Journal of Network and Systems Management, Vol. 14, No. 2, June 2006 (© 2006)
DOI: 10.1007/s10922-006-9027-8

Identifying Control and Management Plane Poison Message Failure by K-Nearest Neighbor Method

Xiaojiang Du [1]

Published online: 22 June 2006

Poison message failure is a mechanism that has been responsible for large-scale failures in both telecommunications and IP networks. The poison message failure can propagate in the network and cause unstable network. In this paper, we apply machine learning, data mining technique in network fault management area. We use k-nearest neighbor method to identify the poison message failure. Also we integrate the k-nearest neighbor method with message filtering approach. We also propose a "probabilistic" k-nearest neighbor method that outputs a probability distribution (rather than the identity) of the poison message. Through extensive simulations, we show that k-nearest neighbor method is very effective in identifying the responsible message type.

KEY WORDS: Fault management; Fault diagnosis; Poison message failure; K-nearest neighbor method .

INTRODUCTION

There have been a number of incidents in which "bad behavior" has propagated from network element to network element and resulted in a significant degradation of network throughput and performance, e.g., routing system failures, routing flaps, congestion and deadlock scenarios, system crash chain reactions, etc. In the past, unintentional system faults and propagating failures have led to Internet routing instabilities [1], major outages of AT&T and MCI-Worldcom frame relay networks [2, 3], Signaling System No. 7 (SS7) common channel signaling network outages of Regional Bell Operating Company in 1990 and 1991, and major service disruption of the AT&T 4ESS^{TM} switched voice telephone network in 1990 [4, 5].

The above incidents were caused by unintentional triggers activating underlying system defects (e.g., software bugs, protocol deficiencies and system design problems) that create the propagation mechanism for instability. An important

[1] To whom correspondence should be addressed at Department of Computer Science, North Dakota State University, Fargo, ND 58105. E-mail: xiaojiang.du@ndsu.edu.

aspect of the system defects is that they are very unlikely to be discovered in normal testing and operation. Since the details of the system defects are not known in advance, effective control mechanisms tailored to the specifics of the vulnerability are virtually impossible to achieve. And new defects are constantly introduced. Furthermore, a serious concern is that we know there can be many such ‘unknown’ software bugs and protocol deficiencies lurking in deployed systems, and if these become known by a malicious party the defects can be exploited to cause significant harm. Thus, the network instability problems should be considered not only as network fault management issues, but also as network security issues.

Previous efforts to address major network outages have focused on areas such as software reliability, disaster prevention and recovery, network topological design, network engineering, and congestion control. In relation to fault propagation, the main idea that has previously been considered is software diversity [6]. However, whenever network providers have studied software diversity they have determined that to achieve it would be too costly and an unmanageable solution. Furthermore, software diversity does not address flaws in a standardized protocol. Virtually all of the previous works 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 control.

Our goal is to develop a fault management framework that can identify the failure and protect networks from unstable behavior when the trigger mechanism and defect causing instability are unknown. There are several generic failure mechanisms that can cause unstable networks [4]. In this paper, we focus on one type of the generic propagation mechanisms. This mechanism can occur in any network that passes control or management messages between network elements, and it involves a ‘poison’ message that propagates and induces system failures, which we refer to as poison message failure.

The Poison Message Failure Problem

We present a description of the generic problem. A trigger event causes a particular network control/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. This activated ‘bug’ can 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, 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 [4, 5]. We are also aware of an incident in which malformed Open Shortest Path First

(OSPF) packets functioned as poison messages and caused failure of the routers in an entire routing area for an Internet Service Provider.

In the AT&T incident above, the trigger event is the normal maintenance event that caused the first switch to take itself out of service. The poison message is the normal out-of-service message sent to neighboring switches informing them that it is temporarily going out of service. The poison message creates a state in the neighboring switches in which faulty code may be executed. In this case, an auxiliary event must occur for the faulty code to be executed causing the node to fail, namely the arrival of a pair of closely spaced call setup messages. The dependence on the auxiliary event makes the node failure probabilistic with the probability depending on network load (the rate of call setup messages). While in this particular example, the poison message itself is not flawed; in other examples such as the OSPF case referred to above, the poison message may be malformed or contain fields with abnormal values. Obviously, the more challenging case is the one in which the message itself is completely normal and is ‘poison’ only because of software defects in the router or switch.

Our objective is to design a fault management framework that can identify the message type, or at least the protocol carrying the poison message, and block the propagation of the poison message to prevent network instability.

The Problem Features

This problem has several differences from traditional network fault management problems. Typical network fault management deals with localized failure. E.g., there is something wrong with a switch. What propagates is not the failure itself but the consequences of the failure on the data plane–e.g., congestion builds up at upstream nodes. Then multiple alarms are generated that need to be correlated to find the root cause. In our problem, the failure itself propagates, and propagation occurs through messages associated with particular control plane or management plane protocols. It is also different from 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 uses flooding so a poison message carried by OSPF link state advertisements is passed to all neighbors. In contrast, Resource ReSerVation Protocol (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.

We make the following two assumptions:

1. The node status information (normal or failed) is available.

2. Because the probability of two message types being the poison message at the same time is extremely small, we assume there is only one message type being the poison message when such failure occurs.

In the paper, we apply a data mining technique - K-Nearest Neighbor (KNN) method to identify the poison message failure. The KNN method can be used in different network management architectures, including centralized Network Management System (NMS), hierarchical NMS, and distributed NMS. To apply KNN method, only the node status information (normal or failed) is needed. We assume the node status information is available to a certain node, which is referred to as the *management node*. The training data for KNN method is stored in the *management node*. In a centralized NMS, the node status information is available to the centralized Network Manager. Distributed network management schemes [17, 19–21] have received increasing attention in recent years. In a hierarchical NMS or a distributed NMS, a distributed Network Manager or Network Element (NE) can be elected as the *management node* to collect node status information and perform the KNN diagnosis. When a node fails, its neighbor can send this information to the *management node* - the distributed Network Manager or NE, then the KNN method can be applied there. Different distributed Network Managers or NEs may serve the role in turn. The detail of the distributed solutions is out of the scope of this paper. In this paper, we will focus on the diagnosis of the poison message failure by using KNN method. In the extreme case, a node (say C) and all its neighbors could fail at the same time, and the information of node C's failure may not be available to the *management node*. In such case, the *management node* can still perform the KNN diagnosis, but the performance may degrade because the information is not complete.

The rest of the paper is organized in the following way. Section 2 gives a brief introduction of passive diagnosis and active diagnosis. Section 3 describes our OPNET simulation testbed and the k-nearest neighbor method. In Section 4, we present various experimental results of using KNN to identify the poison message failure. And Section 5 concludes the paper.

PASSIVE DIAGNOSIS AND ACTIVE DIAGNOSIS

For reader's convenience, we briefly describe passive diagnosis and active diagnosis in this section.

Passive Diagnosis

Passive diagnosis includes several real-time inference and reasoning techniques. It only observes information from the network. Passive diagnosis includes analyzing protocol events at an individual failed node, correlating protocol events across multiple failed nodes, and classifying the observed pattern of failure propagation. The details of the above passive diagnosis approaches can be found in our previous works [4, 7, 9]. The output of passive diagnosis is a probability distribution of each suspected message type being the poison message.

In our previous work, we used neural network approach to identify the poison message in [7, 9]. Here we want to give a comparison between neural network approach and the K-Nearest Neighbor method, which is discussed in this paper. Although they both are adaptive learning methods, we notice there are several differences between these two approaches after applying them to the poison message problem:

(1) A major difficulty in applying neural network approach is that neural networks need large amount of training data. A lot of training data is needed for a neural network to perform well, while KNN only needs relatively small amount of training data. For example, in the poison message problem, more than 500 set of data is needed to train the neural network designed for a 50-node communication network, while only 50 set of training data is needed by KNN for the same communication network. This is a great reduction in the complexity.
(2) Large neural networks may not converge. So the neural network approach may not work well for large communication networks. However, KNN does not have this problem. Our simulations show that KNN performs well for large networks, i.e., it has good scalability.
(3) KNN is a simple algorithm, it is easier to understand and implement in real communication networks than neural network approach. This is an important reason that we still want to investigate KNN method after using neural network approach.

In summary, KNN needs much less training data, has better scalability and robustness than neural network approach. Also, KNN is easier to be implemented in real networks than neural network approach.

Active Diagnosis

Active diagnosis intervenes with the network and changes the dynamic of the network. Active diagnosis can be very effective in many fault identification problems [18]. For the poison message failure, active diagnosis uses message filtering to block suspected message type. From passive diagnosis we have an estimated probability distribution over the possible poison message types. In active diagnosis, filters are dynamically configured to block suspected message types. 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.

In addition to its use as a diagnostic tool, filtering offers the possibility of interrupting failure propagation while the culprit message type is being identified. For example, all suspected message types can be initially blocked to stop 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 or management messages may result in unacceptable degradation of network performance. Consequently, the decision making for filter configuration must take into account tradeoffs involving the time to complete diagnosis, the degradation of network performance (node failure) due to poison message propagation, and the cost to network performance of disabling each of those message types. Each decision on filter configuration leads to further observations, which may call for changing the configuration of the filters. This suggests that policies for dynamic filter configuration may be obtained by formulating and solving a sequential decision problem.

The Sequential Decision Problem

The sequential decision problem is stated in the following.

- At each stage, the state consists of the recent history of the node failures, and a probability distribution vector where each element is the probability of a suspected message type being the poison message.
- Based on the current state, a decision (action) is made as to how to configure filters.
- 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 a proper objective function. There are three possible outcomes when message filtering is used.

(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 filtered) or the types that are not filtered are ruled out (two or more types are filtered).
(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. Collect more information, update the probability vector and reconfigure the filters.

(3) If the current action is not to filter any message types, then we simply take another observation. Several other nodes may fail as the poison message propagates in the network. This information is used to update the probability vector. Based on the updated state, a new action is taken to configure the filters.

The sequential decision problem is modeled as a Partially Observed Markov Decision Process (POMDP) in [8].

K-NEAREST NEIGHBOR METHOD

In this paper, we apply machine learning, data mining [10, 11] technique in network fault management. We use k-nearest neighbor method to identify the poison message failure in communication networks.

The OPNET Simulation Testbed

We have implemented an OPNET testbed to simulate an MPLS network in which the poison message can be a message in Border Gateway Protocol (BGP), Label Distribution Protocol (LDP), or OSPF. The training data for k-nearest neighbor method is generated from the simulation testbed. The testbed has 50 routers, and the testbed topology is shown in Fig. 1.

For each simulation run, one message type in BGP, LDP or OSPF is randomly selected as the poison message. When a node in the network receives the poison message, it fails with certain probability. Simulations are run with different message types being the poison message and different node failure probabilities. For each simulation, the node status vector is recorded at every time step (a pre-defined time interval). After extensive simulations, large amount node status vectors are obtained. And these node status vectors are used as the base data in the k-nearest neighbor method. Table I gives an example of the node status vectors, where the Poison Message is OSPF update message.

The node status vector in Table I looks like a matrix. But actually it is a vector. We just split it into four rows to correspond to the four subnetworks in the testbed. In the node status vector, “1” means the node is failed, and “0” means the node is normal.

K-Nearest Neighbor Method

Here we give a brief introduction about the k-nearest neighbor method. Given a training set M of m labeled patterns, a nearest neighbor procedure decides that some new pattern X belongs to the same category as do its closest neighbors in M. More generally, a k-nearest neighbor method assigns a new pattern X to that

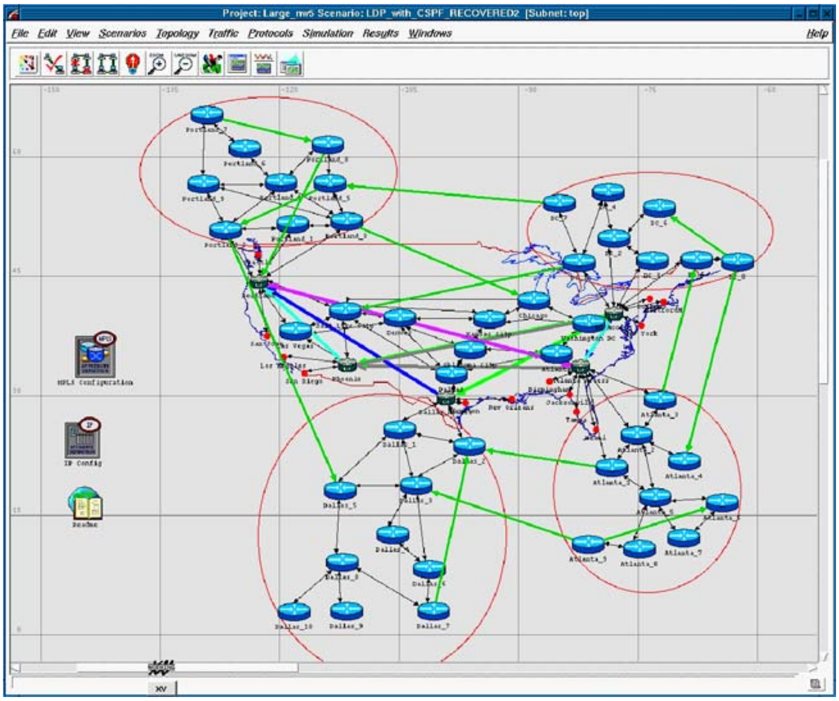
Project: Large_mv5 Scenario: LDP_with_CSPF_RECOVERED2 [Subnet: top]
File Edit View Scenarios Topology Traffic Protocols Simulation Results Windows Help
MPLS Configuration
IP Config
Readme

Fig. 1. The OPNET simulation testbed.

category to which the plurality of its k closest neighbors belong. Using relatively large values of k decreases the chance that the decision will be unduly influenced by a noisy training pattern close to X. But large values of *k* also reduce the acuity of the method. The k-nearest neighbor method can be thought of as estimating the values of the probabilities of the classes given X. Of course the denser are the points around X, and the larger the value of k, the better the estimate [12]. The k-nearest neighbor method is simple but could be very powerful in some

Table I. The node status vector

Failed Nodes	Node Status Vector
at2, at3, at4 at5, at7,	[01111010
da1 da5, da6, da7,	10001110
dc5, dc6, dc8,	00001101
po1, po2, po5, po6, po7	11001110]

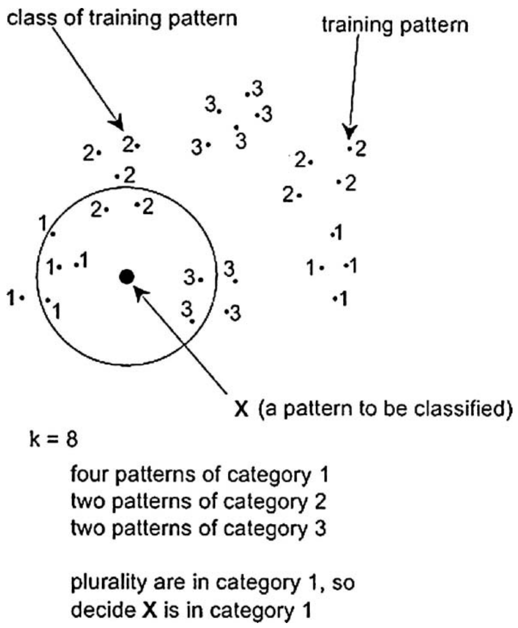
class of training pattern
training pattern
X (a pattern to be classified)
k = 8
four patterns of category 1
two patterns of category 2
two patterns of category 3
plurality are in category 1, so
decide X is in category 1

Fig. 2. K-nearest neighbor method.

applications [13, 14]. There have been several practical applications of k-nearest neighbor method [15, 16].

We need to have a metric to measure the distance between two patterns. The distance metric used in nearest neighbor methods for numerical attributes can be simple Euclidean distance. That is the distance between two patterns $(x_{11}, x_{12}, \ldots, x_{2n})$ and $(x_{21}, x_{22}, \ldots, x_{2n})$ is $\sqrt{\sum(x_{1i} - x_{2i})^2}$. This distance measure is often modified by scaling the features so that the spread of attribute values along each dimension is approximately the same. In that case the distance between the two vectors would be$\sqrt{\sum a_i(x_{1i} - x_{2i})^2}$, where a_i is the scale factor for dimension i. An example of 8-nearest neighbor decision problem [12] is shown in Fig. 2. In the figure, the class of a training pattern is indicated by the number next to it.

In this problem, we use number 1 and 0 to represent the pattern. So we can use another metric-the Hamming distance. The Hamming distance between two vectors $X_1 = (x_{11}, x_{12}, K, x_{1n})$ and $X_2 = (x_{21}, x_{22}, K, x_{2n})$ is defined as the number of different bits between vector X_1 and X_2. Since the bit is either 1 or 0, the Hamming distance metric is equivalent to the Euclidean distance metric.

Table II. The test result of k-nearest neighbor method when $k = 1$

	Test 1	Test 2	Test 3	Test 4
Failed Nodes	at2, at4 at5, at7, da6, da7, dc2, dc6	at2, 3, 4, 5, 6, 7, da1,da5, da6, da7, dc2, dc5, dc6, dc8, po1, 2, 5, 6, 7	at1, at5, at6, at8, da2, da4, dc1, dc4, dc7, po1, po8	at1, at2, at3, at4, at8, da3, da4, da8, dc3, dc5, dc7, po4, po5
Testing Data	[01011010 00000110 01000100 00000000]	[01111110 10001110 01001101 11001110]	[10001101 01010000 10010010 10000001]	[11110001 00110001 00101010 00011000]
Nearest Neighbor	[0101101$\underline{1}$ 00000110 01000100 00000000]	[01111110 1$\underline{1}$001110 01001101 1100111$\underline{1}$]	[1000$\underline{0}$101 0101$\underline{1}$000 1$\underline{1}$010010 10000001]	[111$\underline{0}$0001 00110001 0$\underline{1}$101010 00011000]
Output Message	BGP open alive	BGP keep alive	OSPF update	$\underline{\text{LDP label release}}$
Poison Message	BGP open alive	BGP keep alive	OSPF update	LDP label request

EXPERIMENTAL RESULTS

Test Results from K-Nearest Neighbor Method

After obtaining enough training data, we run new simulations and obtain new node status vectors to test the k-nearest neighbor method. First we test the k-nearest neighbor method when k is 1. And the results are very good. We tested 30 new node status vectors by using the nearest neighbor method, and in 26 cases the poison message type is correctly identified. Parts of the results are given in Table II.

In Table II, the red underlined numbers are the different bits between the input node status vector and its nearest neighbor. From Table II, we can see that test 1, test 2 and test 3 all give correct diagnosis. The nearest neighbor indicates the poison message type correctly. But in test 4, this method gives the wrong diagnosis. In that case, the poison message is LDP label request message, but the nearest neighbor method outputs LDP label release message.

To reduce the influence of noise, we also tested nearest neighbor method when k is larger than one. In particular, we did test when k equals 3, 5 and 7. Some test results for $k = 3$ are reported in Table III. In Table III, the 2nd row from the bottom is the output message types from the 3 nearest neighbors. The last row is the actual poison message. In $k = 3$ case, if the nearest three neighbors are three different message types, we choose the message type with the minimum Hamming distance. If the Hamming distances are also the same, a message type is chosen randomly. (Note: Keep denotes keep alive message).

Table III. Test results for k-nearest neighbor method ($k = 3$)

	Test 1	Test 2	Test 3
Testing Data	[01011000 00010110 01000000 00000000]	[01001010 00000110 01000100 00000000]	[01011100 01000000 01000100 00000000]
Nearest Neighbor 1	[01011000 00010110 $0\underline{0}00000\underline{1}$ 00000000]	[$010\underline{1}1010$ 00000110 01000100 00000000]	[01011100 01000000 $0\underline{0}000\underline{0}00$ 00000000]
Nearest Neighbor 2	[01011000 00010110 $01000\underline{1}00$ 00000000]	[$010\underline{1}10\underline{0}0$ $000\underline{1}0110$ 01000100 00000000]	[$0101110\underline{1}$ $01000\underline{1}00$ $01000\underline{0}00$ 00000000]
Nearest Neighbor 3	[01011000 00010110 $01000\underline{1}00$ 00000000]	[01001010 $\underline{1}0000\underline{00}0$ 01000100 00000000]	[$01011\underline{01}0$ $010000\underline{1}0$ 01000100 00000000]
Output Message	1. Open 2. Open 3. Open	1. Keep 2. Open 3. Keep	1. Update 2. Open 3. Keep
Poison Message	BGP open	BGP keep alive	BGP update

As we can see from Table III, in test 2 even the 2nd nearest neighbor indicates the wrong message type–BGP open message, the output is still correct since the 1st and 3rd nearest neighbors all indicate the poison message–BGP keep alive message. Thus larger k value does reduce the noise influence. In test 3, three nearest neighbors belong to three different message types: BGP update message, open message and keep alive message. The output is BGP update message. Because BGP update message has the minimum Hamming distance with the testing data.

Comparison of Different K Values

We have tested k-nearest neighbor method with k being 1, 3, 5 and 7. And we compared the performance for different k values. The correct percentage for different k value nearest neighbor method is plotted in Fig. 3.

From Fig. 3, we can see that the larger k value, the better testing result. This is due to large k reduce the noise influence. However, large k also means more computation is needed to search the nearest neighbor. We find out that the testing result is already pretty good when $k = 3$, where the correct rate is 92.3%. Increasing k does not improve the testing result very much. The correct rate is

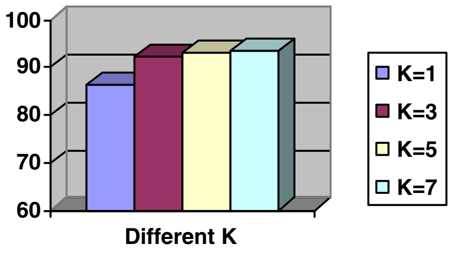

Fig. 3. Performance of different k-nearest neighbor method.

93.6% when k is 7, only 1.3% increase compared to $k = 3$. This suggests that we should choose a proper k value that obtains good result while does not need too much computation. In this test, $k = 3$ seems to be a good choice.

The Second Nearest Neighbor

When $k = 1$, the correct rate of the k-nearest neighbor method is about 86%. And we observe that for most of the wrong diagnosis cases, the second nearest neighbor indicates the poison message correctly. We summarize the result of correct identification rate in Table IV. From Table IV, we can see that in 97% cases, the poison message is either given by the nearest neighbor or the second nearest neighbor.

The Serial Tests

In the previous tests, we use the node status vector at one time step as input to find out the k-nearest neighbors. And then decide the poison message type. Another approach could be to use the node status vectors at several consecutive time steps as input, and combine the results to decide the poison message type – the serial test. An example of the serial test is given in Table V.

The serial test can give more robust results. One way to combine several test results is to use a voting algorithm. All the tests vote for the poison message, and the plurality wins. In case tie happens, the poison message is chose randomly.

Table IV. Correct diagnosis ratio when $k = 1$

The Nearest Neighbor	The 2nd Nearest Neighbor	Totally
86%	11%	97%

Table V. Serial test result

Time	$t=1$	$t=2$	$t=3$
Failed Nodes	at2, at4, at5, at7, da1, da5, dc2, dc6	at3, da6, dc5, dc8, po1, 2, 5, 6, 7, 8	at6, da2
Testing Data	[01011010 10001000 01000100 00000000]	[01111010 10001100 01001101 11001111]	[01111110 11001100 01001101 11001111]
Nearest Neighbor	[010$\underline{0}$1010 10001000 01000100 00000000]	[01111010 1$\underline{1}$001100 01001101 11001111]	[01111110 11001100 01001101 11001111]
Output Message	BGP open	BGP keep alive	BGP keep alive
Voting	BGP keep alive		
Poison Message	BGP keep alive	BGP keep alive	BGP keep alive

In Table V, the poison message is BGP keep alive message. At time $t=1$, the nearest neighbor method outputs the wrong message type–BGP open message. But at time $t=2$ and $t=3$, the nearest neighbor method all diagnosis correctly. And by combining the results at the three time steps, the final result is correct.

Integrating K-Nearest Neighbor Method with Sequential Decision Problem

The nearest neighbor method provides good result to identify the poison message. But we still need some actions to confirm that we find the poison message. One action is to use message filtering–block the possible poison message, and see if the failure propagation stops in a certain time. If it stops, then the identity of the poison message is confirmed. On the other hand, if failures continue, then we will observe some other node failures and use the nearest neighbor method to test the new node status vector again. Since a previously filtered message will not be the poison message, we need to take into account the knowledge about past filtering. I.e., the previously filtered messages should be excluded.

One of the integration test results is given in Table VI. At time $t=1$, the nearest neighbor method determines the poison message is BGP open message. Then BGP open message is blocked in the network. But the failure propagation does not stop in a certain time. This means BGP open message is not the poison message. Then three more node failures–at2, at4 and da4 are observed. The new node status vector is tested by the nearest neighbor method. And the nearest neighbor still belongs to BGP open message. But we already know that BGP open message is not the poison message. So we forget about this nearest neighbor, and

Table VI. Integration test

Time	Failed Nodes	Testing Data	Nearest Neighbor	2nd Nearest Neighbor	Output Message	Poison Message
t = 1	at5 at6, da6, dc2	[00001110 00000100 01000000 00000000]	[00001110 $0\underline{11}00100$ 01000000 00000000]		BGP open	BGP update
		BGP open message is filtered, but the failure is not stopped				
t = 2	at2 at4, da2	[01011110 01000100 01000000 00000000]	[01011110 $01\underline{1}00100$ $01000\underline{1}00$ 00000000]	$[010\underline{0}1110$ $01000\underline{0}00$ $0\underline{0}000000$ 00000000]	BGP update	BGP update

find the 2nd nearest neighbor. And the 2nd nearest neighbor indicates the exact poison message–BGP update message. In Table VI, the underlined number means the different bit between the node status vectors. The italic vector is the node status vector that we should exclude based on previous filtering information.

Probabilistic K-Nearest Neighbor Method

The k-nearest neighbor method determines the class that the target belongs to based on the class of the k-nearest neighbors. So the outputs of the k-nearest neighbor method are always deterministic. I.e., which class the target belongs to. However, sometimes the outputs are not the correct class. In order to increase the robustness of the k-nearest neighbor method, we suggest a "Probabilistic" K-Nearest Neighbor (PKNN) method. In PKNN method, the output is not a deterministic class, but rather is a probability distribution about the possible classes. In the poison message problem, the output is the probability distribution about the poison message (rather than the identity of the poison message in k-nearest neighbor method). So the PKNN method does not completely rule out possible classes other than the one that has the most nearest neighbors.

We also implemented simulations to test the performance of PKNN method. In the simulation, the poison message is randomly selected according to a pre-set probability distribution P_0 from six different message types. The PKNN finds out the ten-nearest neighbors for the node status vector and assign the probability according to the number of different message types in the ten-nearest neighbors. One of the examples is given in Table VII. In Table VII, the first row is the six message types in the simulation. The second row is the numbers of each message type in the ten-nearest neighbors. And the third row is the probabilities–the output of the PKNN method.

Table VII. Output of PKNN method

Ten-Nearest Neighbors	Message Type 1	Message Type 2	Message Type 3	Message Type 4	Message Type 5	Message Type 6
Number of Messages	2	5	0	1	0	2
Probability	0.2	0.5	0.0	0.1	0.0	0.2

Two hundred simulations are run to test PKNN method. We average the probability outputs from these tests and the data is compared with the actually probability distribution of each message type being selected as poison message in the simulations. The results are presented in the Table VIII.

As we can see from Table VIII, the results are very good. The average output of the poison message probability distribution from PKNN method is very close to the pre-set probability distribution in the simulation. This shows that the PKNN method work very well in providing a good probability distribution for the poison message.

CONCLUSION

We have discussed a particular network failure propagation mechanism - poison message failure. In this paper, we applied data mining technique – k-nearest neighbor method to identify the poison message. We have built an OPNET simulation testbed to generate training data as well as testing data. Our tests show that the k-nearest neighbor method is very effective in identifying the identity of the poison message. The correctness rate is greater than 86% even when *k* is 1, and larger *k* value gives better result. However, the tests for different *k* values show that performance does not increase much as *k* increases. The tests suggest 3 is good choice for *k* value in this particular case, which balances the testing performance and the computational complexity. We also performed serial test that combined tests in several consecutive time steps, and the serial test gives better result than standard test. In addition, we integrated the k-nearest neighbor method with the sequential decision problem. Based on the past filtering

Table VIII. Average Probability Distribution from PKNN Method

	Message Type 1	Message Type 2	Message Type 3	Message Type 4	Message Type 5	Message Type 6
Pre-Set Probability	0.16	0.36	0.10	0.15	0.05	0.18
Average Output	0.17	0.33	0.09	0.15	0.06	0.20

information, the test excludes message types filtered before, and thus improves the performance. We proposed a new approach – probabilistic k-nearest neighbor method, which outputs a probability distribution rather than the identity of the target. The probabilistic approach may be favorable in some situations. To sum up, our experiments demonstrate that k-nearest neighbor method is very effective in diagnosing the poison message failure. In our future work, we will study using machine learning and data mining techniques for other network fault and security management problems.

REFERENCES

1. C. Labovitz, C. Malan, and F. Jahanian, Internet routing instability, *Proceedings of the ACM SIGCOMM*, Nice, France, August 1997.
2. AT&T, AT&T announces cause of frame-relay network outage, *News Release*, April, 22, 1998, www.att.com/press/0498/980422.bsb.html.
3. T. Sweeny and C. Moozakis, MCI frame net melts down, *Tech Web*, August 12, 1999.
4. X. Du, M. A. Shayman, and R. Skoog, Preventing network instability caused by control plane poison messages, *Proceedings of the IEEE MILCOM 2002*, Anaheim, CA, October 2002.
5. D. J. Houck, K. S. Meier-Hellstern, and R. A. Skoog, Failure and congestion propagation through signaling controls, *Proceedings of the 14th International Teletraffic Congress*, Elsevier, Amsterdam, pp. 367–376, 1994.
6. N. L. Hung, A. R. Jacob, and S. E. Makris, Alternatives to achieve software diversity in common channel signaling networks, *IEEE JSAC*, Vol. 12, No. 3, pp. 533–538, 1994.
7. X. Du, M. A. Shayman and R. A. Skoog, Using neural networks to identify control and management plane poison messages, *Proceedings of the Eighth IFIP/IEEE International Symposium on Integrated Network Management (IM 2003)*, Colorado Spring, Colorado, March 2003.
8. X. Du, M. A. Shayman and R. A. Skoog, Markov decision based filtering to prevent network instability from control plane poison messages, *Proceedings of the Conference on Information Sciences and Systems (CISS) 2003*, Baltimore, MD, March 2003.
9. X. Du, M. A. Shayman and R. A. Skoog, Distributed fault management to prevent network instability from control and management plane poison messages, *Proceedings of the IEEE Military Communication (MILCOM) 2003*, Boston, Massachusetts, Oct 2003.
10. M. A. Bramer, *Knowledge Discovery and Data Mining*, The Institute of Electrical Engineers, 1999.
11. A. A. Freitas, *Data Mining and Knowledge Discovery with Evolutionary Algorithms*, Springer, 2002.
12. N. J. Nilsson, Introduction to Machine Learning, robotics.stanford.edu/people/nilsson/mlbook.html
13. T. Mitchell, *Machine Learning*, McGraw Hill, 1997.
14. Z. Zhang, *Association Rule Mining*, Springer, 2002.
15. A. W. Moore, Fast, robust adaptive control by learning only forward models, *Advances in Neural Information Processing Systems*, Morgan Kaufmann, 1992.
16. A. W. Moore, D. J. Hill, and M. P. Johnson, An empirical investigation of brute force to choose features, smoothers and function approximators, *Computational Learning Theory and Natural Learning Systems*, Vol. 3, Cambridge MIT Press, 1994.
17. G. Koutepas, F. Stamatelopoulos, and B. Maglaris, Distributed management architecture for cooperative detection and reaction to DDOS attacks, *Journal of Network and Systems Management*, Vol. 12, No. 1, March 2004.

18. Y. Tang and E. S. Al-Shaer, Active integrated fault localization in communication networks, *Proceedings of the Ninth IFIP/IEEE International Symposium on Integrated Network Management (IM 2005)*, Nice, France, May 2005.
19. A. L. dos Santos, E. P. Durate Jr., and G. M. Keeni, Reliable distributed network management by replication, *Journal of Network and Systems Management*, Vol. 12, No. 2, June 2004.
20. D. Raz and Y. Shavitt, Toward Efficient distributed network management, *Journal of Network and Systems Management*, September 2001.
21. K. Yoshihara, M. Isomura, and H. Horiuchi, Dynamic load balancing for distributed network management, *Proceedings of the Eighth IFIP/IEEE International Symposium on Integrated Network Management (IM 2003)*, Colorado Spring, Colorado, March 2003.

Xiaojiang (James) Du is an assistant professor in Department of Computer Science, North Dakota State University. Dr. Du received his B.E. degree from Tsinghua University, Beijing, China in 1996, and his M.S. and Ph.D. degrees from University of Maryland, College Park in 2002 and 2003, respectively, all in Electrical Engineering. His research interests are wireless sensor networks, mobile ad hoc networks, network security and network management. Dr. Du is an associated editor of Wiley Journal of Wireless Communication and Mobile Computing. He is the program chair of Computer and Network Security Symposium of IEEE International Wireless Communication and Mobile Computing Conference (IWCMC) 2006. He is (was) a TPC member for many major IEEE conferences such as INFOCOM, ICC, GLOBECOM, IM, and NOMS.