Integrated Fault and Security Management

Ehab Al-Shaer and Yan Chen†
School of Computer Science, DePaul University, Chicago, IL, USA
†Department of Electrical Engineering and Computer Science,
Northwestern University, Evanston, IL, USA
ehab@cs.depaul.edu, ychen@northwestern.edu

1 INTRODUCTION

Network problems such as faults and security attacks are expressed in the network as one or more symptoms (e.g., alarms, logs, troupe tickets). Network problem diagnosis is the process of correlating or analyzing the observed symptoms in order to identify the root cause. As network faults and security attacks might show similar symptoms, it is possible to misidentify faults as security attacks or vise versa. For example, host/network reachablility problems could be due to either a denial of service attack or link or protocol failure. This causes more false alarms and incorrect response actions. Therefore, integrating fault and security management is important for practical network management systems in order to diagnose and fix problems accurately.

Fault and security intrusion diagnosis exhibit a similar reasoning process, which includes symptom collection, correlation and evaluation. This makes the integration of fault and security management even more sensible. However, to achieve an optimal integration, a number of challenges need to be addressed. First, the root cause should be accurately identified even with incomplete symptom information. Second, problem identification should be fast to handle high-speed networks and wide sensor (e.g., IDS) distribution.

In Section 2 of this chapter, we present an active problem diagnosis framework for integrating the reasoning of fault or security alarms within the same engine. The presented framework uses an active diagnosis approach to deal with incomplete symptom information and identify faults and intrusions. In

Section 3, we show an architecture for network-based intrusion detection systems that analyzes traffic collected from different sensors on a high speed network, identifies faults and intrusions, and initiates proper mitigation actions for various intrusions.

2 ACTIVE INTEGRATED FAULT IDENTIFICATION FRAMEWORK

2.1 Background

Fault localization is a basic component in a fault management system because it identifies the fault reason which can best explain the observed network disorders (called symptoms). Examples of fault symptoms include host or network unreachable, slow response, high utilization, *etc.*. Most fault reasoning algorithms use a bipartite directed acylic graph to describe the Symptom-Fault correlation, which represents the causal relationship between each fault f_i and a set of its observed symptoms S_{f_i} [23]. Symptom-Fault causality graph provides a vector of correlation likelihood measure $p(s_i|f_i)$, to bind a fault f_i to a set of its symptoms S_{f_i}. Similarly, Symptom-intrusion causality graph can be constructed based on *attack graphs* [18] to describe the alarm-intrusion correlation. For simplicity, we will focus in the rest of this section on fault reasoning and diagnosis, however, similar techniques are applied to security attacks/intrusions identification.

Two approaches are commonly used in fault reasoning and localization: passive diagnosis ([19], [23], [22], [13] and active probing ([2], [9], [6], [10]). In passive approach, all symptoms are passively collected and then processed to infer the root faults. In active approach, faults are detected by conducting a set of probing actions. Passive approach causes less intrusiveness in management networks. However, it may take long time to discover the root faults, particularly if symptom loss ratio is high. On the other hand, although active probing approach is more efficient to identify faults quickly, probing might cause significant overhead particularly in large-scale networks. In this section, we will show a novel fault localization technique that integrates the advantage of both passive and active monitoring into one framework, called *Active Integrated fault Reasoning* or *AIR*. In our approach, if the passive reasoning is not sufficient to explain the problem, AIR selects optimal probing actions to discover the most critical symptoms that are important to explain the problem but they have been lost or corrupted during passive fault reasoning. Our approach significantly improves the performance of fault localization while

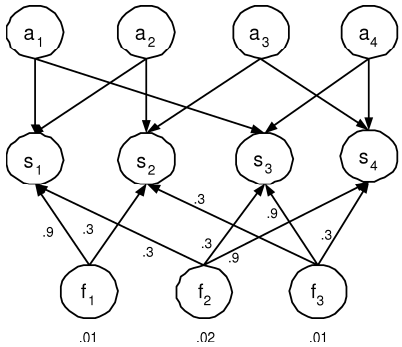

Figure 1: **Action-Symptom-Fault Model**

minimizing the intrusiveness of active fault reasoning.

AIR consists of three modules: Fault Reasoning (FR); Fidelity Evaluation (FE); and Action Selection(AS). *Fault reasoning* module passively analyzes observed symptoms and generates a fault hypothesis. The fault hypothesis is then sent to *fidelity evaluation* module to verify if the fidelity value of the reasoning result is satisfactory. If the correlated symptoms necessary to explain the fault hypothesis are observed (i.e. high fidelity), then fault reasoning process terminates. Otherwise, a list of most likely unobserved symptoms that can contribute to the fault hypothesis fidelity is sent to the *action selection* module, which then performs selected actions to determine which symptoms has occurred but not observed (i.e. lost) and accordingly adjust hypothesis fidelity value. If the new fidelity value is satisfactory, then the reasoning process terminates; otherwise, the new symptom evidence is fed into the *fault reasoning* module to create a new hypothesis. This process is recursively invoked until a highly credible hypothesis is found.

The section is organized as follows. In section 2.2, we discuss our research motivation and the problem formalization. In section 2.3, we describe the components and algorithms of AIR. In Section 2.4, we present a simulation study to evaluate AIR performance and accuracy. In Section 2.5, related work is discussed.

2.2 Challenges and Problem Formalization

In general, active fault management does not scale well when the number of managed nodes or faults in the network grows significantly. In fact, some faults such as an intermittent reachability problem may not even be identified if only active fault management is used. However, this can be easily reported using passive fault management systems because agents are configured to report abnormal system conditions or

Notation	Definition
S_{f_i}	a set of all symptoms caused by the fault f_i
F_{s_i}	a set of all faults that might cause symptom s_i
S_O	a set of all observed symptoms so far
S_{O_i}	a set of *observed* symptoms caused by fault f_i
S_{U_i}	a set of *not-yet-observed* (lost) symptoms caused by the fault f_i
h_i	a set of faults that constitute a possible hypothesis that can explain S_O
Φ	a set of all different fault hypotheses, h_i, that can explain S_O
S_N	a set of correlated but not-yet-observed symptoms associated with any fault in a hypothesis
S_V	a subset of S_N, which includes symptoms that their *existence* is confirmed
S_U	a subset of S_N, which includes symptoms that their *non-existence* is confirmed

Figure 2: **Active Integrated Fault Reasoning Notation**

symptoms such as high average packet drop ratio. On the other hand, symptoms can be lost due to noisy or unreliable communications channels, or they might be corrupted due to spurious (untrue) symptoms generated as a result of malfunctioning agents or devices. This significantly reduces the accuracy and the performance of passive fault localization. Only the integration of active and passive reasoning can provide efficient fault localization solutions.

To incorporate actions into traditional Symptom-Fault model, we propose an extended Symptom-Fault-Action model as shown in Fig. 1. In our model, actions are properly selected probes or test transactions that are used to detect or verify the existence of observable symptoms. Actions can simply include commonly used network utilities, like ping and traceroute; or some proprietary fault management system, like SMRM [6]. We assume that symptoms are verifiable, which means that, if the symptom ever occurred, we could verify the symptom existence by executing some probing actions or checking the system status such as system logs.

In this section, we use $F = \{f_1, f_2, \ldots, f_n\}$ to denote the *fault set*, and $S = \{s_1, s_2, \ldots, s_m\}$ to denote the *symptom set* that can be caused by one or multiple faults in F. Causality matrix $P_{F\times S} = \{p(s_i|f_j)\}$ is used to define causal certainty between fault $f_i(f_i \in F)$ and symptom $s_i(s_i \in S)$. If $p(s_i|f_j) = 0$ or 1 for all (i, j), we call such causality model a deterministic model; otherwise, we call it a probabilistic model. We also use $A = \{a_1, \ldots, a_k\}$ to denote the list of actions that can be used to verify symptom existence. We describe the relation between actions and symptoms using *Action Codebook* represented as a bipartite graph as shown in Fig. 1. For example, the symptom s_1 can be verified using

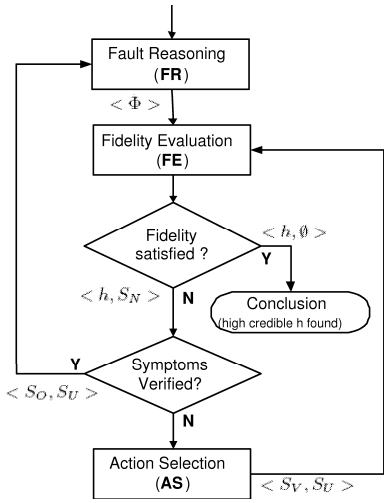

Figure 3: **Active Action Integrated Fault Reasoning**

action a_1 or a_2. The Action Codebook can be defined by network managers based on symptom type, the network topology, and the available fault diagnostic tools. The extended Symptom-Fault-Action graph is viewed as a 5-tuple (S, F, A, E_1, E_2), where fault set F, symptom set S, and action set A are three independent vertex sets. Every edge in E_1 connects a vertex in S and another vertex in F to indicate causality relationship between symptoms and faults. Every edge in E_2 connects a vertex in A and another vertex in S to indicate the Action Codebook. For convenience, in Fig. 2, we introduce the notations used in our discussion throughout this section. The basic Symptom-Fault-Action model can be described as the following:

- For every action, associate an action vertex a_i, $a_i \in A$;
- For every symptom, associate a symptom vertex s_i, $s_i \in S$;
- For every fault, associate a fault vertex f_i, $f_i \in F$;
- For every fault f_i, associate an edge to each s_i caused by this fault with a weight equal to $p(s_i|f_i)$;
- For every action a_i, associate an edge of weight equal to the action cost to each symptom verifiable by this action.

The performance and accuracy are the two most important factors for evaluating fault localization techniques. Performance is measured by fault detection time T, which is the time between receiving the fault symptoms and identifying the root faults. The fault diagnostic accuracy depends on two factors:

(1) the detection ratio (α), which is the ratio of the number of *true* detected root faults (F_d is the total detected fault set) to the number of *actual* occurred faults F_h, formally $\alpha = \frac{|F_d \cap F_h|}{|F_h|}$; and (2) false positive ratio (β), which is the ratio of the number of *false* reported faults to the total number of detected faults; formally $\beta = \frac{|F_d - F_d \cap F_h|}{|F_d|}$ [23]. Therefore, the goal of any fault management system is to increase α and reduce β in order to achieve high accurate fault reasoning results.

The task of the fault reasoning is to search for root faults in F based on the observed symptoms S_O. Our objective is to improve fault reasoning by minimizing the detection time, T and the false positive ratio, β, and maximizing the detection ratio, α.

In order to develop this system, we have to address the following three problems: (1) Given the Fault-Symptom correlation matrix and the set of observed symptoms (S_O), construct a set of the most possible hypotheses, $\Phi = \{h_1, h_2, \ldots, h_p\}, h_i \subseteq F$, that can explain the current observed symptoms; (2) Given a set of possible hypotheses, find the most credible hypothesis h, that can give the best explanation for the current observed symptoms; (3) If the selected hypothesis does not satisfy fidelity requirement, then given the unobserved symptoms S_N and select the minimum-cost actions to search for an acceptable hypothesis. In the following, we will discuss the solution for each problem.

2.3 Integrated Fault-Intrusion Reasoning

The Active Integrated Fault Reasoning (AIR) process (Fig. 3) includes three functional modules: Fault Reasoning (FR), Fidelity Evaluation (FE), and Action Selection (AS). The Fault Reasoning module takes passively observed symptoms S_O as input and returns fault hypothesis set Φ as output. The fault/intrusions hypothesis set Φ might include a set of hypotheses ($h_1, h_2, \ldots, h_n$) where each one contains a set of faults or intrusions that explains all observed symptoms so far. Then, Φ is sent to the Fidelity Evaluation module to check if any hypothesis h_i ($h_i \in \Phi$) is satisfactory. If the most correlated symptoms necessary to explain the fault hypothesis h_i are observed (i.e. high fidelity), then the Fault Reasoning process terminates. Otherwise, a list of unobserved symptoms/alarms, S_N, that contribute to explain the fault hypothesis h_i of the highest fidelity, is sent to the Action Selection module to determine which symptoms have occurred or to distinguish between faults and security breaches. As a result, the fidelity value of hypothesis h_i is adjusted accordingly. The conducted actions return the test result with a set of existing symptoms S_V and non-existing symptoms S_U. The corresponding fidelity value might be increased or decreased based on the action return results. If the newly calculated fidelity is satisfied, then the reasoning process terminates; otherwise, S_O, S_U are sent as new input to the Fault Reasoning

module to create a new hypothesis. This process is repeated until a hypothesis with high fidelity is found. Fidelity calculation is explained later in this section. In the following, we describe the three modules in detail, then discuss the complete Active Integrated Fault Reasoning algorithm.

2.3.1 Heuristic Algorithm for Fault Reasoning

In the Fault Reasoning module, we use a *contribution function*, $C(f_i)$, as a criteria to find faults or intrusions that have the maximal contribution of the observed symptoms. In the probabilistic model, symptom s_i can be caused by a set of faults $f_i, (f_i \in F_{s_i})$ with different possibilities $p(s_i|f_i) \in (0, 1]$. We assume that the Symptom-Fault correlation model is sufficient enough to neglect other undocumented faults (i.e., prior fault probability is very low). Thus, we can also assume that symptom s_i will not occur if none of the faults in F_{s_i} happened. In other words, if s_i occurred, at least one f_i ($f_i \in F_{s_i}$) must have occurred. However conditional probability $p(s_i|f_i)$ itself may not truly reflect the chance of fault f_i occurrence by observing symptom s_i. For example, in Fig. 1, by observing s_1, there are three possible scenarios: f_1 happened, f_2 happened or both happened. Based on the heuristic assumption that the possibility of multiple faults happened simultaneously is low, one of the faults (f_1 or f_2) should explain the occurrence of s_1. In order to measure the contribution of each fault f_i to the creation of s_i, we normalize the conditional probability $p(s_i|f_i)$ to the normalized conditional probability $\hat{p}(s_i|f_i)$ to reflect the relative contribution of each fault f_i to the observation of s_i.

$$\hat{p}(s_i|f_i) = \frac{p(s_i|f_i)}{\sum_{f_i \in F_{s_i}} p(s_i|f_i)} \tag{1}$$

With $\hat{p}(s_i|f_i)$, we can compute normalized posterior probability $\hat{p}(f_i|s_i)$ as follows.

$$\hat{p}(f_i|s_i) = \frac{\hat{p}(s_i|f_i)p(f_i)}{\sum_{f_i \in F_{s_i}} \hat{p}(s_i|f_i)p(f_i)} \tag{2}$$

$\hat{p}(f_i|s_i)$ shows the relative probability of f_i happening by observing s_i. For example, in Fig. 1, assuming all faults have the same prior probability, then $\hat{p}(f_1|s_1) = 0.9/(0.9 + 0.3) = 0.75$ and $\hat{p}(f_2|s_1) = 0.3/(0.9 + 0.3) = 0.25$. The following contribution function $C(f_i)$ evaluates all contribution factors $\hat{p}(f_i|s_i)$, $s_i \in S_{O_i}$ with the observation S_{O_i}, and decides which f_i is the best candidate with maximum contribution value $C(f_i)$ to the currently not yet explained symptoms.

$$C(f_i) = \frac{\sum_{s_i \in S_{O_i}} \hat{p}(f_i|s_i)}{\sum_{s_i \in S_{f_i}} \hat{p}(f_i|s_i)} \tag{3}$$

Therefore, fault reasoning becomes a process of searching for the fault or security breach (f_i) with maximum $C(f_i)$. This process continues until all observed symptoms are explained. The contribution function $C(f_i)$ can be used for both deterministic and probabilistic model.

In the deterministic model, the more the number of symptoms observed, the stronger the indication that the corresponding fault has occurred. Meanwhile, we should not ignore the influence of prior fault probability $p(f_i)$, which represents long-term statistical observation. Since $p(s_i|f_j) = 0$ or 1 in the deterministic model, the normalized conditional probability reflects the influence of prior probability of fault f_i. Thus, the same contribution function can seamlessly combine the effect of $p(f_i)$ and the ratio of $\frac{|S_{O_i}|}{|S_{f_i}|}$ together.

In the fault reasoning algorithm, first it finds the fault candidate set F_C including all faults that can explain at least one symptom s_i ($s_i \in S_O$), then it calls the function $HU()$ to generate and update the hypothesis set Φ until all observed symptoms S_O can be explained. According to the contribution $C(f_i)$ of each fault f_i ($f_i \in F_C$), algorithm 1 searches for the best explanation of S_K, which is currently observed but not yet explained symptom by the hypothesis h_i (lines 2-12). Here $S_K = S_O - \cup_{f_i \in h_i} S_{O_i}$ and initially $S_K = S_O$. If multiple faults have same contribution, multiple hypotheses will be generated (lines 13-17). The searching process (HU) will recursively run until all observed symptoms explained (lines 18-24). Notice that only those hypotheses with minimum number of faults that cover all observed symptoms are included into Φ (lines 23-24).

The above Fault Reasoning algorithm can be applied to both deterministic and probabilistic models with same contribution function $C(f_i)$ but different conditional probability $p(s_i|f_i)$.

2.3.2 Fidelity Evaluation of Fault Hypotheses

The fault hypotheses created by the Fault Reasoning algorithm may not accurately determine the root faults because of lost or spurious symptoms. The task of the Fidelity Evaluation is to measure the credibility of hypothesis created in the reasoning phase given the corresponding observed symptoms. How to objectively evaluate the reasoning result is crucial in fault localization systems.

We use the fidelity function $FD(h)$ to measure the credibility of hypothesis h given the symptom observation S_O. We assume that the occurrence of each fault is independent.

- For deterministic model:

$$FD(h) = \frac{\sum_{f_i \in h} |S_{O_i}|/|S_{f_i}|}{|h|} \tag{4}$$

- For probabilistic model:

$$FD(h) = \frac{\prod_{s_i \in \bigcup_{f_i \in h} S_{f_i}} (1 - \prod_{f_i \in h} (1 - p(s_i|f_i)))}{\prod_{s_i \in S_O} (1 - \prod_{f_i \in h} (1 - p(s_i|f_i)))} \tag{5}$$

Obviously in the deterministic model, if the hypothesis h is correct, $FD(h)$ must be equal to 1 because the corresponding symptoms can be either observed or verified. In the probabilistic model, if related symptoms are observed or verified, $FD(h)$ of a credible hypothesis can still be less than 1 because some symptoms may not happen even when the hypotheses are correct. In either case, our fidelity algorithm takes in consideration a target Fidelity Threshold, $FD_{THRESHOLD}$, that the user can configure to accept hypothesis. System administrators can define the threshold based on long-term observation and previous experience. If the threshold is set too high, even correct hypothesis will be ignored; but if the threshold is too low, then less credible hypothesis might be selected.

Fidelity evaluation function is used to evaluate each hypothesis and decides if the result is satisfactory by comparing to the pre-defined threshold value. If an acceptable hypothesis that matches the fidelity threshold exists, the fault localization process can terminate. Otherwise, the best available hypothesis and a non-empty set of symptoms (S_N) would be verified in order to reach a satisfactory hypothesis in the next iteration.

2.3.3 Action Selection Heuristic Algorithm

The main purpose of this component is to verify or investigate the existence of symptoms/alarms that have the most contribution to identify faults or intrusions. We verify symptoms rather than faults because they are the manifestation of faults in the network and easily trackable. The Action selection also performs actions to identify if the problem is fault or security intrusion. The Action Selection finds the least-cost actions to verify S_N (unobserved symptoms) of the hypothesis that has highest fidelity. As the size of S_N grows very large, the process of selecting the minimal cost action that verifies S_N becomes non-trivial. The Action-Symptoms correlation graph can be represented as a 3-tuple (A, S, E) graph such that A and S are two independent vertex sets representing Actions and Symptoms respectively, and every edge e in E connects a vertex $a_j \in A$ with a vertex $s_i \in S$ with a corresponding weight (w_{ij}) to denote that a_j can verify s_i with cost $w_{ij} = w(s_i, a_j) > 0$. If there is no association between s_i and a_j,

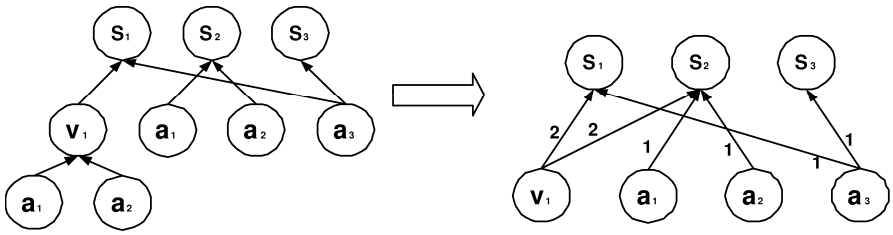

Figure 4: **Symptom-Action Bipartite Graph**

then w_{ij} = 0. Because a set of actions might be required to verify one symptom, we use a virtual action vertex, v_j, to represent this case. The virtual action vertex v_j is used to associate a set of conjunctive actions to the corresponding symptom(s). However, if multiple actions are directly connected to a symptom, then this means any of these actions can be used disjunctively to verify this symptom (Fig. 4). To convert this to a bipartite graph, (1) we set the weight of v_j, $w(s_i, v_j)$, to the total cost of the conjunctive action set, (2) then eliminate the associated conjunctive set to the v_j, (3) associate v_j with all symptoms that can be verified by any action in the conjunctive action set.

The Symptom-Action graph in Fig. 4 presents the verification relationship between symptoms $\{s_1, s_2, s_3\}$ and actions $\{a_1, a_2, a_3\}$. Symptom s_1 can be verified by taking a combination of action a_1 and a_2, which causes a new virtual action vertex v_1 to be created with weight 2. Action v_1 can verify all symptoms (s_1, s_2) that are verifiable by either a_1 or a_2. After converting action combination to a virtual action, Symptom-Action correlation can be represented in a bipartite graph.

The goal of the Action Selection algorithm is to select the actions that cover all symptoms S_N with a minimal action cost. With the representation of Symptom-Action bipartite graph, we can model this problem as a weighted set-covering problem. Thus, the Action Selection algorithm searches for A_i such that A_i includes the set of actions that cover all the symptoms in the Symptoms-Action correlation graph with total minimum cost. We can formally define A_i as the covering set that satisfies the following conditions: (1) $\forall s_i \in S$, $\exists a_j \in A_i$ s.t. $w_{ij} > 0$, and (2) $\sum_{a_i \in A_i, s_j \in S_N} w_{ij}$ is the *minimum*.

The weighted set-covering is an NP-complete problem. Thus, we developed a heuristic greedy set-covering approximation algorithm to solve this problem. The main idea of the Algorithm is simply first selecting the action (a_i or v_i) that has the maximum *relative covering ratio*, $R_i = \frac{|S_{a_i}|}{\sum_{s_j \in S_{a_i}} w_{ij}}$, where this action is added to the final set A_f and removed from the candidate set A_c that includes all actions. Here, S_{a_i} is the set of symptoms that action a_i can verify, $S_{a_i} \subseteq S_N$. Then, we remove all symptoms that are covered by this selected action from the unobserved symptom set S_N. This search continues to find the next action a_i ($a_i \in A_c$), that has the maximum ratio R_i until all symptoms are covered (i.e.,

S_N is empty). Thus, intuitively, this algorithm appreciates actions that have more symptom correlation or aggregation. If multiple actions have the same relative covering weight, the action with more covered symptoms (i.e., larger $|S_{a_i}|$ size) will be selected. If multiple actions have the same ratio, R_i, and same $|S_{a_i}|$, then each action is considered independently to compute the final selected sets for each action and the set that has the minimum cost is selected. Finally, it is important to notice that each single action in the A_f set is necessary for the fault determination process because each one covers unique symptoms.

2.3.4 Algorithm for Active Integrated Fault Reasoning

The major contribution of this work is to incorporate active actions into fault reasoning. Passive fault reasoning could work well if enough symptoms can be observed correctly. However in most cases, we need to deal with interference from symptom loss and spurious symptoms, which could mislead fault localization analysis. As a result of fault reasoning, the generated hypothesis suggests a set of selected symptoms S_N that are unobserved but expected to happen based on the highest fidelity hypothesis. If fidelity evaluation of such a hypothesis is not acceptable, optimal actions are selected to verify S_N. Action results will either increase fidelity evaluation of the previous hypothesis or bring new evidence to generate a new hypothesis. By taking actions selectively, the system can evaluate fault hypotheses progressively and reach to root faults.

Algorithm 2 illustrates the complete process of the AIR technique. Initially, the system takes observed symptom S_O as input. Fault Reasoning is used to search the best hypothesis Φ (Line 3). Fidelity is the key to associate passive reasoning to active probing. Fidelity Evaluation is used to measure the correctness of corresponding hypothesis h ($h \in \Phi$), and produce expected missing symptoms S_N (Line 3). If the result h is satisfied, the process terminates with current hypothesis as output (Line 5 - 6). Otherwise, AIR waits until Initial Passive Period (IPP) expired (Line 8) to initiate actions to collect more evidence of verified symptoms S_V and not-occurred symptoms S_U (Line 10). New evidence will be added to re-evaluate previous hypothesis (Line 13). If fidelity evaluation is still not satisfied, the new evidence with previous observation is used to search another hypothesis (Line 3) until the fidelity evaluation is satisfied. At any point, the program terminates and returns the current selected hypothesis, if either the fidelity evaluation does not find symptoms to verify (S_N is $\emptyset$), or none of the verified symptom had occurred (S_V is $\emptyset$). In either case, this is an indication that the current selected hypothesis is creditable.

(a) (b) (c)

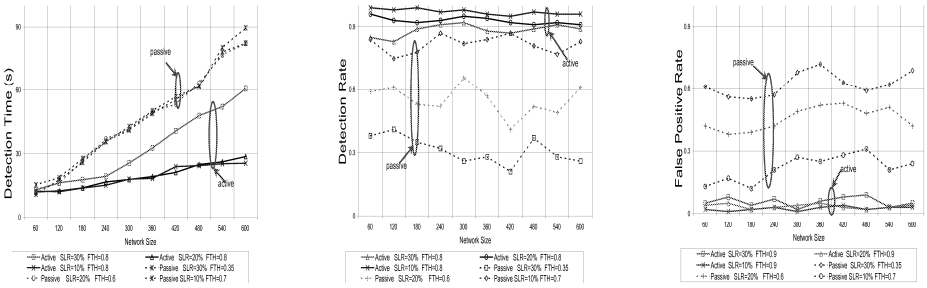

Figure 5: **The Impact of Symptom Loss Ratio (a) Detection Time T (b) Detection rate α (c) False positive rate β**

2.4 Simulation Study

In this section, we describe our simulation study to evaluate Action Integrated fault Reasoning (AIR) technique. We conducted a series of experiments to measure how AIR improves the performance and the accuracy of the fault localization compared with Passive Fault Reasoning (PFR). The evaluation study considers fault detection time T as a performance parameter and the detection rate α and false positive rate β as accuracy parameters.

In our simulation study, the number of monitored network objects D ranged from 60 to 600. We assume every network object can generate different faults and each fault could be associated with 2 to 5 symptoms uniformly distributed. The number of simulated symptoms vary from 120 to 3000 uniformly distributed. We use fault cardinality (FC), symptom cardinality (SC) and action cardinality (AC) to describe the Symptom-Fault-Action matrix such that FC defines the maximal number of symptoms that can be associated with one specific fault; SC defines the maximal number of faults one symptom might correlate to; AC defines the maximal number of symptoms that one action can verify. The independent prior fault probabilities $p(f_i)$ and conditional probabilities $p(s_i|f_j)$ are uniformly distributed in ranges [0.001, 0.01] and (0, 1] respectively. Our simulation model also considers the following parameters: Initial Passive Period (IPP); Symptom Active Collecting Rate ($SACR$); Symptom Passive Collecting Rate ($SPCR$); Symptom Loss Ratio (SLR); Spurious Symptom Ratio (SSR); Fidelity Threshold $FD_{THRESHOLD}$.

The major contribution of this work is to offer an efficient fault reasoning technique that provides accurate results even in worst cases like when symptom passive collecting rate ($SPCR$) is low, and/or symptom loss ratio (SLR) and spurious symptom ratio (SSR) are high. We show how these factors affect

(a) (b) (c)

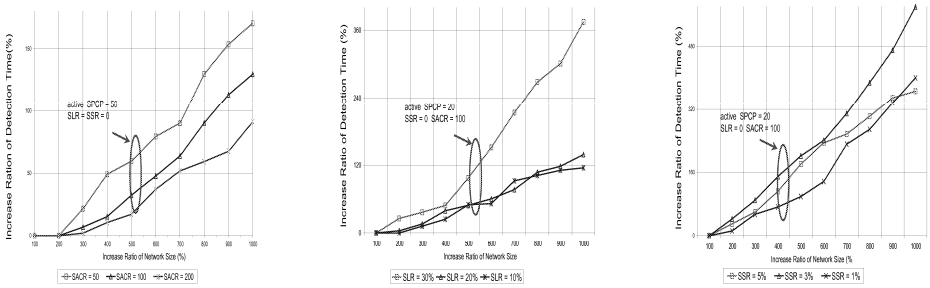

Figure 6: **The Impact of Network Size (a) Without Symptom loss and spurious symptoms (b) With symptom loss (c) With Spurious symptoms**

the performance (T) and accuracy (α and β) of our approach and passive fault reasoning approach.

2.4.1 The Impact of Symptom Loss Ratio

Symptom loss hides fault indications, which negatively affects both accuracy and performance of fault localization process. In order to study the improvement on both the performance and the accuracy of AIR approach, we fix the value of spurious symptom ratio ($SSR = 0$), the initial passive period (IPP = $10sec$), symptom active collecting rate ($SACR$ = 100 symptoms/sec) and symptom passive collecting rate ($SPCR$ = 20 symptoms/sec). In this simulation, we use SLR value that varies from 10% to 30%. With the increase of symptom loss ratio, passive fault reasoning system becomes infeasible. Therefore, in this experiment, we had to reduce the fidelity threshold to a relatively lower value based on the symptom loss ratio so the passive reasoning process can converge in reasonable time. From Fig. 5(a), in contrast to passive approach, AIR system can always reach relatively high fidelity threshold with average performance improvement of 20% to 40%. Hence, when SLR increases, the advantage of active fault reasoning in the performance aspect is more evident. In addition to performance improvement, AIR approach shows high accuracy. With the same settings, Fig. 5(b) and (c) show that active approach gains 20-50$\%$ improvement of detection rate and 20-60$\%$ improvement of false detection rate, even with much different fidelity criteria over the passive reasoning approach.

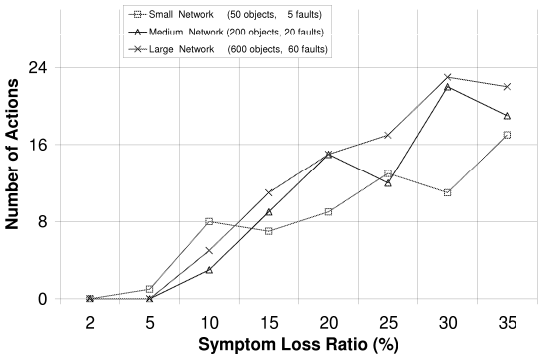

Figure 7: **Intrusiveness Evaluation**

2.4.2 The Impact of Network Size

In this section, we examine the scalability of AIR when network size and the number of symptoms significantly increase. To show this, we measure AIR detection time under different scenarios: (1) without symptom loss and spurious symptom (Fig. 6(a)); (2) with symptom loss only (Fig. 6(b)), and (3) with spurious symptoms only (Fig. 6(c)). In all three cases, when the network size increases 10 times (from 100% to 1000%), the detection time has slowly increased by 1.7 times (170%) and 3.7 times (370%) and 5.8 times (580%) in Fig. 6(a), (b) and (c) respectively. This shows that even in the worst case scenario (Fig. 6(c)), the growth in network size causes a slow linear increases on AIR performance.

2.4.3 The Impact of Symptom Loss on AIR Intrusiveness

AIR intrusiveness is measured by the number of total actions performed to localize faults. As shown in Section 2.3, the intrusiveness of AIR was algorithmically minimized by (1) considering the fault hypothesis of high credibility, and (2) selecting the minimum-cost actions based on the greedy algorithm described in Section 2.3.3. We also conducted experiments to assess the intrusiveness (i.e., action cost) when the loss ratio increases. Loss ratio and network size are the most significant factors that might affect the intrusiveness of AIR. Fig. 7 shows that, with different scale of network sizes and prior fault probability as high as 10%, the number of actions required for fault localization increases slow linearly (from 1 - 22) even when the loss ratio significantly increases (from 2%-35%). For example, in large-scale network of size 600 objects and fault rate is 60 faults per iteration, the number of action performed did not exceed 0.37 action/fault ratio. In addition, AIR was deliberately designed to give the user the control to adjust the intrusiveness of active probing via configuring the following fault reasoning parameters: fidelity threshold, IPP and action coverage.

2.5 Related Work

Many proposed solution were presented to address fault localization problem in communication networks. A number of these techniques use different causality model to infer the observation of network disorder to the root faults. In our survey, we classify the related work into two general categories:

Passive Approach. Passive fault management techniques typically depended on monitoring agents to detect and report network abnormality using alarms or symptom events. These events are then analyzed and correlated in order to reach the root faults. Various event correlation models were proposed including rule-based analyzing system [7], model-based system [11], case-based diagnosing system and model traversing techniques. Different techniques are also introduced to improve the performance, accuracy and resilience of fault localization. In [13], a model-based event correlation engine is designed for multi-layer fault diagnosis. In [19], coding approach is applied to deterministic model to reduce the reasoning time and improve system resilience. A novel incremental event-driven fault reasoning technique is presented in [22] and [23] to improve the robustness of fault localization system by analyzing lost, positive and spurious symptoms.

The techniques above were developed based on passively received symptoms. If the evidence (symptoms) are collected correctly, the fault reasoning results can be accurate. However, in real systems, symptom loss or spurious symptoms (observation noise) are unavoidable. Even with a good strategy [23] to deal with observation noise, those techniques have limited resilience to noise because of their underlying passive approach, which might also increase the fault detection time.

Active Probing Approach. Recently, some researchers incorporate active probings into fault localization. In [2], an active probing fault localization system is introduced, in which pre-planned active probes are associated with system status by a dependency matrix. An on-line action selection algorithm is studied in [9] to optimize action selection. In [10], a fault detection and resolution system is proposed for large distributed transaction processing system.

Active probing approach is more efficient in locating faults in timely fashion and more resilient to observation noise. However, this approach has the following limitation:

- Lack of integrating passive and active techniques in one framework that can take advantage of both approaches.
- Lack of a scalable technique that can deal with multiple simultaneous faults.
- Limitation of some approaches to track or isolate intermittent network faults and performance

related faults because they solely depend on the active probing model.

- The number of required probes might be increased exponentially to the number of possible faults ([9]).

Both passive and active probing approaches have their own good features and limitations. Thus, integrating passive and active fault reasoning is the ideal approach. Our approach combines the good features of both passive and active approaches and overcome their limitations by optimizing the fault reasoning result and action selection process.

3 FAULT AND SECURITY MANAGEMENT ON HIGH-SPEED NETWORKS

3.1 Background

In this section, we discuss network-level fault (*i.e.*, anomalies) and intrusions detection, particularly for high-speed networks. It is very important to have such an integrated fault and intrusion detection because many network faults are often misidentified as intrusions. Such false alerts often make the network administrators turn off the IDS systems. Thus it is of crucial importance to identify both faults and intrusions rapidly and accurately for network-based IDS systems. With the rapid growth of network bandwidth and fast emergence of new attacks/viruses/worms, existing network intrusion detection systems (IDS) are insufficient due to lack of the following features.

First, separating anomalies from intrusions for false positive reduction. To detect unknown attacks and polymorphic worms, statistics-based instead of signature-based intrusion detections have been adopted widely. However, many network element faults, *e.g.*, router misconfigurations and polluted DNS entries, can lead to traffic anomalies that may be detected as attacks.

Second, scalability to high-speed networks. Today's fast propagating viruses/worms (*e.g.*, SQL Slammer worm) can infect most vulnerable machines within ten minutes [17]. Thus, it is crucial to identify such outbreaks in their early phases, which is achievable only in high speed routers. However, existing schemes are not scalable to the link speeds and number of flows for high-speed networks.

It is in general difficult for software-based data recording approaches in IDSes to keep up with the link speed in a high-speed router. Thus, the data recording of high-speed IDSes has to be hardware

implementable, and it is strongly desirable to achieve the following three capabilities: 1) small memory usage; 2) sparse memory accesses per packet [4]; and 3) scalability to large key size.

Third, attack resiliency. To bypass an IDS, attackers can execute denial-of-service (DoS) attacks, or fool the IDS to raise many false positives to conceal the real attack. Thus, the attack resiliency of an IDS is very important. However, existing IDSes often keep per-flow states for detection, which is vulnerable.

Forth, attack root cause analysis for mitigation. Accurate attack mitigation requires IDSes to pinpoint the attack type and flows. This advocates to detect intrusions at the *flow level* instead of the overall traffic. Furthermore, we want to differentiate different types of attacks to choose different mitigation schemes accordingly.

Fifth, aggregated detection over multiple vantage points. Most existing network IDSes assume detection be on a single router or gateway. However, as multi-homing, load balancing based routing, and policy routing become prevalent, even for a connection between a certain source and destination, the packets may traverse different paths [3]. Thus, observation from a single vantage point is often incomplete and affects detection accuracy. Meanwhile, it is very hard to copy all traffic from one router to other routers/IDSes due to the huge data volume.

To meet the requirements above, we propose a new paradigm called DoS resilient High-speed Flow-level INtrusion Detection, HiFIND, leveraging recent work on data streaming computation and in particular, sketches [21]. Essentially, we want to detect as many attacks as possible. As the first step towards this ambitious goal, we aim to detect various port scans (which covers most large-scale worm propagation) and TCP SYN flooding.

While each of these attacks seems relatively easy to be detected, it is indeed very hard to detect a mixture of attacks online at the flow-level. To the best of our knowledge, HiFIND is the *first* DoS resilient high-speed flow-level IDS for port scans and TCP SYN flooding for high-speed networks.

To this end, we leverage and improve sketches to record flow-level traffic as the basis for statistical intrusion detection. Firstly proposed in [20, 21], sketches have not been applied to building IDSes for the following challenges:

- Sketches can only record certain aggregated metrics for some given keys. Since it is not feasible to try all possible combinations of the metrics, what would be the minimal set of metrics for monitoring?

- Existing sketches are all one dimensional. However, various forms of attacks are often hard to

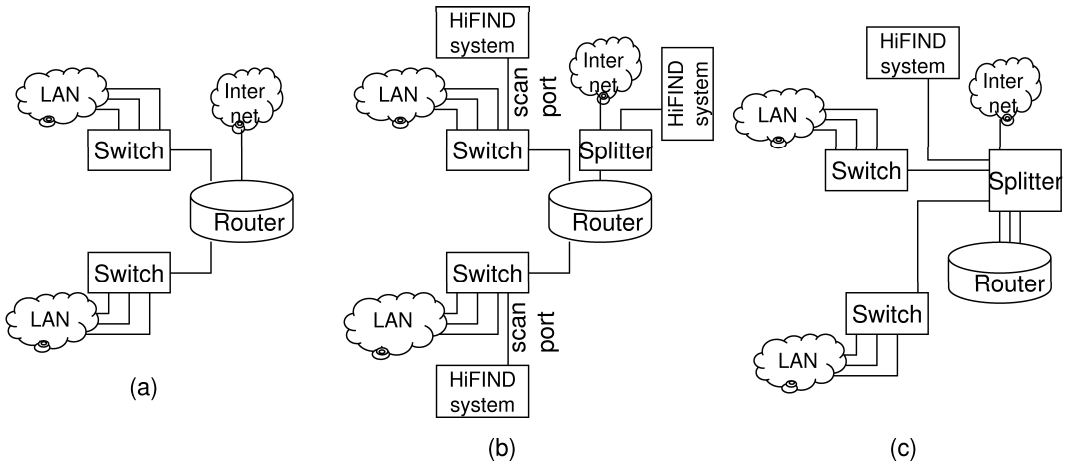

Figure 8: Attaching the HiFIND systems to high-speed routers. (a) original configuration, (b) distributed configuration for which each port is monitored separately, (c) aggregate configuration for which a splitter is used to aggregate the traffic from all the ports.

identify with such single dimensional information.

In this section, we address these two challenges and build the HiFIND prototype system to meet the aforementioned five requirements. We make the following contributions:

- We analyze the attributes in TCP/IP headers and select an optimal small set of metrics for flow-level sketch-based traffic monitoring and intrusion detection. Based on that, we build the HiFIND online high-speed flow-level IDS prototype which is DoS resilient.
- To analyze the attack root cause for mitigation, we design efficient two-dimensional (2D) sketches to distinguish different types of attacks.
- We aggregate the compact sketches from multiple vantage points (*e.g.*, routers) to detect intrusion in the face of asymmetric routing and multi-path routing caused by per-packet load balancing of routers.
- For false positive reduction, we propose several heuristics to separate SYN floodings from network/server congestions and misconfigurations.

As shown in Figure 8, HiFIND detection systems can be implemented as black boxes attached to high-speed routers (edge or backbone routers) of ISPs without affecting the normal operation of the routers.

Approaches	Spoofed DoS	Non-spoofed DoS	HScan	VScan
HiFIND	Yes	Yes	Yes	Yes
TRW(AC)	No	No	Yes	(Yes)
CPM	Yes, but with high FP with port scans		No	No
Backscatter	Yes	No	No	No
Superspreader	No	No	Yes	No

Table 1: Functionality comparison.

For evaluation, we tested the router traffic traces collected at Northwestern University (NU) and Lawrence Berkeley National Labs (LBL). We validate the SYN flooding and port scans detected, and find the HiFIND system is highly accurate. The 2D sketches successfully separate the SYN flooding from port scans, and the heuristics effectively reduce false positives of SYN flooding. Our approach is also very fast in terms of data recording and detection.

3.2 Related Work

3.2.1 Intrusion Detection Systems

Some vendors claim to have multi-gigabit statistical IDSes [1], they usually refer to *average* traffic conditions and use packet sampling [5]. Recent work has proposed detecting large scale attacks, like DoS attacks, port scans, *etc.*, based on the statistical traffic patterns. They can roughly be classified into two categories: Detecting based on the overall traffic [16, 26] and flow level detection [12].

With the first approach, even when we can detect the attack, we still do not have any flow or port knowledge for mitigation. Moreover, attacks can be easily buried in the background network traffic. Thus, such detection schemes tend to be inaccurate; for example, CPM [26] will detect port scans as SYN floodings as verified in Section 3.4. For the second approach, such schemes usually need to maintain a per-flow table (*e.g.*, a per-source-IP table for TRW [12]) for detection, which is not scalable and thus provides a vulnerability to DoS attacks with randomly spoofed IP addresses, especially on high-speed networks. TRW was recently improved by limiting its memory consumption with approximate caches (TRW-AC) [27]. However, spoofed DoS attacks will still cause collisions in TRW-AC, and leave the real port scans undetected[1].

[1]As the authors mentioned in [27], when the connection cache size of 1 million entries reaches about 20% full, each new scan attempt has a 20% chance of not being recorded because it aliases with an already-established connection. Actually,

Functions	Descriptions	k-ary sketch	Reversible sketch
UPDATE(S, y, v)	Update the corresponding values of the given key into the sketch in the monitoring module	$\surd$	$\surd$
v = ESTIMATE (S, y)	Reconstruct the signal series for statistical detection for a given key in the anomaly detection module	$\surd$	$\surd$
S = COMBINE (c_1, S_1,..., c_k, S_k)	Compute the linear combination of multiple sketches $S = \sum_{k=1}^{l} c_k.S_k$ (c_i is coefficient.) to aggregate signals in the anomaly detection module	$\surd$	$\surd$
Y = INFERENCE (S, t)	Return the keys whose values are larger than the threshold in the anomaly detection module		$\surd$

Table 2: Function of sketches (S-Sketch, v-Value, y-Key, Y-Set of keys, t-Threshold).

The existing schemes can detect specific types of attacks, but will perform poorly when facing a mixture of attacks as in the real world. People may attempt to combine TRW-AC and CPM to detect both scans and SYN flooding attacks. However, each of these two approaches can work properly only when the other one works well.

Table 1 shows the high-level functionality comparison of our approach to the other methods. Backscatter detects the spoofed SYN flooding attacks by testing the uniform distribution of destination IPs to which the same source (potential victim) sends SYN/ACK [16]. We use this for validating the SYN flooding detected by HiFIND. Venkataraman *et al.*propose efficient algorithms to detect superspreaders, sources that connect to a large number of distinct destinations [24]. But they may have high false positives with P2P traffic where a single host may connect to many peers for download. PCF was recently proposed for scalable network detection [14]. They do not differentiate among various attacks.

3.2.2 Sketches for Network Monitoring

There is a significant amount of prior work on efficient and online heavy hitter detection [29, 4]. However, these approaches are limited in their applicability to online intrusion detection in that 1) they lack the ability to differentiate different types of attacks; 2) they cannot work with Time Series Analysis based detection algorithms; and 3) they cannot be applied to asymmetric routing environments.

To this end, we designed the original k-ary sketchs [15], and further enhanced them to be reversible sketches [20, 21], which allow us to have separate stages for update, combine and inference, so that we can easily solve the problems mentioned before. In Table 2, we summarize the functions supported by

during spoofed DoS attacks, such collisions can become even worse.

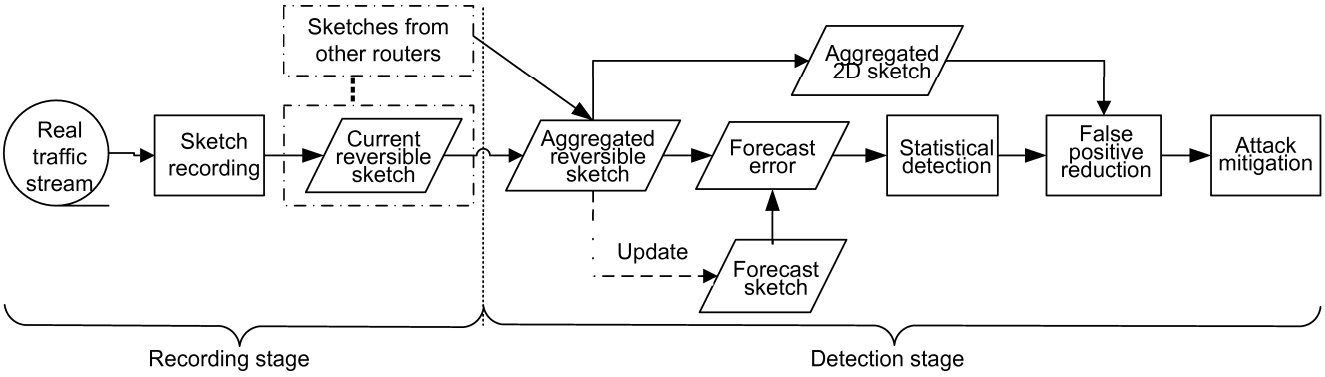

Figure 9: HiFIND System architecture.

sketches.

3.3 Architecture of the HiFIND System

3.3.1 System Architecture

Figure 9 shows the architecture of the HiFIND system. First, we record the network traffic with sketches in each router. Based on linearity of the sketches, we summarize the sketches over multiple routers into an aggregate sketch, and apply time series analysis methods for aggregate sketches to obtain the forecast sketches for change detection. The forecast time series analysis method, *e.g.*, EWMA (exponentially weighted moving average), can help remove noise. By subtracting the forecast sketch from the current one, we obtain the forecast error sketches. Intuitively, a large forecast error implies there is an anomaly, thus the forecast error is the key metric for detection in our system. Moreover, we aggregate the 2D sketches in the same way and adopt them to further distinguish different types of attacks. We also apply other false positive reduction techniques as discussed in Section 3.3.4. Finally, we use the key characteristics of the culprit flows revealed by the reversible sketches to mitigate the attacks. Note that the streaming data recording process needs to be done continuously in real-time, while the detection process can be run in the background executing only once every interval (*e.g.*, every second or minute) with more memory (DRAM).

To deal with asymmetric routing (in Figure 10), for most existing IDS systems, all the packet traces or all connection states have to be transported from one router to the other. Obviously this is very expensive. Moreover if the link is congested when an attack happens, transmission of this data can be

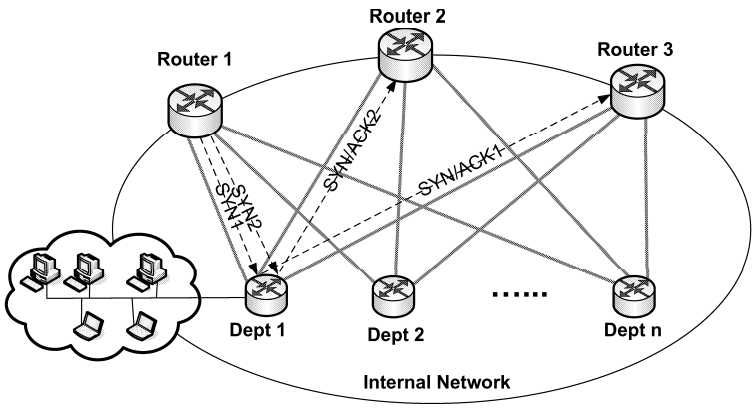

Figure 10: Sample network topology with asymmetric routing and multi-path routing.

very slow. Furthermore, some routers may use per-packet load balancing so that packets of the same flow may traverse different paths.

In contrast, for HiFIND, we summarize the traffic information with compact sketches at each edge router, and deliver them quickly to some central site. Then, with the linearity of the sketches, we can aggregate them and the resulting sketch has all the information as if all the traffics went through the same router.

3.3.2 The Threat Model

Ultimately, we want to detect as many different types of attacks as possible. As a first step, we focus on detecting the two most popular intrusions: TCP SYN flooding (DoS) attack and port scans/worm propagation, which include *horizontal scan (Hscan), vertical scan (Vscan),* and *block scan* [25]. It is also crucial to distinguish them because network administrators need to apply different mitigation schemes for different attacks.

Keys	SYN flooding	Hscan	Vscan	uniqueness
{SIP,Dport}	non-spoofed	Yes	No	1.5
{DIP,Dport}	Yes	No	No	1
{SIP,DIP}	non-spoofed	No	Yes	1.5
{SIP}	non-spoofed	Yes	Yes	2.5
{DIP}	Yes	No	Yes	2
{Dport}	Yes	Yes	No	2

Table 3: The uniqueness of different types of keys.

3.3.3 Sketch-based Detection Algorithm

We denote the key of a sketch as K, the feature value recorded as V, and the reversible sketch as RS(K,V). We also denote the number of SYN packets as #SYN, and the number of SYN/ACK packets as #SYN/ACK.

Here, we only consider the attacks in TCP protocol, *i.e.*, the TCP SYN flooding attacks and TCP port scans. Normally, attackers can choose source ports arbitrarily, so Sport is not a good metric for attack detection. For the other three fields, we can consider all the possible combinations of them, but the key (SIP, DIP, Dport) can only help detect non-spoofed SYN flooding, so we do not use it in the detection process. Table 3 shows the other combinations and their uniqueness. Here, we define the *uniqueness* of a key as the capability of differentiating between different types of attacks. For example, the count of unsuccessful connections aggregated by {SIP} can be used to detect non-spoofed SYN flooding attacks (we count it as 0.5), horizontal scans and vertical scans, so its value of uniqueness is 2.5. The best key would ideally correspond to only one type of attack. Normally a key can be related to several types of attacks, so we need to use more than one dimension to differentiate these attacks as shown in [28]. In this section, we use the tree combinations of two fields as keys for the reversible sketches. Our detection has the following three steps:

Step 1, we use RS({DIP, Dport}, #SYN-#SYN/ACK) to detect SYN flooding attacks because it usually targets a certain service as characterized by the Dport on a small set of machine(s). The value of #SYN-#SYN/ACK means that for each incoming SYN packet, we will update the sketch by incrementing one, while for each outgoing SYN/ACK packet, the sketch will be updated by decrementing one. In fact, similar structures can be applied to detect any partial completion attacks [14]. The reversible

sketch can further provide the victim IP and port number for mitigation as in Section 3.3.2. We denote this set of DIPs as $FLOODING\_DIP\_SET$.

Step 2, we use RS({SIP, DIP}, #SYN−#SYN/ACK) to detect any intruder trying to attack a particular IP address. The detected attacks can be non-spoofed SYN flooding attacks or vertical scans. For each {SIP, DIP} entry, if DIP $\in FLOODING\_DIP\_SET$, we put the SIP into the $FLOODING\_SIP\_SET$ for the next step; otherwise the {SIP, DIP} is the attacker's IP and victim IP of a vertical scan.

Step 3, we use RS({SIP, Dport}, #SYN−#SYN/ACK) to detect any source IP which causes a large number of uncompleted connections to a particular destination port. For each {SIP, Dport} entry, if SIP $\in FLOODING\_SIP\_SET$, it is a non-spoofed SYN flooding; otherwise, it is a horizontal scan.

Here we apply EWMA algorithm as the forecast models to do change detection. We denote $\mathbf{M}_0(t)$ as the current #SYN−#SYN/ACK at the time interval t, and $\mathbf{M}_f(t)$ as the forecasted #SYN−#SYN/ACK at the time interval t, we have

$$\mathbf{M}_f(t) = \begin{cases} \alpha \mathbf{M}_0(t-1) + (1-\alpha)\mathbf{M}_f(t-1) & t > 2 \\ \mathbf{M}_0(1) & t = 2 \end{cases} \tag{6}$$

The difference between the forecasted value and the actual value, $\mathbf{e}_t = \mathbf{M}_0(t) - \mathbf{M}_f(t)$, is then used for detection.

3.3.4 Separating Faults/Anomalies from SYN Flooding

There can be a number of factors other than SYN flooding that may cause a particular destination IP and port with a large number of unacknowledged SYNs. For instance, flash crowds, network/server congestions/failures, and even polluted or outdated DNS entries may cause a large number of SYNs without SYN/ACK at the edge routers. These may cause high false positives in our detection scheme. For the flash crowds, it is difficult, if not impossible, to differentiate it from the SYN flooding attacks without payload information as discussed in [12]. Thus we aim at reducing the false positives caused by the other two behaviors listed above.

First, we add filters to reduce false positives caused by bursty network/server congestions/failures based on the ratio of #SYN comparing with #SYN/ACK and the fact that attacks may last some time. Second, we add filters to reduce the false positives caused by misconfigurations or related problems based on the fact that DoS attacks attack some active IP addresses and services.

3.4 Evaluation

3.4.1 Evaluation Methodology

In this section, we evaluate HiFIND with two dataset. One is the router traffic traces collected at the Lawrence Berkeley National Laboratory (LBL) which consists of about 900M netflow records. The other is the traffic traces of Northwestern University (NU, which has several Class B networks) edge routers. The router exports netflow data continuously which is recorded with sketches of HiFIND on the fly. The one day experiment in May 2005 consists of 239M netflow records, which comes 1.8T total traffic.

Unless denoted otherwise, the default time interval for constructing the time series is one minute. The data recording part of the HiFIND system consists of 1) three reversible sketches (RS), one for $\{\texttt{SIP}, \texttt{Dport}\}$, one for $\{\texttt{DIP}, \texttt{Dport}\}$, and the other for $\{\texttt{SIP}, \texttt{DIP}\}$, 2) one original sketch (OS) for $\{\texttt{DIP}, \texttt{Dport}\}$, and 3) two 2D sketches for $\{\texttt{SIP}, \texttt{Dport}\} \times \{\texttt{DIP}\}$ and $\{\texttt{SIP}, \texttt{DIP}\} \times \{\texttt{Dport}\}$. For all the RS and 2D sketches we update $\texttt{\#SYN} - \texttt{\#SYN/ACK}$ as the value, and only for the OS, we use $\texttt{\#SYN}$ as the value.

The following parameters are chosen based on systematic study as in [21, 15]. We adopt 6 stages for each RS and OS, and 5 stages for each 2D sketch in our system. We use 2^{12} buckets for each stage in 48-bit RS, 2^{16} buckets for each stage in the 64-bit RS, and 2^{14} buckets for all their verification sketches. 2^{14} buckets are applied for each stage in OS. We also use $2^{12} \times 64$ buckets for each stage of the 2D sketches. Therefore, the total memory is 13.2MB.

Both NU and LBL have a large amount of traffic, so we set the detection threshold to be one unresponded SYN packet per second.

3.4.2 Sketches Highly Accurate in Recording Traffic for Detection

Table 4 shows the three phases of our detection results. We first detect attacks using reversible sketches with algorithms described in Section 3.3.3. The results are shown as "Raw results"("Phase 1") in Table 4. 2D sketches reduce the false positives for port scans introduced by SYN flooding attacks ("Phase 2") of Table 4. The heuristics in Section 3.3.4 reduce false positives of SYN flooding attacks("Phase 3").

To evaluate the errors introduced by sketches, we compare the results obtained from the same detection algorithm but with two different types of traffic recording: 1) sketches; 2) accurate flow table to hold per-flow information (we call it non-sketch method). We find that we detect exactly the same attacks for

Traces	Attack type	Phase1: Raw results	FP reduction	
			Phase2: Port scan	Phase3: Flooding
NU	SYN flooding	157	157	32
	Hscan	988	936	936
	Vscan	73	19	19
LBL	SYN flooding	35	35	0
	Hscan	736	699	699
	Vscan	40	1	1

Table 4: Detection results under three phases.

Data	TRW	HiFIND	Overlap number
NU	497	512	488
LBL	695	699	692

Table 5: Horizontal scans detection comparison of HiFIND and TRW aggregated by source IP.

Data	CPM	HiFIND	Overlap number
NU	1422	1427	1422
LBL	1426	0	0

Table 6: TCP SYN flooding detection comparison of HiFIND and CPM.

the two configurations with very different amounts of memory (see memory consumption discussion in Section 3.4.5). There is no false positive in our results. This shows sketches are highly accurate in recording the traffic for detection.

3.4.3 HiFIND Outperforms Other Existing Network IDSes

Detection Over a Single Router We compare the HiFIND with other state-of-the-art work as introduced in Section 3.2: the TRW [12] for port scan detection and the CPM [26] for SYN flooding detection.

For TRW experiments, we choose similar parameters as those in their paper. We apply the TRW on both datasets with the same threshold. Repeated alerts are removed from the results of both methods. Table 5 shows the comparison results of our methods with TRW for Hscan detection. We observe that the scans detected by these two methods have very good overlap, except for a few special cases. There are a small number of Hscans detected by HiFIND but not TRW, because some attacks have both successful and unsuccessful connection attempts, but TRW cannot detect those suspicious ones in this category. There are also a very small number of Hscans detected by TRW but not HiFIND, because they are the combination of multiple small scans, which are too stealthy to be captured by our threshold. It

Anonymized SIP	Dport	#DIP	Cause
204.10.110.38	1433	56275	SQLSnake scan
109.132.101.199	22	45014	Scan SSH
95.30.62.202	3306	25964	MySQL Bot scans
162.39.147.51	6101	24741	Unknown scan
15.192.50.153	4899	23687	Rahack worm

Table 7: Five major senarios of the top 10 Hscans in NU experiment.

Anonymized SIP	Dport	#DIP	Cause
98.198.251.168	135	64	Nachi or MSBlast worm
3.66.52.227	445	64	Sasser and Korgo worm
2.0.28.90	139	64	NetBIOS scan
98.198.0.101	135	64	Nachi or MSBlast worm
165.5.42.10	5554	62	Sasser worm

Table 8: 5 major scenarios of the bottom 10 Hscans in NU experiment.

is our future work to further investigate it.

Next, we compare our method with CPM for SYN flooding attack detection. The results are shown in Table 6. In the LBL traces, there is no SYN flooding, but a very large number of scans. CPM cannot differentiate them. On the other hand, CPM and HiFIND have very similar results for the NU data because most time intervals contain SYN flooding. Meanwhile, there is a small number of intervals in which SYN flooding is buried in the rest of the normal traffic, so CPM cannot detect them.

Aggregated Detection over Multiple Routers In this section, we consider the network topology of Figure 10 discussed in Section 3.3 and evaluate the performance of HiFIND and TRW under such scenarios. To simulate asymmetric routing and multi-path routing caused by per-packet load balancing on routers, we split the packet level trace from a Northwestern University edge router into three routers randomly, for both inbound and outbound packets. For each packet, we randomly select an edge router to deliver, *i.e.*, for any single connection, the incoming SYN packet and the outgoing SYN/ACK packet have 2/3 probability to go through different routers.

For HiFIND , we obtain the same results as those when the traffic goes through the same router, *i.e.*,

Methods	2.5Gbps		10Gbps	
	1min	5min	1min	5min
HiFIND w/ sketch	13.2M		13.2M	
HiFIND w/ complete info	10.3G	51.6G	41.25G	206G
TRW	5.63G	28G	22.5G	112.5G

Table 9: Memory comparison(bytes).

the results in Section 3.4.3. In comparison, we apply TRW to the data on each router for detection and then sum the result up. We found their approach had high false positives or negatives in this case.

3.4.4 Detected Intrusions Successfully Validated

In this section we manually examine a certain number of attacks for validation.

SYN Flooding We validate our SYN flooding detection results with backscatter [16]. Among the 32 SYN floodings detected, there are 21 matched with backscatter results. For the other 11 attacks, three are due to threshold boundary effect.

Horizontal Scans We manually validate horizontal scans, in particular, the top 5 and bottom 5 attacks in terms of their change difference. Due to limited space, Table 7 shows the top 5 Hscans, and Table 8 shows the bottom 5 Hscans from the NU experiment. Detailed evaluation can be found in our technical report [8].

Vertical Scans We also manually validate vertical scans. In the LBL trace, we found one vertical scan. It scanned some well-known service ports, such as HTTPS(81), HTTP-Proxy(8000,8001,8081). In the NU experiment, we found in total 19 vertical scans. We manually checked all of them and found the vertical scans are mostly interested in the well known service ports and Trojan/Backdoor ports.

3.4.5 Evaluation Results for Online Performance Constraints

Small Memory Consumption In our experiments, we only use a total memory of 13.2MB for traffic recording. Note that such settings work well for a large range of link speeds.

On the other hand, if hash tables are used to record every flow, much larger memory is required as shown in Table 9. We consider the worst-case traffic of all-40-byte packet streams with 100% utilization

of the link capacity. There is a spoofed SYN flooding attack with a different source IP for each packet. For the method without sketch, it needs at least three hash tables corresponding to the three reversible sketches in our detection methods.

Small Memory Access per Packet There are 15 memory accesses per packet for 48 bit reversible sketches and 16 per packet for 64-bit reversible sketches (see [21] for details). For each two-dimensional sketch, we only need 5 memory accesses per packet, one for each 2D hash matrix. Thus, when recording these sketches in parallel or in pipeline, the HiFIND system has a very small number of memory accesses per packet and is capable of online monitoring.

High Speed Traffic Monitoring In HiFIND system, the speed of 2D sketches is much faster than that of the reversible sketches. Thus, the speed is dominated by the latter. With our prototype single FPGA board implementation, we are able to sustain 16.2 Gbps throughput for recording all-40-byte packet streams (the worst case) with a reversible sketch.

We can also use multi-processors to record multiple sketches simultaneously in software. We record 239M items with one reversible sketch in 20.6 seconds, *i.e.*, 11M insertions/sec. For the worst case scenario with all 40-byte packets, this translates to around 3.7 Gbps. These results are obtained from code that is not fully optimized and from a machine that is not dedicated to this process.

For the on-site NU experiments, the HiFIND system used 0.34 seconds on average to perform detection for each one-minute interval, and the standard deviation is 0.64 seconds. The maximum detection time (for which the interval contains the largest number of attacks) is 12.91 seconds, which is still far less than one minute. In order to show the scalability of HiFIND, we further do some stress experiments. We compress the NU data by the factor of 60, and detect the top 100 anomalies in each interval. The HiFIND system used 35.61 seconds on average in detection for each interval. The maximum detection time is 46.90 seconds.

4 SUMMARY

Analyzing fault and security alarms is very crucial for identifying and localizing network problems such as failure or intrusions. In this chapter, we show how to integrate fault and security management to relieve the heavy burden of manual diagnosis by system administrators and improve the accuracy of fault and intrusion identification. In the first section, a novel technique called ACTIVE INTEGRATED

FAULT REASONING or AIR is presented. This technique is the first to seamlessly integrate passive and active fault reasoning in order to reduce fault detection time as well as improve the accuracy of fault diagnosis. AIR can be similarly used to correlate security alarms and identify potential intrusions or attacks. When there are incomplete symptoms to identify the root cause, AIR initiates an optimal active probing to investigate and identify the problem with reasonable certainly. The AIR approach is designed to minimize the intrusiveness of active probing via enhancing the fault hypothesis and optimizing the action selection process. Our simulation results show that AIR is robust and scalable even in extreme scenarios such as large network size and high spurious and symptom loss rate.

In the second section, we show how network-level intrusion detection systems can integrate fault and intrusion detection to avoid misidentifying faults as intrusions. This decreases false alerts which often make the network administrators turn off the IDS systems. Thus it is of crucial importance to identify both faults and intrusions rapidly and accurately for network-based IDS systems. Leveraging data streaming techniques such as the reversible sketch, we propose HiFIND, a High-speed Flow-level Intrusion Detection system. In contrast to existing intrusion detection systems, HiFIND 1) separates anomalies to limit false positives in detection; 2) is scalable to flow-level detection on high-speed networks; 3) is DoS resilient; 4) can distinguish SYN flooding and various port scans (mostly for worm propagation) for effective mitigation; and 5) enables aggregate detection over multiple routers/gateways. Both theoretical analysis and evaluation with several router traces show that HiFIND achieves these properties.

References

[1] Arbor Networks. Intelligent Network Management with Peakflow Traffic. `http://www.arbornetworks.com/download.php`.

[2] R. I. Brodie, M. and S. Ma. Optimizing probe selection for fault localization. In *IEEE/IFIP (DSOM)*, 2001.

[3] Cisco Inc. Per-Packet Load Balancing, 2003. `http://www.cisco.com/univercd/cc/td/doc/product/software/ios120/120newft/120limit/120s/120s21/pplb.pdf`.

[4] G. Cormode and S. Muthukrishnan. What's new: Finding significant differences in network data streams. In *Proc. of IEEE Infocom*, 2004.

[5] N. Duffield, C. Lund, and M. Thorup. Flow sampling under hard resource constraints. In *Proc. of ACM SIGMETRICS*, 2004.

[6] Y. T. Ehab Al-Shaer. Qos path monitoring for multicast networks. In *Journal of Network and System Management (JNSM)*, 2002.

[7] A. K. M. G. Liu and E. J. Yang. Composite events for network event correlation. In *Integrated Network Management VI*, pages p247–260, Boston, MA, 1999.

[8] Y. Gao, Z. Li, and Y. Chen. Towards a high-speed router-based anomaly/intrusion detection system. `http://list.cs.northwestern.edu/hpnaidm.html`.

[9] N. O. S. M. G. G. I. Rish, M. Brodie. Real-time problem determination in distributed systems using active probing. In *IEEE/IFIP (NOMS)*, Soul, Korea, 2004.

[10] G. K. J. Guo and P. Kermani. Approaches to building self healing system using dependency analysis. In *IEEE/IFIP (NOMS)*, Soul, Korea, 2004.

[11] . Jakobson and M. D. Weissman. Alarm correlation. In *IEEE Network*, pages p52–59, 1993.

[12] J. Jung et al. Fast portscan detection using sequential hypothesis testing. In *Proc. of the IEEE Symposium on Security and Privacy*, 2004.

[13] M. S. K. Appleby, G. Goldszmidt. Yemanja - a layered fault localization system for multi-domain computing utilities. In *Journal of Network and Systems Management*, 2002.

[14] R. R. Kompella, S. Singh, and G. Varghese. On scalable attack detection in the network. In *Proc. of ACM/USENIX IMC*, 2004.

[15] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen. Sketch-based change detection: Methods, evaluation, and applications. In *Proc. of ACM SIGCOMM Internet Measurement Conference (IMC)*, 2003.

[16] D. Moore et al. Inferring Internet denial of service activity. In *Proceedings of the 2001 USENIX Security Symposium*, Aug. 2001.

[17] D. Moore et al. The spread of the Sapphire/Slammer worm. http://www.caida.org, 2003.

[18] X. Ou, W. F. Boyer, and M. A. McQueen. A scalable approach to attack graph generation. In *13th ACM Conference on Computer and Communications Security (CCS 2006)*, Alexandria, VA, October 2006.

[19] Y. Y. D. O. S. Kliger, S. Yemini and S. Stolfo. A coding approach to event correlation. In *Proceedings of the Fourth International Symposium on Intelligent Network Management*, 1995.

[20] R. Schweller, A. Gupta, E. Parsons, and Y. Chen. Reversible sketches for efficient and accurate change detection over network data streams. In *IMC*, 2004.

[21] R. Schweller, Z. Li, Y. Chen, et al. Reverse hashing for high-speed network monitoring: Algorithms, evaluation, and applications. In *Proc. of IEEE Infocom*, 2006.

[22] M. Steinder and A. S. Sethi. Increasing robustness of fault localization through analysis of lost, spurious, and positive symptoms. In *In Proc. of IEEE INFOCOM*, New York, NY, 2002.

[23] M. Steinder and A. S. Sethi. Probabilistic fault diagnosis in communication systems through incremental hypothesis updating. In *Computer Networks*, pages p537–562, 2004.

[24] S. Venkataraman, D. Song, P. Gibbons, and A. Blum. New streaming algorithms for superspreader detection. In *the Annual Network and Distributed System Security Symposium (NDSS)*, 2005.

[25] Vinod et al. Internet intrusions: Global characteristics and prevalence. In *Proc. of ACM SIGMETRICS*, 2003.

[26] H. Wang, D. Zhang, and K. G. Shin. Detecting SYN flooding attacks. In *Proc. of IEEE INFOCOM*, 2002.

[27] N. Weaver et al. Very fast containment of scanning worms. In *USENIX Security Symposium*, 2004.

[28] G. Y, Z. Li, and Y. Chen. A dos resilient flow-level intrusion detection approach for high-speed networks. In *Proc. of the IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2006.

[29] Q. G. Zhao, A. Kumar, and J. J. Xu. Joint data streaming and sampling techniques for detection of super sources and destinations. In *Proc. of ACM/USENIX Internet Measurement Conference*, 2005.

Algorithm 1 Hypothesis Updating Algorithm **HU**(h, S_K, F_P)

Input: hypothesis h, observed but uncovered symptom set S_K, fault candidate set F_P

Output: fault hypothesis set Φ

$c_{max} = 0$
for all $f_i \in F_P$ **do**
  if $C(f_i) > c_{max}$ **then**
    $c_{max} \leftarrow C(f_i)$
    $F_S \leftarrow \emptyset$
    $F_S \leftarrow F_S \cup \{f_i\}$
  else
    if $C(f_i) = c_{max}$ **then**
      $F_S \leftarrow F_S \cup \{f_i\}$
    end if
  end if
end for
for all $f_i \in F_S$ **do**
  $h_i \leftarrow h \cup \{f_i\}$
  $S_{K_i} \leftarrow S_K - S_{O_i}$
  $F_{P_i} \leftarrow F_P - \{f_i\}$
end for
for all $S_{K_i} = \emptyset$ **do**
  if $S_{K_i} = \emptyset$ **then**
    $\Phi \leftarrow \Phi \cup \{h_i\}$
  end if
end for
if $\Phi \neq \emptyset$ **then**
  return $< \Phi >$
else
  /* No h_i can explain all S_O*/
  for all h_i **do**
    $HU(h_i, S_{K_i}, F_{P_i})$
  end for
end if

Algorithm 2 Active Integrated Fault Reasoning S_O

Input: S_O

Output: fault hypothesis h

$S_N \leftarrow S_O$
while $S_N \neq \emptyset$ **do**
  $\Phi = FR(S_O)$
  $< h, S_N >= FE(\Phi)$
  if $S_N = \emptyset$ **then**
    return $< h >$
  else
    if IPP experied **then**
      /*used to schedule active fault localization periodically*/
      $< S_V, S_U >= AS(S_N)$
    end if
  end if
  $S_O \leftarrow S_O \cup S_V$
  $< h, S_N >= FE(\{h\})$
  if $S_N = \emptyset \parallel S_V = \emptyset$ **then**
    return $< h >$
  end if
end while