rates, we must consider the type of component (software, hardware, mechanical, electrical, solid-state), operation characteristics (used continuously or on-demand), maintenance frequency, test frequency and severity,[3] and operating environment. These factors influence the choice of probability distribution used to model the failure rate.

Advanced Models

Calculating the failure rate of repairable systems can be quite complex, especially systems with redundant backups and multiple operation modes. These situations lead to state machine models requiring continuous Markov analysis for solution. It is not uncommon for analysts to conservatively assume systems cannot be repaired just to simplify the analysis.

Quantifying the Model

A Simplified Model

Let us only consider hardware failures, common cause failure, and maintenance activity in our model, e.g., the probability of all other basic events is zero (alternately, we may treat them as external or “house” events and set them to false.)[4] Then our model reduces to

$$\begin{aligned}
E_1 = {} & or(e_5, e_7, e_8, and(e_{16}, e_{18}), and(e_{16}, e_{24}), \\
& and(e_{19}, e_{18}), and(e_{19}, e_{24}))
\end{aligned}$$

where

	Event	Probability
e_5	Router OOS for maintenance	3.81×10^{-4}
e_7	Common cause failure of both MTAs	1.00×10^{-5}
e_8	Router hardware failure	1.00×10^{-4}
e_{16}	MTA1 OOS for maintenance	1.03×10^{-3}
e_{18}	MTA2 OOS for maintenance	1.03×10^{-3}
e_{19}	MTA1 hardware failure	2.0×10^{-2}
e_{24}	MTA2 hardware failure	2.0×10^{-2}

Probabilities of events e_7, e_8, e_{19}, and e_{24} are estimated using engineering judgment. With time, we

[3]This is especially important for backup diesel generators. Rapid start times and frequent testing cause substantial maintenance problems. Ironically, the tests designed to assure availability may tend to reduce it.

[4]There is a distinction between events which don’t occur (set to false) and events that occur with zero probability.

would replace probabilities of events e_8, e_{19}, and e_{24} with observed data and perform additional common-cause analysis to estimate the probability of event e_7.

Probability Theory

Quantifying a fault tree is different than quantifying an event tree since the logical combination of probabilities requires some knowledge of basic set theory and Boolean algebra. Readers unfamiliar with these topics are encouraged to review a reference (any of [2, 19, 20, 21, 5, 6]) for more information.

Before converting a logical model into a probabilistic model, we must understand how to model individual logic gates in terms of probability.

The or-*gate*

The or-gate represents the union of the events attached to the gate. If events A and B are inputs of an or-gate and event Q is the output, that is, $Q = or(A, B)$, then the probability of event Q (i.e., $Pr(Q)$) is given by

$$\begin{aligned} Pr(Q) &= Pr(A) + Pr(B) - Pr(A \bigcap B) \\ &= Pr(A) + Pr(B) - Pr(A)Pr(B|A) \\ &= Pr(A) + Pr(B) - Pr(B)Pr(A|B) \end{aligned}$$

where $Pr(B|A)$ is the conditional probability of B occurring, given that A has occurred.

Some important results from probability and set theory:

- If A and B are mutually exclusive events then $Pr(A \bigcap B) = 0$ and $Pr(Q) = Pr(A) + Pr(B)$
- If A and B are independent events then $Pr(B|A) = Pr(B)$ and $Pr(Q) = Pr(A) + Pr(B) - Pr(A)Pr(B)$
- If event B is completely dependent on event A then $Pr(B|A) = 1$ and $Pr(Q) = Pr(B)$
- In all cases, one may conservatively estimate $Pr(Q) \cong Pr(A) + Pr(B) \geq Pr(A) + Pr(B) - Pr(A \bigcap B)$. That is, any error introduced by neglecting $Pr(A \bigcap B)$ increases $Pr(Q)$ and is therefore conservative.
- For small values of $Pr(A)$ and $Pr(B)$, say < 0.1, $Pr(A \bigcap B)$ is small compared to $Pr(A) + Pr(B)$ so there is little error in estimating $Pr(Q) \cong Pr(A) + Pr(B)$, provided A and B are independent. This is known as the *rare event approximation*.

The exclusive-or-*gate*

If events A and B are inputs into an exclusive-or-gate and event Q is the output, that is, $Q = xor(A, B)$, then $Pr(Q)$ is given by

$$Pr(Q) = Pr(A) + Pr(B) - 2Pr(A \bigcap B)$$

If we compare the numerical probability results for the or-gate with those for the exclusive-or-gate, we see that the difference is negligible if A and B are independent. In all cases, treating exclusive-or-gates as standard (inclusive) or-gates is conservative. For this reason, exclusive-or-gates are rarely seen in fault trees.

The and-*gate*

The and-gate represents the intersection of the events attached to the gate. If events A and B are inputs into an and-gate and event Q is the output, that is, $Q = and(A, B)$, then $Pr(Q)$ is given by

$$\begin{aligned} Pr(Q) &= Pr(A)Pr(B|A) \\ &= Pr(B)Pr(A|B) \end{aligned}$$

where $Pr(B|A)$ is the conditional probability of B occurring, given that A occurred. From probability theory, we find:

- If A and B are independent events then $Pr(B|A) = Pr(B)$, $Pr(A|B) = Pr(A)$ and $Pr(Q) = Pr(A)Pr(B)$
- If A and B are not independent, $P(Q)$ may be much greater than $Pr(A)Pr(B)$, though no greater than the larger of $Pr(A)$ or $Pr(B)$.
- If B is completely dependent on event A then $Pr(B|A) = 1$ and $Pr(Q) = Pr(A)$

Quantification

Combining the model and the basic event probabilities, the failure probability of the system is

$$\begin{aligned} E_1 &= or(e_5, e_7, e_8, and(e_{16}, e_{18}), and(e_{16}, e_{24}), \\ &\quad and(e_{19}, e_{18}), and(e_{19}, e_{24})) \end{aligned}$$

$$\begin{aligned} Pr(E_1) &\cong Pr(e_5) + Pr(e_7) + Pr(e_8) \\ &\quad + Pr(e_{16})Pr(e_{18}) + Pr(e_{16})Pr(e_{24}) \\ &\quad + Pr(e_{19})Pr(e_{18}) + Pr(e_{19})Pr(e_{24}) \\ &\cong 3.81 \times 10^{-4} + 1.00 \times 10^{-5} + 1.00 \times 10^{-4} \\ &\quad + (1.03 \times 10^{-3} \cdot 1.03 \times 10^{-3}) \\ &\quad + (1.03 \times 10^{-3} \cdot 2.00 \times 10^{-2}) \\ &\quad + (2.00 \times 10^{-2} \cdot 1.03 \times 10^{-3}) \\ &\quad + (2.00 \times 10^{-2} \cdot 2.00 \times 10^{-2}) \\ &\cong 9.33 \times 10^{-4} \end{aligned}$$

This result assumes all events are independent and uses the failure probabilities listed at the beginning of this section. A number of simplifying assumptions are made, chiefly the rare event approximation. It is left for the reader to derive the full analytical expression for $Pr(E_1)$ and to compare the true numerical value to the result approximated here (note that the full expression for $Pr(E_1)$ has over 120 terms.) Here we see the value of using a computer code to evaluate and quantify fault trees. While this technique may be performed manually, the calculations for even a simple model quickly become tedious.

A final note on quantification: when combining fault trees and event trees, be sure to combine cutsets and eliminate non-minimal cutsets for each event tree end state before quantifying. Fault trees should be as independent as possible but need not be completely independent, provided that redundant and impossible cutsets are removed before generating numerical results.

Importance Ranking

Now that we have determined the minimal cutsets and have quantified the tree, we can quantitatively assess the importance of each component.

Popular importance measures are Birnbaum and Fussell-Vessely (named after their inventors), risk achievement worth (RAW) and risk reduction worth (RRW.) Birnbaum importance measures the sensitivity of risk to changes in component reliability over the entire range of reliability – from "component always failed" to "component always available." RAW is the fractional increase in risk assuming a particular event always occurs and RRW is the fractional decrease in risk assuming a particular event never occurs.

Using the notation above, Birnbaum importance of event e_j to top event T is given by

$$I_B^j(T) = Pr(T, Pr(e_j) = 1) - Pr(T, Pr(e_j) = 0)$$

We calculate the Birnbaum importance for router hardware failure e_8 as

$$\begin{aligned} I_B^8 &= Pr(E_1, Pr(e_8) = 1) - Pr(E_1, Pr(e_8) = 0) \\ &= 1.000923 - 9.23 \times 10^{-4} \\ &= 1 \end{aligned}$$

And for MTA1 maintenance outages, we calculate Birnbaum importance as

$$\begin{aligned} I_B^{16} &= Pr(E_1, Pr(e_{16}) = 1) - P(E_1, Pr(e_{16}) = 0) \\ &= (2.19 \times 10^{-2} - 9.12 \times 10^{-4}) \\ &= 2.10 \times 10^{-2} \end{aligned}$$

Qualitatively, Birnbaum importance tells us that risk is much more sensitive to changes in router reliability, less so to changes in mail server reliability. Note that since router reliability is already fairly high, Birnbaum importance tells us that system reliability will fall faster with a decrease in router reliability than with a decrease in mail server reliability.

The Fussell-Vessely importance of a component is the sum of the probability of all event sequences (cutsets) containing that component divided by the total risk, i.e., the fraction of total risk related to this component. For the router

$$\begin{aligned} I_{GV}^{router}(E_1) &= \frac{Pr(e_5) + Pr(e_8)}{Pr(E_1)} \\ &= \frac{3.81 \times 10^{-4} + 1.00 \times 10^{-4}}{9.33 \times 10^{-4}} \\ &= 0.516 \end{aligned}$$

and for MTA1

$$\begin{aligned} I_{GV}^{MTA1}(E_1) &= \frac{Pr(e_7) + Pr(e_{16}e_{18}) + Pr(e_{16}e_{24})}{Pr(E_1)} \\ &\quad + \frac{Pr(e_{19}e_{18}) + Pr(e_{19}e_{24})}{Pr(E_1)} \\ &= \frac{1.00 \times 10^{-5} + 1.06 \times 10^{-6}}{9.33 \times 10^{-4}} \\ &\quad + \frac{2.06 \times 10^{-5} + 2.06 \times 10^{-5}}{9.33 \times 10^{-4}} \\ &\quad + \frac{4.00 \times 10^{-4}}{9.33 \times 10^{-4}} \\ &= 0.485 \end{aligned}$$

In this case Fussell-Vessely importance shows router failure contributes slightly more to overall risk than MTA1 failure. The router's high reliability makes up for the lack of a redundant backup router.

Risk achievement worth and risk reduction worth are calculated for each basic event and a component's importance is taken as the maximum of the RAW or RRW for the basic events associated with a component.

$$\begin{aligned} RAW_i &= \frac{Pr(T, Pr(e_i) = 1)}{Pr(T)} \\ RRW_i &= \frac{Pr(T)}{Pr(T, Pr(e_i) = 0)} \end{aligned}$$

These metrics are related to Birnbaum and Fussell-Vessely importance by

$$\begin{aligned} I_B^j &= \left(RAW_j - \frac{1}{RRW_j}\right) Pr(T) \\ I_{FV}^j &= \left(1 - \frac{1}{RRW_j}\right) \end{aligned}$$

Based on the values above, we find

$$\begin{aligned} RAW_5 &= 1072.10 \\ RAW_8 &= 1072.41 \\ RAW_{16} &= 23.51 \\ RAW_{19} &= 23.08 \end{aligned}$$

and

$$\begin{aligned} RRW_5 &= 1.69 \\ RRW_8 &= 1.12 \\ RRW_{16} &= 1.02 \\ RRW_{19} &= 1.82 \end{aligned}$$

Qualitatively, the RAW results show that a decrease in router reliability will affect the system much more than a proportional decrease in MTA1 reliability. The RRW results show that increasing MTA1 hardware reliability is slightly more effective at reducing risk as a proportional reduction in router unavailability due to maintenance. Both RAW and RRW metrics add additional meaning to component reliability trends.

Compare these metrics to an ad hoc analysis which could either claim that the router is most important because it's a single point of failure (SPOF) or that mail servers are most important because of their much higher failure rate. Note that the ad hoc analysis breaks down rapidly as the complexity of the system increases. In this simple case it's not so apparent but if we made the system more complex by adding more mail servers and a standby router, the ad hoc analysis becomes less useful, approaching mere speculation. This is especially true when all SPOFs are found and eliminated.

Regardless of the figure-of-merit used, one must understand what it represents and calculate it consistently. When using FTA as a risk communication tool, it helps to use simple, intuitive importance measures.

Expanding The Fault Tree Model

This model is trivial though not totally contrived. One benefit of explicitly stating our assumptions at the outset is that we can revisit them individually and systematically to expand our model. Some issues not addressed in this simple analysis include:

- human error including errors of omission and errors of commission
- component dependencies (e.g., if MTA1 fails, will the increased load on MTA2 increase MTA2's failure probability? If we patch machines less often to reduce maintenance unavailability, will the probability of software failure increase?)
- recovery actions
- sensitivity and uncertainty analysis, showing how the model reacts to statistical variations and uncertainty in component reliability.

Of these, human reliability is the most important since human error often dominates risk in high-reliability systems. However, the field of human reliability analysis (HRA) is extremely complex, far beyond the scope of this paper.

Comparison between ETA and FTA

Event tree analysis appears simplistic, even obvious. The technique is important for this very reason – event trees clearly communicate failure modes (consequences) and the event sequences that lead to them. ETA provides a straightforward, consistent, systematic approach to modeling complex systems at a high level. This high-level approach provides a basis for more detailed modeling with fault trees.

Fault trees do not model degraded performance well; since they are built on Boolean operations such and and or, fault trees are best at yielding binary results (e.g., success/failure.) Event trees aren't limited to binary outcomes and can show a variety of consequences.

Event trees serve another purpose; they are often used to break up a large analysis into smaller, more manageable parts, simplifying construction and review. And unlike fault trees, event trees clearly show consequences; fault trees are more useful for showing the existence and probability of failure sequences. Fault trees are usually far more detailed than event trees, modeling low-level component failure and human action. This allows components to be ranked according to their contribution to overall risk. Finally, undeveloped events in fault trees explicitly show the limits of the analysis. Both ETA and FTA have their strengths and weaknesses but the combination of the two provides balance and results in a powerful analytical technique.

Limitations of Probabilistic Risk Analysis

PRA requires skilled analysts, a thorough understanding of the systems to modeled, observed or estimated reliability data, and fault tree analysis software. For many systems, the cost of analysis is excessive. One is cautioned against putting too much faith in absolute failure probabilities; the quantitative measures are most effectively used as relative measures of risk (i.e., is component X more important than component Y, or which sequence of events is most likely?)

Both the aerospace and nuclear industries both have a very strong configuration management (CM) culture while formal configuration management is almost nonexistent in the computer industry. This presents us with a serious though not insurmountable challenge – how can we have faith in a model when the system the model based on is so easily changeable?

We can categorize system changes as topological changes and status changes. Topological changes are changes in which components are added, deleted, or rearranged; for example, adding a new web server or moving machines to a new switch or network. These changes require modifying the structure of a fault tree. Status changes are changes to the failure probabilities based on the observed or postulated condition of the system – these changes only modify event probabilities, they do not affect the model's structure.

Since computer systems can produce copious amounts of diagnostic data, I believe one could compensate for status changes by substituting monitoring for strict CM. Also, by decomposing fault trees into modules representing generic components and by mapping network topology [4] and services, we may be able to compensate for topological changes as well.

Note that monitoring, CM, and PRA are complementary approaches. CM is costly and never perfect; monitoring can show where CM is failing and PRA can show where strict CM is warranted (or wasted.)

Finally, PRA does not handle time-dependent failure well. Skilled reviewers are required to assure fault tree completeness. Also, estimating the reliability of redundant, repairable components can become quite complex, leading to state machines that require sophisticated mathematics[5] for solution.

Suggestions for Future Work

Analysis software

Most PRA software is archaic or proprietary and based on my cursory searches, I have found no modern PRA code that runs under anything but DOS or some form of Microsoft Windows [8]. To reach a wider audience of system administrators, we need a freely-available fault tree analysis code, preferably released under a license that allows the source code to be reviewed, modified, and redistributed. The code should support standard databases (MySql, Oracle,

[5]Laplace transforms, discrete and continuous Markov analysis, and other techniques usually forgotten immediately after passing ones final exams, based on the author's experience.

Postgres, etc.) and produce reports and images in common, portable formats.

Generic Model and Failure Rate Data Repository

Unlike the chemical, nuclear, or aerospace industries, the computer industry relies almost completely on commodity components. A repository of generic fault models and failure rate data would simplify fault tree construction and quantification. While generic models and data are no substitute for a careful site-specific analysis, they would speed learning and would help analysts build the framework for a site-specific analysis.

Integrate Monitoring and Management Tools with FTA Software

Provided PRA analysis software and generic models and data were readily available, the next step would be to link common monitoring and management software with a site-specific failure rate database. One could use network analysis tools to mechanically generate system models or confirm the accuracy of existing models. Integration with monitoring and management tools would allow near-realtime risk profiling of systems, similar to refueling outage risk management software used today in the U. S. nuclear industry.

Research Topics

While software fault tree analysis (SFTA) is an active field of research, I know of little system administration research involving PRA.

Security Analysis and Threat Assessment

One can treat security as a subset of reliability. Since PRA was originally developed to tackle the problem of analyzing low-probability high-consequence events, it seems natural to use the technique for security analysis. Bruce Schneier uses fault tree analysis[6] in Chapter 21 of "Secrets and Lies" [3], though he stops short of using fault trees as a probabilistic model. This may be due a lack of publicly-accessible statistics on attacks [7] but I suspect he uses fault trees as a source of data for game-theoretical security models or other cost-risk-benefit analyses, not as probabilistic models.

Some researchers use fault trees to generate the requirements specification for an intrusion detection system [18]. By analyzing protocols and typical implementations, they found potential vulnerabilities and developed their software specification accordingly.

Analyze Common Internet Protocols

Security considerations are often neglected when a protocol is first designed with the intent of adding security features after the protocol has gained public acceptance. Many protocols are easily abusable or may unnecessarily reveal sensitive information to third parties. PRA can be used to analyze common protocols for potential security, privacy, and abuse vulnerabilities.

Investigate Common Operating System Process Management Schemes to Model Process Failure

PRA can be used to analyze common process management schemes to suggest ways of increasing the robustness and reliability of existing operating systems. Once could model permissions, resource requirements, use, and depletion, etc. to estimate and improve process reliability. It may also suggest coding standards to help increase software reliability within particular operating environments.

Develop a Common Risk Assessment Notation or Language

One could add risk assessment notation and techniques to UML (Unified Modeling Language.) New software would be more secure and more reliable if risk assessment were considered part of the design process. UML is often used to describe complex relationships within and among systems leading to its popularity as a design and communication tool. It seems natural to extend UML with risk assessment and management notation.

Beyond ETA/FTA

Cause-consequence analysis and decision table analysis may be even better methods for analyzing computer system reliability than ETA/FTA. Tutorials on CCA and decision tables are less accessible than those for ETA/FTA.

Conclusion

PRA is a powerful technique for analyzing and communicating the reliability of complex systems. Yielding both qualitative and quantitative results, PRA provides a rational basis for decisions and resource allocation in the face of complexity and uncertainty.

Acknowledgments

I would like to thank Jim Robinson and Kristin Epley and everyone in Excite@Home's Product Operations staff for giving me the time and motivation to work on this paper. Special thanks go to Loys Bedell, William Salyer, Vicki Bier, and Alexei Novikov for technical review and support. I am forever in the debt of Marrit Ingman and Susan Pinsonneault for editorial review. Finally, I'd like to thank Mark Burgess and William Annis for their constant motivation and encouragement.

Author Information

Bob Apthorpe is a Senior System Administrator in the Product Operations group of Excite@Home, Inc. He is primarily responsible for content service design, support, and reliability but over the past five years has worked on host and network security, traffic analysis, monitoring, automation, forensic operations, incident response, abuse handling, project management, and documentation.

Prior to joining Excite, he worked as a nuclear safety analyst with Entergy Operations at River Bend

[6] He calls his fault trees "attack trees." His notation is slightly different than mine but his methodology is similar.

Station in St. Francisville, Louisiana. He earned an M.S. and a B.S. in Nuclear Engineering and Engineering Physics from the University of Wisconsin. In his copious free time, he performs with the We Could Be Heroes improvisational comedy troupe in Austin, Texas and practices just a little aikido. He may be reached at arclight@jump.net or apthorpe@excitecorp.com.

References

[1] Leveson, Nancy, ''High-Pressure Steam Engines and Computer Software,'' Presented as a keynote address at the International Conference Software Engineering in Melbourne Australia, http://www.safeware-eng.com/pubs/HiPreStEn.pdf, 1992.

[2] US Nuclear Regulatory Commission, ''Fault Tree Handbook NUREG-0492,'' http://www.nrc.gov/NRC/NUREGS/SR0492/index.html, 1981.

[3] Schneier, Bruce, *Secrets and Lies: Digital Security in a Networked World*, 2000.

[4] Cheswick, Bill, Hal Burch, and Steve Branigan, ''Mapping and Visualizing the Internet,'' *Proceedings of the USENIX Annual 2000 Technical Conference*, June 18-23, 2000.

[5] United States Naval Institute, *Naval Operations Analysis*, p. 249, 1968.

[6] Gonick, Larry, and Woollcott Smith, *The Cartoon Guide to Statistics*, p. 42, 1993.

[7] Moore, David, Geoffrey Voelker, and Stefan Savage. ''Inferring Internet Denial-of-Service Activity,'' To appear in *Proceedings of the 2001 USENIX Security Symposium*, http://www.caida.org/outreach/papers/backscatter/, 2001.

[8] Idaho National Engineering and Environmental Laboratory, ''SAPHIRE – Systems Analysis Programs for Hands-on Integrated Reliability Evaluations,'' http://saphire.inel.gov/, May 23, 2001.

[9] Burgess, Mark, Hårek Haugerud, Sigmund Straumsnes, "Measuring System Normality I: Scales and Characteristics," http://www.iu.hio.no/mark/SystemAdmin/papers/Normal1.pdf, January, 2001.

[10] Burgess, Mark, *On the Theory of System Administration*, http://www.iu.hio.no/mark/SystemAdmin/papers/SysAdmTheory.pdf, March, 2000.

[11] Leveson, N., L. Alfaro, C. Alvarado, M. Brown, E. B. Hunt, M. Jaffe, S. Joslyn, D. Pinnel, J. Reese, J. Samarziya, S. Sandys, A. Shaw, Z. Zabinsky. ''Demonstration of a Safety Analysis on a Complex System,'' Presented at the Software Engineering Laboratory Workshop,'' NASA Goddard, http://www.safeware-eng.com/pubs/DemSafAn.pdf, December, 1997.

[12] Neumann, Peter G., *Computer-Related Risk*s, 1995.

[13] Baker, D. N., J. H. Allen, S. G. Kanekal, and G. D. Reeves, ''Pager Satellite Failure May Have Been Related to Disturbed Space Environment,'' http://www.agu.org/sci_soc/articles/eisbaker.html .

[14] Stevens, W. Richard, *TCP/IP Illustrated, Volume 1: The Protocols*, 1994.

[15] Partridge, Craig, *RFC 974: Mail Routing and the Domain System*, January, 1986.

[16] Postel, Jonathan B., *RFC 821: Simple Mail Transfer Protocol*, August, 1992.

[17] Hazel, Philip, *Exim: The Mail Transfer Agent*, June, 2001.

[18] Helmer, Guy, Johnny Wong, Mark Slagell, Vasant Honavar, Les Miller, and Robyn Lutz, ''A Software Fault Tree Approach to Requirements Analysis of an Intrusion Detection System,'' *Proceedings, Symposium on Requirements Engineering for Information Security*, Indianapolis, IN, http://latte.cs.iastate.edu/ghelmer/SFTA-ID.ps, March, 2001.

[19] Henley, Ernest J., Hiromitsu Kunamoto, *Probabilistic Risk Assessment: Reliability Engineering, Design and Analysis*, 1992.

[20] Billinton, Roy, Ronald N. Allan. *Reliability Evaluation of Engineering Systems: Concepts and Techniques, Second Edition*, 1992.

[21] Grimaldi, Ralph P., *Discrete and Combinatorial Mathematics: An Applied Introduction*, June 1989.