SECURITY AND COMMUNICATION NETWORKS
Security Comm. Networks. 2008; **1**:161–177
Published online 29 February 2008 in Wiley InterScience
(www.interscience.wiley.com) DOI: 10.1002/sec.9

Designing fault tolerant networks to prevent poison message failure

Xiaojiang Du[1]*,†,‡, Mark A. Shayman[2§] and Ronald A. Skoog[3¶]

[1]*Department of Computer Science, North Dakota State University, N.D., U.S.A.*
[2]*Department of Electrical and Computer Eng., University of Maryland, College Park, M.D., U.S.A.*
[3]*Telcordia Technologies, U.S.A.*

Summary

Poison message failure is a mechanism that has been responsible for large-scale failures in both telecommunications and IP networks. We design a fault management framework that integrates passive diagnosis and active diagnosis to identify the poison message and prevent network instability. Passive diagnosis uses real-time inference and reasoning techniques to analyze network information and generates a probability distribution of the poison message, and the probability distribution is used in active diagnosis for further failure identification. In active diagnosis, message filtering is used to block suspect message types. Blocking messages affects network performance and service. The tradeoff of message filtering is formulated as a Markov Decision Process (MDP). The large size of the state space makes it impractical to use traditional techniques to solve the MDP. Consequently, we use a combination of reinforcement learning and feature-based function approximation to obtain a suboptimal policy. Extensive simulations demonstrate the effectiveness of passive diagnosis, and show that the suboptimal policy performs significantly better than a well-known heuristic policy. Copyright © 2008 John Wiley & Sons, Ltd.

KEY WORDS: fault management; fault identification; poison message failure; IP networks; passive diagnosis; active diagnosis; Markov Decision Process

1. Introduction

There have been a number of incidents in which 'bad behavior' propagated among network elements and resulted in a significant degradation of network throughput and performance. Labovitz *et al.*, have reported on Internet routing instabilities and related network problems in References [1] and [2]. ATM networks running routing protocols that were proprietary forms of PNNI have caused major outages of AT&T and MCI-WorldCom Frame Relay networks [3,4]. Regional Bell Operating Company (RBOC) Signaling System No. 7 of Common Channel Signaling Network (CCSN) outages in 1990 and 1991

*Correspondence to: Xiaojiang (James) Du, Department of Computer Science, North Dakota State University, 258 IACC, Fargo, N.D. 58105, U.S.A.
†E-mail: dxj@ieee.org
‡Assistant Professor.
§Professor.
¶Senior Scientist.

caused prolonged disruptions in voice telephone networks [5]. The AT&T 4ESS^{TM} switched voice telephone network became unstable for around 6 hours (with greater than a 50% call blocking for most of that time) caused by propagating switching system failures [6].

The above incidents were all the result of non-malicious causes such as software bugs, protocol deficiencies, and system design problems. An important aspect of the causes of these network events is that they are very unlikely to be discovered in normal testing, due to either the code causing the problem is in an obscure, rarely used branch of code; or the trigger event drives a portion of the network into conditions outside the normal parameter ranges. As a result, these types of problems appear to be rare, isolated events. The concern, however, is that there can be many such undetected software bugs and protocol deficiencies lurking in deployed systems, and if these become known by a malicious party they can be exploited to cause significant harm. We already see a large number of Distributed Denial of Service (DDoS) attacks that occur daily, and virus and worm attacks that occur less frequently, but inflict significant economic costs.

Previous efforts to address major network outages have focused on areas such as software reliability, disaster prevention and recovery, network topology design, network engineering, and congestion control. In relation to fault propagation, the main idea that has previously been considered is software diversity (e.g., [7]). However, network providers have studied software diversity and they have determined that this solution is too costly and unmanageable. Furthermore, software diversity does not address flaws in a standardized protocol. To the best of our knowledge, there is no previous work that explores the idea of overlaying a control capability on a network with 'unreliable' components to prevent the propagation of failures. Virtually all of the previous work related to this problem is directed at how to design systems and protocols to be robust assuming *complete knowledge and control* of the underlying protocols and systems. A typical example is the design of system and network congestion controls. There seems to be no previous work on developing techniques for containing undesirable network behavior under the assumption that network elements and protocols are vulnerable due to unknown flaws or malicious alterations.

Our approach is to identify generic propagation mechanisms that can cause network instability, and design control techniques to stabilize the network, without knowing ahead of time what specific mechanisms are being used to drive the network into an unstable state. There are several generic failure propagation mechanisms that can cause unstable network. In this paper, we focus on one of these that we refer to as the *poison message failure* propagation mechanism. This is the propagation mechanism that caused the AT&T 4ESS^{TM} network to go unstable in 1990 [6]. Recently, it has been responsible for the failure of a large number of routers in an ISP network. Next, we describe the problem and its features.

1.1. The Poison Message Failure Problem

The poison message failure problem is stated as follows: a trigger event causes a particular network control or management message (the poison message) to be sent to other network elements. Some or all of the network elements have a software or protocol 'bug' that is activated on receipt of the poison message. The activated 'bug' will cause the node to fail with some probability. If the network control or management is such that this message is persistently passed among the network nodes, and if the node failure probability is sufficiently high, then large-scale instability can result. Several such incidents have occurred in telecommunication and other networks, such as an AT&T telephone switching network incident in 1990 [6]. In this case, the poison message was an ordinary control message passed between switches; there was nothing abnormal about the message itself. It was a software defect in the nodes that caused the message to have a 'poison' effect. We are also aware of an incident in which malformed OSPF packets functioned as poison messages and caused failure of the routers in an entire routing area for an Internet Service Provider.

1.2. The Problem Features

The poison message problem has several differences from traditional network fault management problems. Typical network fault management deals with localized failures [8,9], for example, there is something wrong with a switch. What propagates is not the failure itself but the consequences of the failure on the data plane, for example, congestion builds up at upstream nodes. Then multiple alarms are generated that need to be correlated to find the root cause [10]. In the poison message problem, the failure itself propagates, and the propagation occurs through control or management plane messages. It is also different from

DOI:10.1002/sec

worms or viruses in that worms and viruses propagate at the application layer.

A message type may have a characteristic pattern of propagation. For example, OSPF Link State Advertisements (LSA) messages use flooding so a poison message carried by OSPF LSA messages is passed to all neighbors. In contrast, RSVP Path messages follow shortest paths so a poison message carried by RSVP is passed to a sequence of routers along such a path. Consequently, we expect pattern recognition techniques to be useful in helping to infer the responsible message type.

To study the poison message problem, we make the following three assumptions:

(1) Centralized network management is available.
(2) Recent communication history (exchanged messages) of each node in a communication network can be recorded.
(3) Because the probability of two message types carrying poison messages at the same time is very small, we assume there is only one message type carrying the poison message when such failure occurs.

There are several ways to record recently exchanged messages. One way is to let each node record messages that were recently exchanged with its neighbors. When a node fails, all its neighbors send the recorded messages to a central manager. In our experiments, each node stores a certain number (or for a certain time period) of most recent control/management messages exchanged with its neighbors. Of course, there is a problem when multiple nodes fail simultaneously. If a neighbor node B fails at the same time as (or in a very short time after) a node A fails, then the recorded message in node B cannot be delivered to the central manager. In today's high-speed network, the transmission of a few control messages (not potentially large amount of data packets) only takes a little time so the probability of having the above event should be low. In case it happens, then it means that the central manager only has partial information about message exchanged at failed nodes. The passive diagnosis (details given in Section 2) can still be performed by using the partial information. Of course the diagnosis may have less accuracy. In Reference [9], Bouloutas already addressed the problem of doing finite state machine (FSM) analysis based on partial information.

Our goal is to design a fault management framework that can identify the poison message type, or at least the protocol, carrying the poison message, and block the propagation of the poison message until the network is stabilized. We propose integrating passive diagnosis and active diagnosis to identify and block the poison message.

The contributions of this paper are: (1) we designed a network fault management framework that can effectively identify the poison message. (2) We proposed multiple approaches for passive diagnosis to deal with different networks. Each approach is suitable for certain networks and these approaches can be combined together. In particular, we show that pattern classification using neural networks is an effective technique. (3) We modeled active diagnosis as a Markov Decision Process (MDP) and obtained a good suboptimal policy using reinforcement learning and function approximation techniques. The rest of the paper is organized as follows. In Section 2, we present passive diagnosis, three different approaches in passive diagnosis, and the simulation results. We describe active diagnosis in Section 3 and present the experimental results in Section 4. Finally, we give the conclusions in Section 5.

2. Passive Diagnosis

Passive diagnosis uses real-time inference and reasoning techniques to analyze network information and generates a probability distribution of the poison message. We propose three approaches for passive diagnosis. Different approaches are suitable for different networks, and they can be combined together. We briefly describe each approach below.

1. *Finite state machine approach:* this is a distributed approach used at single failed node. All communication protocols can be modeled as FSM [9]. When a node fails, the neighbor of the failed node will retrieve stored messages that were exchanged with the failed node. Messages are classified according to protocols. Then the message sequence is used to match the FSM model of the corresponding protocol. We can check whether those messages match (are consistent with) the FSM model, and we can also determine what state a protocol was in immediately prior to failure. If there are one or more mismatches between the message sequence and the FSM model, that probably means there was something wrong in the protocol. We have reported the details of the FSM approach in Reference [11].
2. *Correlating message approach:* event correlation is an important technique in fault management.

DOI:10.1002/sec

Recently exchanged messages are stored by neighbor nodes. We analyze stored messages of multiple failed nodes. Since multiple nodes are failed by the same poison message, there must be some common features in the stored messages. One can compare the stored messages of those failed nodes. If for a protocol, there are no common received messages among the failed nodes, then we can rule out this protocol, that is, this protocol is not responsible for the poison message. On the other hand, if many failed nodes have the same final message in one protocol, we can use Bayes' Rule to calculate the probability of the final message being the poison one. The details of the correlating message approach are given in Subsection 2.2.

3. *Utilizing node failure pattern:* different message types have different failure propagation patterns. One way to exploit the node failure pattern is to use a neural network classifier. The neural network is trained *via* simulation. A simulation testbed is set up for a real communication network. The testbed has the same topology and protocol configuration as the real network. Every message type used in the network is simulated as the poison message, and the simulation is run for the node failure probability being different values. We record the node failure patterns for each poison message type, and these data are used to train the neural network. After training, the neural network is used for poison message identification in the real communication network. The details of using neural network classifiers are reported in Subsection 3.3.

The output of passive diagnosis is a probability distribution of each suspect message type being the poison message.

2.1. Examples of Poison Message Failure

We have mentioned the poison message failure incidents of AT&T and the ISP in Section 1. The poison message failures could happen in many protocols and scenarios. In our OPNET [12] simulation testbed, we have constructed three scenarios where Label Distribution Protocol (LDP), OSPF, and BGP are the responsible protocols carrying the poison messages. The first example is the LDP case. LDP is a core protocol in MPLS, and it defines the mechanism for distributing label bindings among label switching routers. In the LDP case, for some reason (e.g., software bug) a LDP Label Request message can cause a receiving node to fail with some probability. Consider in a MPLS network, several dynamic Label Switched Paths (LSPs) are set up. The term 'dynamic path' means that the path from ingress router to egress router is not fixed, but is found dynamically by a routing protocol. If any link or router along the path fails, the ingress router will try to find another path to the egress router.

Suppose an ingress router R_0 wants to set up an LSP to an egress router R_e. First, R_0 sends a Label Request message to next router R_1. Because of the poison message, router R_1 fails with some probability P. If R_1 does not fail, then it sends the Label Request message to the next router R_2, and with some probability R_2 will fail, and so on. If any router along the LSP fails, R_0 will try to find another path to R_e, and this may cause some other nodes to fail. After a failed router reboots, it may be failed again by the poison message. If there are many dynamic LSPs in the network, and if the node failure probability P is large enough, then a lot of routers fail and this can cause sustained network instability.

In the OSPF example, the poison message is a LSA message. When a router receives the poison LSA message, it fails with some probability. If it fails, later its neighbors will find the failure and send out LSA messages to other routers to update the network topology. This may cause some other routers to fail, leading to a chain reaction of failures. After a failed router reboots, it may receive the poison LSA messages again, and its failure may be repeated. This also causes an unstable network.

The third example involves BGP. The poison message is a normal BGP update message. In the BGP case, the failure propagation is similar to the OSPF case. A router fails because of a BGP poison message; later other BGP speakers will discover the failure and send out update messages. That could cause other BGP speakers to fail leading to failure propagation among BGP speakers throughout the network.

In order to validate passive diagnosis, we have implemented an OPNET testbed (Figure 1) to simulate an MPLS network in which poison messages can be carried by BGP, LDP, or OSPF. Different probabilities of a poison message failing a router (i.e., node failure probabilities) have been tested. The testbed has 14 routers of which 5 are Label Edge Routers and 9 are (non-edge) Label Switching Routers. We use numbers 1, 2, . . . , 14 to denote these routers in the simulation. We describe the details of the simulations below.

2.2. The Correlating Message Approach

We designed BGP simulation to validate the correlating message approach. As mentioned in Section 1, one

DOI:10.1002/sec

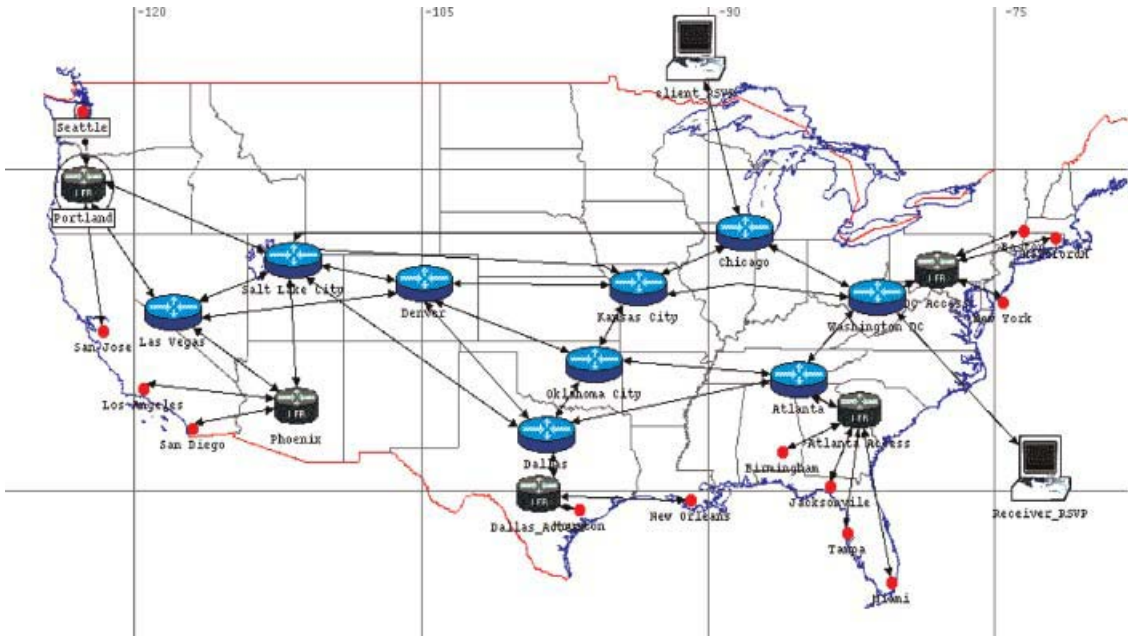

Fig. 1. The topology of the OPNET testbed.

possibility of the poison message failure is that the final message of a protocol is the poison message. That is, there are no more messages exchanged in the protocol between the time the node received the poison message and when it failed. If the node fails shortly after receiving the poison message, this is likely to be the case. However, just because multiple failed nodes have the same final message of a particular protocol does not guarantee that this is the poison message type. For certain protocols, a large proportion of the exchanged messages may be of a particular type. Consequently, when a node fails it is likely that this message type will be the final one observed in that protocol prior to node failure even if it is not the cause of the node failure. We address this issue by using Bayes' Rule to compute the posterior probability of a message being poison given that several nodes have the same type of final message.

First, we need the prior distribution of the final messages. To obtain the prior distribution, we run simulations where the poison message type is sampled from a uniform distribution and we record the relative frequency of each message type being the final message in the protocol. This relative frequency gives prior probability of being the final message.

In the BGP scenarios, the poison message is a normal BGP message. For each simulation, the poison message is either an update message, keep alive message, or open message. Different node failure probabilities have been tested. Results from a typical simulation run with BGP keep alive message being the poison message are given in Table I. The data in the last three rows are the final messages in each protocol. We use number $(1, 2, \ldots)$ to denote message types in all tables. The notations of message types are: BGP: open $= 1$, update $= 2$, keepalive $= 3$; OSPF: hello $= 1$, request $= 2$, update $= 4$, ack $= 5$; LDP:

Table I. BGP simulation data.

Failed nodes	12	7	4	11	5	9	6	13	14	3
Failure time (seconds)	137.0	137.01	137.02	137.03	200.1	200.2	200.3	242.1	242.1	242.2
BGP	3	3	3	2	3	3	3	3	3	3
LDP	No	No	No	No	6	No	No	No	No	No
OSPF	1	1	4	1	1	1	4	1	1	1

DOI:10.1002/sec

init = 2, keepalive = 3, mapping = 4, request = 5, hello = 6, release = 7.

For each node failure, there is a small random delay between its receipt of the poison message and failure. Because of the small random delay, not all failed nodes have the same final message in BGP. For example, when node 11 failed the first time, the final message in BGP is an update message rather than keep alive message. However, there is a high probability that the final message is the same as the poison message.

2.2.1. Calculating posterior probability

From Table I, we observe that many failed nodes do not have LDP messages, so we can rule out LDP for sure, that is, LDP is not the protocol carrying the poison message. Now we only need to consider BGP and OSPF. Most of the failed nodes have the same BGP final messages—BGP keep alive message. Also most final messages in OSPF are hello messages. We use Bayes' Rule to compute the posterior probability given the above final messages.

p(keep) is used to denote the prior probability of {keep alive message is the final message in a failed node}. When the poison message failure happens, a node may fail immediately or in a short time after receiving a poison message. If a node fails immediately, then the poison message is the final message in the protocol. If a node fails with some delay, then any message could be the final message. Based on the above fact and for the simplicity of computation, we assume that the distribution of final messages at different nodes is independent and identical. Then, we can use the multinomial distribution to get the probability of N nodes having certain final message distribution. Recall that a multinomial distribution has the following probability mass function (pmf): $P(r_1, r_2, \cdots, r_n) = k * p_1^{r_1} p_2^{r_2} \cdots p_n^{r_n}$, where r_j is the number of occurrences of outcome type j, p_j is the probability that an individual outcome takes the value j, and $\sum p_j = 1$. The multinomial coefficient $k = N!/(r_1!r_2! \cdots r_n!)$, and $N = r_1 + r_2 + \cdots + r_n$.

Considering node failure time in Table I, we can regard the four nodes—nodes 12, 7, 4, and 11 as the first batch of failed nodes. Applying the multinomial distribution to those four nodes in BGP ($N = 4$), we have,

$$\begin{aligned} &\mathrm{P}(\text{keep}, \text{keep}, \text{keep}, \text{update}) \\ &= 4!/(3!1!) * p(\text{keep})^3 * p(\text{update}) \\ &\equiv P_a \end{aligned} \tag{1}$$

where p(keep) and p(update) are the prior probabilities that we obtained from simulations. We use data from other simulations when the BGP keep alive message is the poison message to compute the following conditional probabilities:

P(keep alive message is the final message | BGP keep alive message is poison) $\equiv P_{b1}$.
P(update message is the final message |, BGP keep alive message is poison) $\equiv P_{b2}$

Next, we want to obtain the probability

$$\begin{aligned} &\mathrm{P}(\text{keep}, \text{keep}, \text{keep}, \text{update} | \text{BGP keep alive message} \\ &\text{is poison message}) \equiv P_b \end{aligned} \tag{2}$$

Here, we make a simplifying assumption: given BGP keep alive message is poison, the distribution of final messages at different nodes is independent. Then, we can calculate P_b by the multinomial distribution and P_{b1}, P_{b2}.

Next, consider the prior probability for each message type of being the poison message. Since we have no prior knowledge which message is the poison one, we use a uniform distribution over all message types for all protocols. That is, the prior probability is

$$\begin{aligned} &\mathrm{P}(\text{BGP keep alive message is poison}) \\ &= \mathrm{P}(\text{OSPF hello message is poison}) = \cdots \equiv \partial \end{aligned} \tag{3}$$

Then by Bayes' Rule, we have

$$\begin{aligned} &\mathrm{P}(\text{BGP keep alive message is poison} \,|\, \text{keep, keep, keep, update}) * \mathrm{P}(\text{keep, keep, keep, update}) \\ &= \mathrm{P}(\text{keep, keep, keep, update} \,|\, \text{BGP keep alive message is poison}) * \mathrm{P}(\text{BGP keep alive is poison}) \end{aligned}$$

Combining Equations (1), (2), (3), we have

$$\begin{aligned} P_1 \equiv \mathrm{P}(&\text{BGP keep alive message is poison} \,| \\ &\text{keep, keep, keep, update}) = \partial * P_b / P_a \end{aligned} \tag{4}$$

Similarly, we can obtain the posterior probability for other message types. Since we assume there is only one kind of poison message, we have $\sum_i P_i = 1$. Using the above procedure, we calculated the poster-

DOI:10.1002/sec

Table II. The posterior probability in BGP simulation.

Poison Message	BGP open	BGP update	BGP update
Number of failed nodes	4	5	10
BGP open	**0.987**	0.000	0.000
BGP keep alive	1.1×10^{-10}	0.002	0.008
BGP update	0.012	**0.497**	**0.577**
OSPF hello	0.001	0.478	0.392
OSPF update	8.3×10^{-8}	0.023	0.030

ior probabilities for each suspect message type. The results show that when BGP keep alive message is the poison message, data from only four failed nodes can generate a good probability distribution about the poison message (i.e., the actual poison message type has a high probability), but in other cases one needs to collect more information (i.e., wait for more failed nodes) in order to obtain a good probability distribution, for example, when BGP update message is poison. We also ran simulations where the poison message is BGP open message or update message. After similar calculation, we have the following results listed in Table II.

In Table II, row 1 is the identity of the poison message in each simulation. Row 2 is the number of failed nodes used in calculation, and data in rows 3 through 7 are the posterior probabilities of different message types. We can see that the BGP open message has a very high posterior probability (0.987) when open message is the poison one. This is because the prior probability of having an open message as the final message is very small (0.34%). The posterior probability also depends on how many failed nodes are included in the calculation. Column 3 indicates that when BGP update message is the poison message and only five failed nodes are considered, the probability of OSPF hello message being poison is very close to that of BGP update message, so at this point it is not clear which message is the poison one. If we wait for more nodes to fail, then more information will be collected and used to calculate the posterior probability. Column 4 shows that when data from 10 failed nodes are used, the probability of BGP update message is much larger than that of OSPF hello and other message types.

2.2.2. LDP and OSPF simulations

We also implemented and tested LDP and OSPF simulations, and we have similar results from these simulations. LDP simulation is also used to verify the FSM method. We implemented a FSM error in the LDP simulation, and the FSM error can be found by matching the message sequence with LDP FSM model. Even if we do not use the FSM information and just use the final message distribution information as in Subsection 2.2.1, we can still find out that LDP is the responsible protocol. The details of the FSM approach can be found in Reference [11]. Our OPNET simulations demonstrate that the correlating message approach and FSM approach are effective tools to identify poison message. In the next subsection, we discuss how to use a neural network to identify poison message.

2.3. The Neural Network Approach

In this subsection, we describe how to use neural networks to identify the poison message based on node failure pattern, that is, the sequence of node failures. The neural network approach combines offline learning and online inference techniques. We use the Neural Network Toolbox in MATLAB to design, implement, and simulate neural networks. We implement two kinds of neural networks in our simulation: (1) feedforward backpropagation; and (2) radial basis.

2.3.1. Neural network structure and training

We have implemented three feedforward backpropagation neural networks and one radial basis neural network. They have similar structure. All of them have three layers.

(1) Input layer with 28 inputs:
There are 14 nodes in the communication network. We use a 14-element vector to denote the node status at time k: $S_k = [s_k^1, s_k^2, \cdots, s_k^{14}]$, where k is the discrete time step, and $s_k^m = 0$ or 1 (0 means this node is normal, and 1 means this node is failed). The 28 inputs represent the status of 14 nodes at two consecutive time steps ($k-1$ and k) $- S_{k-1}$ and S_k.
(2) Hidden layer:
In the middle of the three layers is the hidden layer. There is a transfer function in the hidden layer. Many transfer functions are implemented in MATLAB. In our simulation, we use three kinds of transfer functions for the feedforward backpropagation neural networks:

a. *tansig*—Hyperbolic tangent sigmoid transfer function.

DOI:10.1002/sec

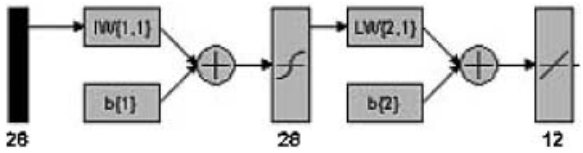

Fig. 2. The structure of a neural network.

b. *purelin*—Linear transfer function.
c. *logsig*—Log sigmoid transfer function.

(3) Output layer with 12 outputs:

The output is the probability distribution vector of the poison message. Twelve outputs represent 12 suspect message types in the OPNET testbed. Each output is the probability of the corresponding message being the poison one. The structure of such a neural network is given in Figure 2.

From the OPNET simulations, we record the node status vector S_k at time k, for $k = 1, 2, \ldots, K$. Then, we use these data to train the neural networks. The input of the neural networks is a 28-element vector representing S_{k-1} and S_k. We use 100 sets of such input data to train the neural networks. The target of the training (i.e., the output during training) is a 12-element vector representing the probability distribution vector of the poison message. Since we know what the poison message is during simulations, we know the target vector. For example, when the 4th message is the poison one, the target vector is [000100000000]. We set the training goal to be: error $< 10^{-10}$. The training epoch number is set as 50. One epoch means that the training data are used once. In our simulations, all neural networks meet the training goal.

2.3.2. Main test results

After training the neural networks, we generate new data to test them. In our simulation, we use 17 sets of 28-element vectors as input to test the four different neural networks. The test results are very good. All of the neural networks can output a good probability distribution of the poison message for an average of 14 out of the 17 input data. Table III lists four sets of test results, where each set of the result is the output of the four neural networks when a particular message serves as the poison message.

In Table III, 'NN' means neural networks. In the 1st row, NN $1 \sim 4$ represent the four neural networks, where 1, 2, and 3 represent three feedforward backpropagation neural networks with transfer function being: 'tansig,' 'purelin' in 1, 'tansig,' 'tansig' in 2, and 'tansig,' 'logsig' in 3, and NN 4 represents the radial basis neural network.

Table III. The outputs of neural networks.

	NN 1	NN 2	NN 3	NN 4
Poison message # 1	0.0841	**0.1953**	**0.9992**	**0.8212**
	0.1235	0.1273	0	0.1127
	0.1146	0	0	0.0078
	0.0427	0.0191	0.0006	0.0322
	0	0.1035	0.0002	0.0262
	0.0892	0.0794	0	0
	0.0898	0.0870	0	0
	0.1331	0.0780	0	0
	0.1269	0.0805	0	0
	0.0446	0.0844	0	0
	0.1143	0.0743	0	0
	0.0373	0.0711	0	0
Poison message # 2	0.0140	0.1302	0	0.0796
	0.1791	**0.1925**	0.0067	**0.3986**
	0.1148	0.0656	0	0.1599
	0	0	0	0.0032
	0.1514	0.1919	*0.9933*	0.3586
	0.0937	0.0471	0	0
	0.1149	0.0793	0	0
	0.1237	0.0375	0	0
	0.1242	0.0844	0	0
	0.0022	0.0519	0	0
	0.0499	0.0488	0	0
	0.0320	0.0708	0	0
Poison message # 3	0.0348	0.1247	0.0061	0.0475
	0	0	0	0.0298
	0.1677	**0.1290**	**0.7421**	**0.6973**
	0.0413	0.1227	0.0000	0.0010
	0.1128	0.1281	0.2518	0.2244
	0.0745	0.0496	0	0
	0.0515	0.0876	0	0
	0.0984	0.0373	0	0
	0.0919	0.1057	0	0
	0.1172	0.0464	0	0
	0.1782	0.0646	0	0
	0.0317	0.1043	0	0
Poison message # 4	0.0003	0.0885	0	0.0096
	0.0132	0	0	0.0032
	0.3187	0.0711	0	0.0032
	0.0777	**0.1551**	**0.9999**	**0.9735**
	0.0388	0.1549	0.0001	0.0105
	0.1162	0.0735	0	0
	0.1164	0.0819	0	0
	0.1452	0.0731	0	0
	0.0738	0.0812	0	0
	0.0653	0.0785	0	0
	0.0345	0.0676	0	0
	0	0.0746	0	0

The original outputs of the neural networks include both positive and negative numbers. We call the outputs the 'distribution scores.' The probability distribution of the poison message is a linear transformation and normalization from the distribution scores, that is, $y = a(x + b)$, where x is the distribution score and y is

DOI:10.1002/sec

the probability distribution, and a and b are parameters. During the neural network training, the output of non-poison message is set to zero, thus we set the transformation of the smallest (negative) distribution score in the output vector to be zero. That is, set b to be $-x_0$ where x_0 is the smallest distribution score, then a can be determined using $\sum y = 1$. The data in Table III are the probability distribution after transformation.

In Table III, the bold numbers are the probabilities of the actual poison message. An italicized number means that the neural network assigns the largest probability to a wrong message type (not the actual poison message). From Table III, we observe that: poison messages # 1, 3, and 4 are correctly diagnosed by neural networks 2, 3, 4 but misdiagnosed by 1. Poison message # 2 is correctly diagnosed by neural networks 1, 2, 4 but misdiagnosed by 3. Table III shows that for most tests, the neural networks can provide a good probability distribution about the poison message, that is, assigning the largest probability to the actual poison message. Comparing all the test results, we find that the radial basis neural network performs the best. Also, we find that different neural networks fail for different cases. This suggests that better results may be obtained by combining the outputs of two (or more) neural networks.

2.3.3. Serial test

In the previous tests, we only use node status at two time steps S_{k-1} and S_k as input to neural networks. Another way to utilize neural networks is to input a series of node status, for example, $S_1, S_2, \cdots, S_{k-1}$, and S_k. That is, we want to input more information to a neural network in the hope of getting better results.

In particular, we carried out the serial tests for the data sets that the neural network failed to correctly diagnose with the original input. The results of the serial tests are encouraging. The neural network can gradually identify the poison message for about $60 \sim 70\%$ of these data sets. It means that the output gradually changes from a bad probability distribution to a good probability distribution about the actual poison message. One example of the serial test is presented in Table IV. In this example, the neural network is a feedforward backpropagation neural network. For each test, we input all the node status up to the current time step. In Table IV, the 1st column is the index of the 12 message types. Columns 2 through 7 are the probability distributions for different inputs. In this test, message # 3 is the poison message.

Table IV. Results from serial test.

	S_1, S_2	$S_1 \sim S_3$	$S_1 \sim S_4$	$S_1 \sim S_5$	$S_1 \sim S_6$	$S_1 \sim S_7$
1	*0.1296*	0.0633	*0.1558*	0.0339	0.1231	0.0811
2	0	0.0056	0.1032	0	0	0.1191
3	0.0743	0.1267	0.1101	**0.1635**	**0.1726**	**0.1461**
4	0.1231	*0.1942*	0.0725	0.0403	0.1205	0.0412
5	0.1258	0.1217	0.1228	0.1100	0.1221	0
6	0.0816	0.0888	0.0937	0.0727	0.0570	0.0860
7	0.0975	0.1145	0.0774	0.0502	0.0664	0.0866
8	0.0569	0.0185	0.0398	0.0960	0.0499	0.1283
9	0.0983	0.0491	0	0.0896	0.0822	0.1224
10	0.0642	0.1107	0.1337	0.1143	0.0473	0.0431
11	0.0652	0	0.0142	0.0961	0.0684	0.1102
12	0.0835	0.1070	0.0769	0.1333	0.0906	0.0359

From Table IV, we observe that at the beginning, the neural network assigns message No. 1 the largest probability, which means that it does not find the poison message. Starting from S_5, the neural network assigns the largest probability to the poison message—message # 3. The neural network consistently finds the poison message in the later tests. The results show that the outputs of the neural network stabilized after time step 5. We also conducted serial tests on the radial basis neural network and got similar results.

Both the main tests and the serial tests show that neural network is an effective tool for identifying the poison message. Next, we discuss active diagnosis where active actions are taken to facilitate the diagnosis.

3. Active Diagnosis

From the passive diagnosis, we have a probability distribution over the possible poison message types. In active diagnosis, message filtering is used to block suspect message types or protocols. Message filtering can be a valuable tool in helping to identify the culprit message type. For example, if a single message type is blocked and the failure propagation stops, this provides strong evidence that the blocked message type is the poison message. On the other hand, if the propagation continues, that message type can be ruled out. Message filtering is implemented at each node. When the central manager decides to filter out a certain message type M, it broadcasts the decision to all nodes in the network, and then each node will filter (block) message M, that is, each node discards the message type without processing it.

DOI:10.1002/sec

In addition to its use as a diagnostic tool, filtering offers the possibility of interrupting failure propagation while the poison message type is being identified. For example, all suspect message types can be initially blocked to stop the failure propagation. Then message types can be turned on one-by-one until propagation resumes. While this approach may be attractive in preventing additional node failures during the diagnostic process, disabling a large number of control messages may result in unacceptable degradation of network performance. Consequently, the decision making for message filtering must take into account tradeoffs involving the time to complete diagnosis, the degradation of network performance due to poison message propagation for each of the suspect message types, and the cost to network performance of disabling each of those message types. Each decision on message blocking leads to further observations, which may call for changing the messages being blocked. This suggests that policies for dynamic filter configuration may be obtained by formulating and solving a sequential decision problem [13,14]. We discuss the sequential decision problem in Subsection 3.1, and we formulate it as a MDP in Subsection 3.2.

3.1. The Sequential Decision Problem

The sequential decision problem of message filtering is described below:

- At each step, the state consists of the recent history of node failures and a probability distribution vector with one component for each suspect message type.
- Based on the current state, a decision (action) is made as which message types to block.
- When new node failures are observed, the state is updated based on the current state, action, and new observation.
- Actions are chosen according to a policy that is computed off-line based on optimizing an objective function taking into account the following factors:
  - degree to which each action will help in isolating the responsible message type,
  - urgency of diagnosis under each potential message type, for example, if a suspect protocol sends its messages to a large number of nodes *via* flooding, the risk of network instability would be particularly great if the messages are poison, and this risk provides additional impetus for filtering such a message type,
  - impact of blocking on network performance. A critical protocol or message type should be blocked only if there is a compelling need to do so.

There are three possible outcomes when message filtering is deployed:

(1) If message filtering is used and the propagation is stopped within a certain time, then either the poison message type is found (if only one message type is blocked) or the message set that includes the poison message) is known (if two or more types are blocked).
(2) If message filtering is used but the propagation is not stopped within a certain time, then the responsible message type is not found. The filtered message types are removed from the possible suspect set. New information (e.g., new node failures) is collected and used to update the probability distribution and reconfigure the filters.
(3) If the current action is not to block any message type, then we simply take another observation. Some active nodes may fail as the poison message propagates in the network, and the messages exchanged with these failed nodes can be retrieved. This information is used to update the probability vector. Based on the updated state, a new decision is made and the filters are reconfigured.

We model the above sequential decision problem as an MDP. The details are given below.

3.2. The Markov Decision Problem

We define the MDP as below: there are N control message types in the network, which are indexed as $[1, 2, \ldots, N]$. One of the types is faulty. We want to find an optimal policy to identify the faulty message type and minimize the expected overall costs.

The initial state of the MDP is a probability vector P_0 (from passive diagnosis) whose components give the probability for each message type being poison, and current node status. We assume a node will either fail in a short time after it receives a poison message or not fail by this poison message. That is, we exclude the possibility that a node becomes 'infected' by a poison message but fails much later. Based on this assumption, a node status can be only one of two

DOI:10.1002/sec

states (normal or failed), which means the node status can be fully observed.

We formulate the problem as a stochastic shortest path problem [15]. The horizon is finite, but its length is random and may be affected by the policy being used. There are cost-free termination states. The termination states are the states with one of the probabilities larger than $(1-\varepsilon)$, where ε is a given threshold close to 0, and all other probabilities are close to 0, which means that the poison message type is found with a desired confidence level (i.e., $1-\varepsilon$). The terminal cost is $J_t = 0$, since the poison message type is found.

The objective is to find a stationary policy μ that minimizes an expected total cost. The total cost (or the objective function) is given in Subsection 3.2.2. A policy is a mapping that tells one which action to take for a given state. Stationary policy means that the mapping does not change over time. We formulate the MDP as follows.

3.2.1. The state space of the MDP

First, we introduce the notations. In the following, k is the discrete time index. A vector is used to denote the status of each node in the network: $S_k = [s_k^1, s_k^2, \cdots, s_k^M]^T$, where M is the total number of nodes in the network, and $s_k^m = 0$ or 1 (0 means normal; and 1 means failed). Let d_k be the random variable representing the identity of the poison message type. $d_k = j$ means the poison message type is j at time k, where $j = 1, \ldots, N$, and N is the total number of message types used in the network. Since the poison message type does not change over time, we have: $P(d_{k+1} = j \mid d_k = i) = 1$, only if $j = i$; otherwise, $P(d_{k+1} = j \mid d_k = i) = 0 \quad (*)$.

Define the information vector up to time k as $I_k = (Z_0, \cdots, Z_k, U_0, \cdots, U_{k-1})$, where U_k, Z_k are the action and observation at time k, respectively. Z_k is the set of newly failed nodes at time k. Possible actions (controls) U_k at time step k are: (1) turning off one message type; (2) turning off multiple message types; (3) do nothing. Note: the actions may also include some tests that have less cost than turning off a message type. However, we do not consider those tests in this paper.

$P_k = P(d_k \mid I_k)$ is the conditional probability vector of the poison message. $P_k = [p_k^1, p_k^2, \cdots p_k^N]^T$, where p_k^j is the conditional probability of message type j being the poison one given the observation history up to time k, that is, $p_k^j = P(d_k = j \mid I_k)$. P_k is a sufficient statistic that contains all information (except node status information) embedded in the history process for control [15].

We make a simplifying assumption that the order of failed nodes does not matter in determining which nodes will fail at next time step. We choose the system state as $[S_k, d_k]$, and the action is given above, then it is a Partially Observed Markov Decision Process (POMDP), since part of the state d_k (the identity of the poison message type) is not observable. Instead, we reformulate the problem as a completely observed MDP by using the sufficient statistic P_k [15]. This formulation is given below:

State: The system state is $X_k = [S_k, P_k]$.

Action: $U_k = [u_k^1, u_k^2, \cdots u_k^N]^T$. If message type j is turned off at time k, then $u_k^j = 1$; otherwise $u_k^j = 0$.

3.2.2. The cost function

There are three kinds of costs:

(1) There may be a cost when message type j is turned off—g_1^j. If the message type turned off is the poison one, then there is no penalty and the cost is zero, otherwise the cost is g_1^j.
(2) There is a cost when a node i is failed—g_2^i. This cost may be different for different nodes. We assume that the total cost is additive over the failed nodes. It would be more realistic, but computationally more difficult, to allow the cost depend on the set of failed nodes. We do, however, take into account whether the set of failed nodes results in a partition of the network.
(3) There is a cost when the network is partitioned by the failed nodes—denoted the cost as g_3.

Thus, the cost function at each time step k is

$$\begin{aligned} g(X_k, U_k) &= g(S_k, P_k, U_k) \\ &= f_1(P_k, U_k) + f_2(S_k) + f_3(S_k) \end{aligned}$$

where f_1 is the cost of blocking one or more message types, f_2 is the cost of node failures, and f_3 is the cost of network partition. Define $N_k = \{j \mid u_k^j = 1,\ 1 \le j \le N\}$ as the index set of all the message types being turned off at time k. A simplified cost function is given below:

$$g(X_k, U_k) = \sum_{j \in N_k} (1 - p_k^j) \times g_1^j + \sum_{i=1}^{M} s_k^i \times g_2^i + g_3 \times L_k$$

where $L_k = 1$ if the network is partitioned; otherwise $L_k = 0$. Note that $s_k^i = 1$ means node i is failed at time k. The cost of filtering a particular message type j (i.e., g_1^j) is weighted by $(1 - p_k^j)$, which is the current

DOI:10.1002/sec

conditional probability that message type j is *not* the poison one.

3.2.3. The objective function

We want to find a stationary policy μ that minimizes the following expected total cost (objective function):

$$J^{\mu}(X_0) = \lim_{N\to\infty} E\left\{\sum_{k=0}^{N-1} g_k(X_k,\mu(X_k))\right\}.$$

This is the expected cost from initial state X_0 to a terminal state, that is, a state in which the poison message type has been identified with the required degree of confidence.

3.2.4. The state transition probability

Given state X_k and action U_k, we want to know the state transition probability to reach next state $X_{k+1}: P(X_{k+1}\,|\,X_k, U_k) = P(S_{k+1}, P_{k+1}\,|\,S_k, P_k, U_k)$. The state X_k has two parts S_k and P_k. We derive the update of each part below.

Update of S_k. Assume there is no reboot of failed nodes during the short period of active diagnosis. The update of S_k is given below:

$$\begin{aligned}
&P(S_{k+1}|S_k, P_k, U_k)\\
&= P(S_{k+1}|S_k, I_k, U_k) \quad (P_k \text{ is sufficient statistic of } I_k)\\
&= \sum_j P(S_{k+1}|S_k, U_k, d_k = j) \times P(d_k = j\,|\,I_k)\\
&= \sum_j P(S_{k+1}|S_k, U_k, d_k = j) \times p_k^j
\end{aligned}$$

Note: $P(S_{k+1}\,|\,S_k, U_k, d_k = j) \triangleq P_{k+1,j}$ can be determined from simulations and online estimation. The details are given in the sequel.

Update of P_k. For each message type j, where $j = 1, \ldots, N$, we have,

$$\begin{aligned}
p_{k+1}^j &= P(d_{k+1} = j\,|\,I_{k+1})\\
&= P(d_{k+1} = j\,|\,Z_0, \cdots, Z_k, Z_{k+1}, U_0, \cdots, U_{k-1}, U_k)\\
&= P(d_{k+1} = j\,|\,S_0, \cdots, S_k, S_{k+1}, U_0, \cdots, U_{k-1}, U_k)\\
&= P(d_{k+1} = j\,|\,I_k, S_{k+1}, U_k)\\
&= P(d_{k+1} = j, S_{k+1}\,|\,I_k, U_k)/P(S_{k+1}\,|\,I_k, U_k)\\
&= \sum_{i=1}^{N} \{P(d_k = i\,|\,I_k) \times P(d_{k+1} = j\,|\,d_k = i, U_k) \times P(S_{k+1}\,|\,I_k, U_k, d_k = i, d_{k+1} = j)\}/P(S_{k+1}\,|\,I_k, U_k)\\
&\text{(from } (*), \text{only when } i = j,\ P(d_{k+1} = j\,|\,d_k = i, U_k) = 1; \text{otherwise it is 0.)}\\
&= P(d_k = j\,|\,I_k) \times P(S_{k+1}\,|\,I_k, U_k, d_{k+1} = j)/P(S_{k+1}\,|\,I_k, U_k)\\
&= \partial_2 \times p_k^j \times P(S_{k+1}\,|\,I_k, U_k, d_{k+1} = j)
\end{aligned}$$

where $\partial_2 = 1/P(S_{k+1}\,|\,I_k, U_k)$ is a normalization constant independent of j. Thus, we have

$$p_{k+1}^j = \partial_2 \times p_k^j \times P(S_{k+1}\,|\,I_k, U_k, d_{k+1} = j)$$

Note: $P(S_{k+1}\,|\,I_k, U_k, d_{k+1} = j)$ can be approximately determined from simulations.

Determine the Probability $P_{k+1,j}$. Recall that $P(S_{k+1}\,|\,S_k, U_k, d_k = j) \triangleq P_{k+1,j}$, that is, the probability of node status being S_{k+1} at time $k+1$, given the poison message type is j, node status vector is S_k and control is U_k at time k.

We define: $p_1 =$ Prob. (node i receives the poison message j | poison message is j, node status vector is S_k and control is U_k), and $p_2 =$ Prob. (a node fails | receiving the poison message j), then we have $P(s_{k+1}^i = 1\,|\,S_k, U_k, d_k = j) = p_1 \times p_2$.

We have the following observations about p_1 and p_2.

(1) p_1 depends on the network topology and how message type j is used in the network, etc. p_1 can be determined from simulation.
(2) p_2 is independent of node status vector S_k and independent of receiving node i. p_2 can be estimated online based on failed node history and

DOI:10.1002/sec

current probability distribution P_k. The details are given below.

Online Estimation of p_2. In most cases, the probability p_2 is unknown. We propose to estimate p_2 by using an online estimation technique that utilizes the node failure sequence observed online. We change the notation to be an estimation context as follows.

Given observation of failed nodes $y = \mathrm{H}$, we want to estimate $\theta = p_2$. We use a popular estimation method—Maximum A Posterior (MAP) estimation, which finds the parameter θ that maximizes $p(\theta \mid y, d_k = j)$. We have

$$\begin{aligned} &\max_{\theta} p(\theta \mid y, d_k = j) \quad \text{(by Bayes' rule)} \\ &= \max_{\theta} p(y \mid \theta, d_k = j) \times p(\theta \mid d_k = j) / \int_{\theta} p(y \mid \theta, d_k = j) \times p(\theta \mid d_k = j) d\theta \end{aligned} \qquad (**)$$

Since we have no prior information about θ, we assume $p(\theta \mid d_k = j)$ has uniform distribution in [0,1]. The probability $p(y \mid \theta, d_k = j)$ is the distribution of failed nodes given the poison message j and parameter θ, so $p(y \mid \theta, d_k = j)$ can be estimated from simulations. Given $p(\theta \mid d_k = j)$ and $p(y \mid \theta, d_k = j)$, we can compute (**) for any θ, and the MAP estimation outputs the θ that achieves the maximum in (**). The estimated $\theta = p_2$ is then used in active diagnosis—the sequential decision problem. At each time step, p_2 is updated when new failed nodes are observed.

The estimation of p_2 may not be very accurate, but it could be enough for choosing a good suboptimal policy. In general, it is desirable to find a policy that works reasonably well for a range of parameter values (e.g., p_2) since there is inherent parameter uncertainty in this problem. Furthermore, when the number of nodes in the network is large, this sequential decision problem becomes a large POMDP, which makes the exact solution of this problem intractable. So the best one can hope for is to find an approximate solution to the sequential decision problem that is robust to parameter uncertainty.

4. Experimental Results of Active Diagnosis

The state space is very large for the sequential decision problem, and it is impossible to solve the problem using classic MDP solution techniques such as value iteration or policy iteration [15]. One heuristic policy that we have considered is to block a single message type at each decision time step. The message type selected for blocking is the one that has the smallest ratio E[C]/p where the E[C] is the expected cost (in terms of network performance) to block the message type for a time step, and p is the current estimate of the probability that the message type is the poison one. The use of these ratios to order diagnostic tests for network fault management has been discussed in References [13,14]. In this section, we compare the performance of the heuristic policy with another policy obtained by using reinforcement learning [16] and Q-factor approximation [17].

4.1. Q-Factor Approximation

Denote the (optimal) Q-factor at state i with action u as: $Q(i,u) = g(i,u) + E\{J(j)\}$, where $J(j)$ is the minimum achievable cost ^{j}to reach termination starting from state j, that is, cost to go from state j under an optimal policy. (Note that j represents a state here, not a message type as before.) The interpretation is that $Q(i, u)$ is the expected cost of starting in state i, initially taking the action u, and thereafter choosing actions according to an optimal policy. Given the Q-factor, $J(i)$ is obtained by minimizing $Q(i, u)$ over all admissible u. Furthermore, the minimizing u gives the action that would be taken in state i by an optimal policy. Thus, determining $Q(i, u)$ for all state-action pairs (i, u) implicitly specifies the optimal policy.

In our problem, state i consists of the status (normal or failed) of each node and the probability vector of each suspect message type being poison. The number of the node status vectors is exponential in the number of nodes, while the number of values of the probability vector (assuming the space of probabilities is discretized) is exponential in the number of message types. Thus, the problem suffers from the well-known 'curse of dimensionality' for dynamic programming [17].

To overcome this computational problem, we use parametrized Q-factors. Instead of associating a value with each state, one uses a parametric form to

DOI:10.1002/sec

approximate the optimal Q-factor. The parametric form we consider is a linear approximation architecture: $\overline{Q}(i,u,r) = \phi(i,u)^T r$, where $\phi(i,u)$ is called the feature vector and r is the parameter vector.

The Q-factor approximation is expressed as a linear combination of so-called features. Each feature is a (possibly nonlinear) function of (i, u). The coefficients in the linear combination are the components of the parameter vector r. A gradient descent algorithm is used to tune the parameter values to attempt to get a good approximation for the actual Q-factor. It is known that the success of such approximations depends on how the feature vector $\phi(i,u)$ is chosen. The feature vector $\phi(i,u)$ is meant to capture those 'features' of the state i and action u that are considered most relevant to the decision making process. Usually, the feature vector is handcrafted based on available insights on the nature of the problem, prior experience with similar problems, or experimentation with simple versions of the problem.

4.2. TD (λ) Estimation

We use online gradient-based temporal difference TD (λ) method [16,17] for approximating the optimal Q-factor $Q(i, u)$ by optimizing the parameter r. The algorithm is stated below, where z_k is called the eligibility trace [16,17]. It is a vector of the same dimension as r.

1. Initialize r and $z_0 = 0$.
2. Repeat:

   At state i_k, take action $u_k = \arg\min_{u \in U_k} \overline{Q}(i_k, u, r)$ that is ε-greedy to the current Q-factor. That is, most of the time, we choose the greedy action, the action that minimizes the current Q-factor approximation for the current state, but with some small probability ε, we choose a non-greedy (e.g., random) action. This is to ensure sufficient exploration of the state and action space. After the action u_k, the one step cost $g(i_k, u_k)$ and the next state i_{k+1} are recorded. Then we can choose next action—u_{k+1} that is ε-greedy with respect to $\overline{Q}(i_{k+1}, u, r)$.

   Calculate TD:
   $d_k = g(i_k, u_k) + \overline{Q}(i_{k+1}, u_{k+1}, r) - \overline{Q}(i_k, u_k, r)$
   Update z_k:
   $z_k = \lambda z_{k-1} + \nabla_r \overline{Q}(i_k, u_k, r) = \lambda z_{k-1}$
   $+\nabla_r \phi(i_k u_k)^T r = \lambda z_{k-1} + \phi(i_k, u_k)$
   or $z_k = \sum_{m=0} \lambda^{k-m} \phi(i_m, u_m)$
   Update r: $r := r + d_k z_k$

until terminal state.

4.3. Test Results

First, we need to determine the feature functions. Once the features have been chosen, we use the algorithm above to tune the parameters in the vector r to obtain a good approximation for the optimal Q-factor. This approximate optimal Q-factor in turn determines a policy. This is the policy that is greedy with respect to the approximate Q-factor; that is, for a state i, it chooses the action u that minimizes the approximate Q-factor over all pairs (i, u). We then compare the performance of the greedy policy with that of the heuristic policy. We perform two types of comparisons. In the first comparison, we fix the initial state and use Monte Carlo simulation [16] to estimate the expected cost-to-go under each policy. In the second comparison, we allow the initial state to vary. For each initial state, we generate 'parallel' trajectories following each of the two policies, and we compare the cost-to-go under the two policies. To simplify the computations, in this subsection we restrict the action space for the MDP to include only actions that block a single message, that is, we exclude actions that block multiple message types at one time step.

We need to select feature functions for the linear approximation architecture of the Q-factor: $\overline{Q}(i,u,r) = \phi(i,u)^T r$. We first tried linear functions of S_k, P_k, and U_k (defined in Section 3). However, they did not provide good approximation for the Q-factor. Then, we tried quadratic functions of S_k, P_k, and U_k and obtained much better results. There are 14 nodes ($M = 14$) and 6 different message types ($N = 6$) in our OPNET simulation testbed. After considerable trial and error, we find a good approximation architecture for the Q-factor to be

$$\begin{aligned}&\overline{Q}(i_k, u_k, r)\\ &= \overline{Q}(S_k, P_k, u_k, r) = r_0 + \sum_{l=1}^{M} s_k^l r_l + \sum_{j=1}^{N} p_k^j r_{j+M}\\ &\quad + \sum_{f=1}^{N} \{u_f^k r_{f+M+N} + u_f^k p_k^f r_{f+M+2N} + u_f^k h(f) r_{f+M+3N}\}\end{aligned}$$

Thus, there are totally $1 + M + N + 3N = 1 + 14 + 6 + 6*3 = 39$ elements in the parameter vector r.

4.3.1. Comparison of Q-factor

Before applying the learning algorithm to tune the components in the parameter vector r, we need to determine the cost functions g_1, g_2, and g_3 (defined in

DOI:10.1002/sec

Table V. The cost of each node being failed.

Node	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Cost	60	40	70	60	55	50	80	70	65	50	90	75	90	70

Section 3). Ideally, we could extend our control plane simulation model to include the data plane as well, and then we could use it to evaluate the actual cost to network performance (e.g., throughput) resulting from blocking message types, node failures, and network partitions. Since our current simulator does not include the data plane, we simply assigned cost to each node according to the location, functionality of the node, and the network topology in the testbed. We assigned the cost of turning off a message type according to the importance of the message type. The cost for each node failure is given in Table V, and the cost of blocking each message type is given in Table VI.

In our simulations, most of the time the network is not partitioned by failed nodes, so we set $g_3 = 0$ in the tests. Monte-Carlo simulations are used to estimate the Q-factor under the heuristic policy at some typical state action pair (i, u). After using the learning algorithm to tune r, we also use Monte-Carlo method to estimate the Q-factor under the policy which determines u in state i by minimizing $\overline{Q}(i,u,r)$ over u. Then, we compare the Q-factors and cost-to-go under the two policies.

For the heuristic policy μ, the Q-factor is: $Q^{\mu}(i_1,u) = g(i_1,u) + \underset{i_2}{E}\{J^{\mu}(i_2)\}$, and it is evaluated by Monte-Carlo method. $Q^{\mu}(i,u)$ is the Q-factor under policy μ and with state action pair (i, u). $J^{\mu}(i)$ is the cost-to-go under policy μ at state i. This is the expected cost-to-go when action u is taken in the initial state i_1 and the remaining actions are chosen according to the heuristic policy.

We compare the heuristic policy with the policy $\overline{\mu} = \arg\min_u \overline{Q}(i,u,r)$, which is greedy with respect to the optimal Q-factor approximation $\overline{Q}(i,u,r)$. The Q-factor for the greedy policy is given by $Q^{\overline{\mu}}(i_1,u) = g(i_1,u) + \underset{i_2}{E}\{J^{\overline{\mu}}(i_2)\}$ and is also evaluated by Monte-Carlo method. This is the expected cost-to-go when action u is taken in the initial state i_1 and the remaining actions are chosen according to the policy $\overline{\mu}$. We call $\overline{\mu}$ the greedy policy in the following discussion.

To calculate the Q-factors by Monte-Carlo simulations, we need to update the state (S_k and P_k defined in Section 3). To do this, the transition probabilities are needed. For the update of S_k, we obtain the propagation probabilities (p_1 in Section 3) through extensive simulations. The failure probability of each poison message—p_2 (i.e., the probability that a poison message fails a node) is also needed. In these tests, we did not use the online estimation technique mentioned in Section 3 to estimate p_2. Instead, we use the known failure probability that is set in the simulation for simplification. The transition probabilities are computed from the propagation probabilities and the failure probabilities. For the update of P_k, the probability $P(S_{k+1} \mid I_k, U_k, d_{k+1} = j)$ is obtained from simulation.

We start from a typical initial state $i_1 = S_1 = [1, 7, 8, 12]$; $P_1 = [1/6, 1/6, 1/6, 1/6, 1/6, 1/6]$, which is a state with four failed nodes: nodes 1, 7, 8, and 12, and initially the probability distribution is uniform. Then, we estimate the Q-factor under each policy for state i_1 with six different actions. Each Q-factor value is estimated over 20 different simulation trajectories starting from state action pair (i_1, u). The results are shown in Table VII. In Table VII, the 1st row is the action u being six different values. $u = 1$ means that the 1st message type is turned off, and the rest is similar. The 2nd row gives the value of the Q-factors

Table VII. Comparison of Q-factors.

Action u	1	2	3	4	5	6
$Q^{\mu}(i_1,u)$	2248	2367	2433	2079	2127	2254
$Q^{\overline{\mu}}(i_1,u)$	1444	1454	1247	1448	1390	1269
$Q^{\overline{\mu}}/Q^{\mu}(\%)$	64.2	61.4	51.3	69.6	65.4	56.3

Table VI. The cost of turning off each message type

Message	BGP update	BGP keepalive	OSPF hello	LDP hello	LDP keepalive	LDP request
Cost	250	350	500	150	120	120

Table VIII. Comparison of value functions.

Value function	Policy μ	Policy $\overline{\mu}$
$J(i_1)$	2433	1247

DOI:10.1002/sec

Table IX. Comparison of two policies.

Case	1	2	3	4	5	6	7	8	Ave steps
μ	63541**2**	35**2**	3524**6**	34**2**	354**1**	34**1**	35**1**	3524**1**	4.000
$\overline{\mu}$	4**2**	3**2**	21**6**	3**2**	32**1**	2**1**	532**1**	32**1**	2.625

with the first action being u and then following the heuristic policy μ. The 3rd row gives the values of the Q-factors corresponding to the greedy policy $\overline{\mu}$. The last row is the ratio between the Q-factors under the two policies. We can see from Table VII, the Q-factor at the same state action pair (i_1, u) under the greedy policy $\overline{\mu}$ is much less than that under the heuristic policy. As the last row shows, $Q^{\overline{\mu}}$ is only about $50\sim70\%$ of Q^{μ}. This means that the greedy policy is much better than the heuristic policy. Thus, we do find a better policy by Q-factor approximation. In Table VIII, we compare the value function $J(i_1)$ at state i_1 under the heuristic policy and the greedy policy. The value function $J^{\overline{\mu}}(i_1)$ under the greedy policy is only about 50% of the value function $J^{\mu}(i_1)$ under the heuristic policy.

4.3.2. *Comparison of two policies*

We run parallel simulations from the same initial states. One simulation uses the heuristic policy while the other uses the greedy policy. We do this for many initial states. The simulations are constructed to be 'coupled' in that unless one policy blocks propagation and the other does not, exactly the same thing happens in the two simulations.

In Table IX, we present the test results for eight different initial states. The 1st row is the index of the initial states, and each number represents a different initial state. The 2nd row is the action sequence under the heuristic policy μ. For example, the action sequence in case 1 is 63541**2**. It means by following the heuristic policy μ, the first action is 6 (which turns off message type 6), then turns off message type 3, message type 5, message type 4, message type 1, and message type 2. The bold number corresponds to the poison message in the simulation. For example, message type 2 is the poison message in case 1. The 3rd row is the action sequence under the greedy policy $\overline{\mu}$. As we can see from Table IX, the greedy policy $\overline{\mu}$ performs much better than the heuristic policy μ. The greedy policy can find out the poison message in two or three steps, while the heuristic policy takes more steps. According to the eight cases we tested, the heuristic policy μ averaged four steps to find out the poison message, while the greedy policy only takes 2.6 steps on average.

5. Conclusions

We have discussed a particular failure propagation mechanism—the poison message failure, and designed a framework to identify the responsible message type. We proposed passive diagnosis, which includes the FSM approach applied at individual failed nodes, correlating protocol events across multiple failed nodes, and using neural networks to exploit node failure pattern. Simulations showed the effectiveness of passive diagnosis. Passive diagnosis generates a probability distribution of the poison message and the probability distribution is used by active diagnosis. In active diagnosis, message filtering is used to block suspect message types or protocols. We formulated message filtering as a MDP, and obtained a good suboptimal policy for the MDP by using reinforcement learning and function approximation. Extensive simulations demonstrate that the fault management framework is very effective in identifying the poison message failure.

Acknowledgement

This work was partially supported by U.S. DARPA under contract N66001-00-C-8037.

References

1. Labovitz C, Malan R, Jahanian F. Internet routing instability. In *Proceedings of the ACM SIGCOMM*, Nice, France, August 1997.
2. Labovitz C, Malan R, Jahanian F. Origins of pathological Internet routing instability. In *Proceedings of the IEEE INFOCOM*, March 1999.
3. AT&T News Release. AT&T announces cause of frame-relay network outage. April 22, 1998. www.att.com/press/0498/980422.bsb.html
4. Sweeny T, Moozakis C. MCI frame net melts down. *Tech Web*, August 12, 1999.

DOI:10.1002/sec

5. Bolotin VA, Kuhn PJ, Pack CD, Skoog RA. Common channel signaling networks: performance, engineering, protocols, and capacity management. *IEEE Journal on Selected Areas in Communications* 1994; **12**(3): 377–379.
6. Houck DJ, Meier-Hellstern KS, Skoog RA. Failure and congestion propagation through signaling controls. In *Proceedings of the 14th International Teletraffic Congress*, Amsterdam, Netherland, 1994; 367–376.
7. Hung NL, Jacob AR, Makris SE. Alternatives to achieve software diversity in common channel signaling networks. *IEEE JSAC* 1994; **12**(3): 533–538.
8. Katzela I, Schwartz M. Schemes for fault identification in communication networks. *IEEE/ACM Transactions on Networking* 1995; **3**(6): 753–764.
9. Bouloutas A, Hart GW, Schwartz M. Simple finite-state fault detection for communication networks. *IEEE Transactions on Communications* 1992; **40**(3): 477–479.
10. Bouloutas A, Calo S, Finkel A. Alarm correlation and fault identification in communication networks. *IEEE Transactions on Communications* 1994; **42**(2): 523–533.
11. Du X, Shayman MA, Skoog RA. Preventing network instability caused by control plane poison messages. In *Proceedings of the IEEE MILCOM 2002*, Anaheim, CA, October 2002.
12. The OPNET network simulator. www.opnet.com
13. Shayman MA, Fernandez-Gaucherand E. Fault management in communication networks: test scheduling with a risk-sensitive criterion and precedence constraints. In *Proceedings of the IEEE Conference on Decision and Control*, Sidney, Australia, December 2000.
14. Huard JF, Lazar AA. Fault isolation based on decision-theoretic troubleshooting. *Technical Report 442-96-08*, Center for Telecommunications Research, Columbia University, New York, NY, 1996.
15. Bertsekas D. *Dynamic Programming and Optimal Control*, Vols. I and II. Athena Scientific: Belmont, MA, 2000.
16. Sutton R, Barto A. *Reinforcement Learning: An Introduction*. MIT Press: Cambridge, MA, 1998.
17. Bertsekas D, Tsitsiklis J. *Neurodynamic Programming*. Athena Scientific: Belmont, MA, 1996.

DOI:10.1002/sec