International Journal of Hybrid Information Technology
Vol.8, No.1 (2015), pp.203-216
http://dx.doi.org/10.14257/ijhit.2015.8.1.18

Monitoring Neighborhood Self-organization and Message Dissemination for Monitoring Large-scale Distributed Systems

ShuYu Chen[1], GuiPing Wang[2*], Jun Liu[2] and MingWei Lin[2]

[1]College of Software Engineering, Chongqing University, Chongqing, China
[2]College of Computer Science, Chongqing University, Chongqing, China
netmobilab@cqu.edu.cn, w_guiping@163.com, liujun_314@cqu.edu.cn, linmwcs@163.com

Abstract

In order to successfully monitor a large-scale distributed system, it is an important issue that the monitoring function fully covers all the entities in the system. To this end, a key challenge is to efficiently transmit state information of the entities in the system. This paper solves this challenge from two aspects. First, in virtue of the idea of self-organizing networks, this paper proposes a neighborhood organization algorithm, which self-organizes the nodes into several monitoring neighborhoods based on the t distance between nodes. The second aspect focuses on message transmission. There are three common message transmission methods in network, i.e., flooding, multicast and unicast. Flooding may cause high network overhead, while unicast may pose high system delay. Based on the idea of Gossip protocol, this paper proposes a directional message dissemination algorithm (D-Gossip), which is a kind of probabilistic multicast. D-Gossip reduces message dissemination uncertainty of traditional Gossip protocols. It effectively improves the efficiency and coverage of message dissemination, while reducing redundant information in the system due to Gossip protocol. The experimental results show that the neighborhood organization algorithm and the D-Gossip can effectively solve the above challenge.

Keywords: *Large-scale Distributed Systems, Self-Organizing Networks (SON), Monitoring, Message Dissemination, Gossip Protocol, D-Gossip*

1. Introduction

In recent years, a variety of large-scale distributed systems, such as P2P, grid, WSN, ad hoc [1], and cloud computing system [2], have been developing rapidly. For these systems, fault detection is valuable for system management, replication, load balancing, and other distributed services [3]. When designing a distributed fault detection system, an important foundation is monitoring all the entities in the system and achieving that fault detection function fully covers the whole system. Along with the rapid development of network technology, distributed systems gradually become more and more open. Any node, as long as it follows the standard protocols, it can dynamically join in or disjoin from the system. Such systems often involve a large number of distributed computing nodes and span different geographic regions with unstable communication delay and loosely management. It is very difficult to achieve that fault detection function fully covers such systems.

Aiming at solving this problem, this paper makes three main contributions listed as follows: 1) in virtue of the idea of self-organizing networks (SON), this paper proposes a neighborhood organization algorithm, which self-organizes nodes into several monitoring neighborhoods based on *t* distance between nodes; 2) to overcome the defects of flooding, multicast and unicast, this paper proposes a message dissemination algorithm (D-Gossip) based on gossip protocol; 3) this paper theoretically analyzes coverage efficiency and network overhead of D-Gossip, and conducts experiments to verify the analyses.

The remainder of this paper is organized as follows. Section 2 summarizes related work. Section 3 introduces SON and *t* distance, and then proposes the neighborhood organization algorithm. Section 4 summarizes two basic modes of transmitting detection messages. Section 5 proposes the D-Gossip algorithm. Section 6 presents experiments and analyses. Finally, section 7 gives conclusions and looks into future work.

2. Related Work

In large-scale distributed systems, due to the large number of nodes and wide geographic distribution, traditional centralized management methods can not meet the high dynamic network structure. In recent years, a variety of network forms with obvious self-organizing characteristics, such as P2P, Grid, Wireless Sensor Network (WSN), ad hoc [1], and cloud [2], have been developing and evolving. Self-organizing significantly improves organization and management of nodes in large-scale distributed systems.

In a large-scale distributed system, a monitoring function which covers all the nodes may be constructed through self-organizing monitoring domains. For fault detection, the construction of a monitoring system covering all the nodes is a fundamental step. Another important issue is how to disseminate the state information of all entities in the network. The traditional dissemination methods include *flooding*, *multicast* and *unicast*. But these methods cannot make a well trade-off between network overhead and real-time messages dissemination. As a kind of probabilistic multicast [4], random dissemination based on gossip protocol provides a better solution to this problem. With certain redundant messages, this random dissemination strikes a better balance between network overhead and real-time messages dissemination.

Gossip-based communication mechanisms [5]-[7] spread information in network in a manner similar to the spread of a virus in a biological community. Therefore, gossip mechanisms are also known as epidemic algorithms [8][10].

The theory of epidemic algorithm can be traced back to the 1920s [11]. A large number of researchers began to observe and study infection modes of epidemic diseases in biological communities. The main purpose is to guide people to prevent epidemic diseases. The first research work which introduces epidemic algorithm to the field of information science is done by Demers [12], who employs epidemic algorithm to understand and maintain database consistency in backup. Since then, epidemic algorithms get broad attention and rapid development.

Currently, gossip mechanisms have been widely used in message propagation [13], data replication [8], data aggregation [9], resource discovery and monitoring [14], performance monitoring [15], fault detection [3], [16][18], *etc*.

Ganesh *et al.*, [19] prove the following result: if there are N nodes in a large-scale distributed system, and each node gossips to $\log(N+c)$ other nodes on average, then the probability that everyone gets the notification converges to $\exp(-e^{-c})$. This means that if the number of messages transmitted by per node exceeds $\log(N)$, then the probability that each node in the system can receive a source message is close to 1. Obviously, the significance of the above result is that it theoretically proves the feasibility and reliability of message dissemination based on Gossip.

Renesse [3] proposes a Gossip-style failure detection protocol, which utilizes high reliability of gossip broadcast in network message dissemination and avoids network congestion due to flooding broadcast. A disadvantage of this Gossip-style protocol is that it also brings a certain amount of redundant information, which degrades the system's scalability.

Although the scale of distributed systems becomes larger in recent years, gossip mechanisms are still widely applied to large-scale cloud systems [20], and exascale systems [21].

This paper combines the idea of SON and gossip mechanisms to construct a monitoring system that fully covers the nodes in a large-scale distributed system.

3. Distributed Monitoring based on Self-organizing Neighborhoods

This section first introduces SON and *t* distance, and then proposes the neighborhood organization algorithm.

3.1. Self-organizing Network (SON)

Based on the existing physical network, SON is a virtual network topology composed of nodes and logical links. Following the logic of applications or services, SON employs the basic transmission capacity provided by the underlying network, and reorganizes the network into a virtual network topology to provide functions that the underlying network can not provides. SON ignores the details of the underlying network topology. Its structure is closely related to specific application purposes. As shown in Figure 1, the logical implementation of SON is often quite different with the underlying network. For two nodes, there is no direct relationship between their logic link in SON and their physical link in the underlying network. In the actual network, several nodes may be in different subnets, but they are organized into a same domain in SON according to the upper application requirements.

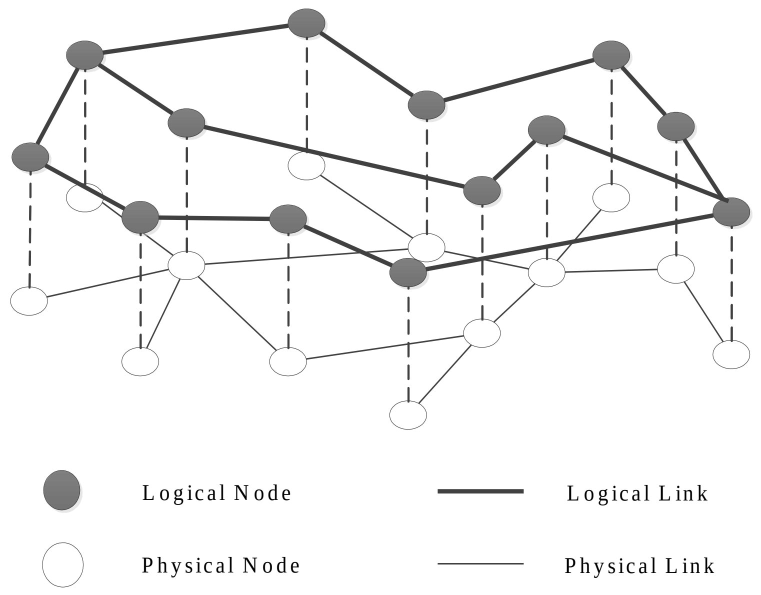


Figure 1. SON and the Underlying Physical Network

3.2. Self-organizing of Monitoring Neighborhoods based on *t* Distance

For SON, an important issue is to partition all the nodes into several domains according their logic relationships. The logical partition needs to be an optimal solution, rather than arbitrarily separating all the nodes into several parts. The most direct solution idea is to partition all the nodes according to their adjacent distance relationship. Since it is a neighbor relationship, there must be a direct relationship with near or far distance. Therefore, an abstract distance can be introduced to characterize the distance relationship between nodes. In general, the abstract distance between nodes in the network can be based on the degree of correlation, network packets forwarding hops, or network delay between nodes, which leads to three different definitions of distance, namely, *s* distance, *h* distance and *t* distance. In practice, the network distance commonly measures as *t distance*, which is defined as follows.

Definition 1(*t* distance): Assuming nodes N_i and N_j are in a fully connected network, the time delay of node N_i sending a message to and back from node N_j is *t*, then $d_i^j = t$ is the *t* distance between nodes N_i and N_j.

Obviously, the smaller the *t* distance, the closer between the nodes N_i and N_j.

According to definition 1, the *t* distance includes the time delay of node N_i sending detection message to node N_j, the time delay of node N_j dealing with the message, and the time delay of node N_j sending back acknowledgement message to node N_i, as shown in Figure 2.

As Figure 2 shows, $t = t_{Req} + t_{Deal} + t_{Ack}$. But in practice, the time delay t_{Deal} is often negligible, Therefore, $t = t_{Req} + t_{Ack}$.

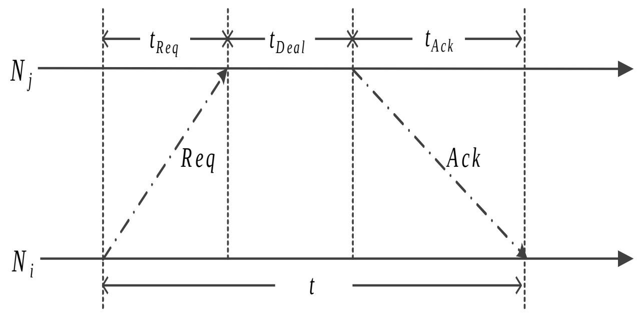


Figure 2. Composition of *t* Distance

Before the fault detection system forming monitoring neighborhoods, the nodes in the system are mutually independent. They need to be self-organized into several monitor neighborhoods according to the algorithm below.

Algorithm 1 (neighborhood organization algorithm): Self-organization of monitoring neighborhoods based on *t* distance

Assuming that the size of self-organized neighborhood is $K = k + l$, where *k* is the initial threshold value of each monitoring neighborhood, *l* is a flexible size for the consideration of redundant space to accommodate newly added nodes. The algorithm works as follows:

Step 1: It broadcasts monitoring neighborhood construction request messages to all nodes in the system through an initially or subsequently selected proxy node.

Step 2: Each surviving node having not joined in any monitoring neighborhood responds to the request message. The proxy node receives the response messages and sorts the responding nodes in accordance with their response time (in ascending order).

Step 3: It judges whether the number of received acknowledgement messages is greater than *k*; and if so, it jumps to step 4; otherwise, it jumps to step 6.

Step 4: According to reach time of acknowledgement messages, it selects top *k* nodes to form a monitoring neighborhood of the proxy node. The upper bound of the neighborhood's *t* distance is 2 times of the *t* distance of the latest reply message. Meanwhile, the proxy node sends a neighborhood construction confirmation message with upper bound information of the neighborhood's *t* distance to all the *k* nodes. And then it completes construction of a monitoring neighborhood.

Step 5: The proxy node sends a node list that includes the remaining nodes not joining in the constructed neighborhoods to a selected node with relatively long response delay. The principle of selecting such a node is that the response delay of this node is not greater than 2 times the upper bound of the neighborhood's *t* distance. It requests the selected node act as the next monitoring proxy node. For the selected node, it jumps to step 1. For the current proxy node, it jumps to step 6.

Step 6: The proxy node sends a neighborhood construction confirmation message to all responding nodes. The upper bound of the neighborhood's *t* distance is 2 times of the *t* distance of the latest reply message. It completes a monitoring neighborhood construction. The algorithm ends.

After all the neighborhoods covering all the nodes in the system have been constructed, the nodes in a single neighborhood execute all-to-all monitoring. These nodes mutually distribute state information of monitored objects in a gossip manner. Meanwhile, the message dissemination between neighborhoods is implemented through the proxy nodes in each neighborhood. Combing within- and between-neighborhood message dissemination, it achieves system-wide message coverage.

4. Transmission Strategies of Detection Messages

In the traditional distributed fault detection methods, there are in general two basic modes to transmit detection messages, *i.e.*, the ping mode and the heartbeat mode, which are described below.

4.1. The Ping Mode

The ping model is also called pull mode or active mode. In this mode, the detector actively sends liveness request periodically to monitored components, as shown in Figure 3. In this case, monitored components do not actively send their state information to the detector, but reply to the detector's request. For non-all-to-all fault detection, this transmission mode can effectively reduce the network overhead. However, in the polling cycle, the detector is unable to obtain any information about monitored components. Meanwhile, for all-to-all fault detection, it means more network overhead, since this strategy generates not only report messages but also query messages. Moreover, monitored components may issue the same message for different inquiry messages. In this case, it will consume more network bandwidth to transfer acknowledge message.

Generally speaking, the ping mode is suitable for a hierarchical management model with a clear framework, and there are dedicated detectors monitoring the monitored components in a management domain.

4.2. The Heartbeat Mode

The heartbeat mode is also known as push mode or passive mode. In this mode, each monitored component periodically sends messages (heartbeat messages) to the failure detector which is monitoring the component [17], as shown in Figure 3. Without polling operation, this method can effectively reduce the workload of the detector itself.

Both transmission modes have advantages and disadvantages. However, the ping mode is not suitable for all-to-all fault detection, since the ping mode will pose certain threat to self-organization. Meanwhile, in the ping mode, detectors need a certain global knowledge for the monitored system, which increases the difficulty to design the detectors.

In order to achieve scalability and meet the high dynamism of nodes, this paper employs the heartbeat mode to effectively transmit state information.

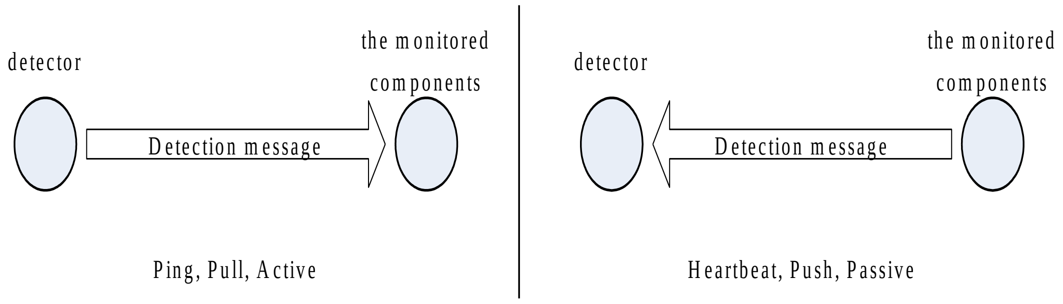


Figure 3. Two Interaction Modes between the Detectors and the Monitored Components

5. Disseminating Detection Messages based on Gossip Protocol

This section proposes an algorithm of disseminating detection messages based on gossip protocol, called D-Gossip in this paper.

5.1. The Basic Principle of Gossip Protocol

The process of information dissemination in a distributed system is similar to the spread of infectious diseases. When an entity receives some information, it randomly disseminates the information following specific rules to other individuals directly connected with it, and so forth. This process disseminates information to a network-wide system without the administrative control of a centralized center. Gossip mechanisms have many advantages, such as low deployment difficulty, high robustness, and high fault-tolerance. In the latest researches, gossip mechanisms act as data distribution means in large-scale networks and attract widespread attention. The research experiments in literature *e.g.*, [19] show that even in high dynamic networks with high node failure rate, high packet loss rate, and limited communication links, gossip mechanisms can still achieve high reliability through appropriately setting dissemination parameters. This means that, with reasonably redundant information, gossip mechanisms ensure all entities receive detection messages in a high probability.

Although gossip mechanisms trade the redundant messages for compensating reliability degrading due to node failure and package losing, the amount of redundant messages is only log(N) for a single node [19], where N is the total number of nodes in a gossip dissemination network. Therefore, gossip mechanisms have good scalability.

5.2. Dissemination Method based on Gossip Protocol

This subsection discusses message dissemination mechanism in a single monitoring neighborhood. According to the discussion about detection message transmission in Section 4, this paper employs gossip-based message dissemination mechanism to transmit heartbeat messages.

Each monitored component, p, is assigned a fault detector, d. All the fault detectors form a set of detectors. Each fault detector d maintains a local view, *LocalView*, which saves the relevant information about other detectors that can communicate directly with *d*. In this paper, the directly interconnected entities are called *neighbors*. In addition, "node" and "detector" are interchangeable in this paper.

A detector divides its neighbors into three states, which corresponding to three different sets: healthy detector set, $D_{\boldsymbol{heal}}$; suspect detector set, $D_{\boldsymbol{susp}}$; and failed detector set, $D_{\boldsymbol{fail}}$.

After a time interval T_{inter}, each detector in the system will actively send its state information in random gossip manner to a specified number of neighbors in the sets of $D_{\boldsymbol{heal}}$ and $D_{\boldsymbol{susp}}$. When sending state information, the detector *d* increases its own heartbeat counter (*HBcounter*) by 1. Meanwhile, it adds a timestamp (*Timestamp*) into the message. Each individual message has a globally unique ID number. The ID value can be easily achieved through hashed value of *d*'s IP address and port number, plus the heartbeat count *HBcounter*. Each node also maintains a list of messages, indexed by the messages' IDs.

When a node receiving the detection message from another node, it first analyzes the message, and then executes corresponding processing according to whether the message has been received before or not. The description of directional gossip-based message dissemination (referred to as D-Gossip) will be given below.

Figure 4 shows the flow chart of D-Gossip message dissemination algorithm.

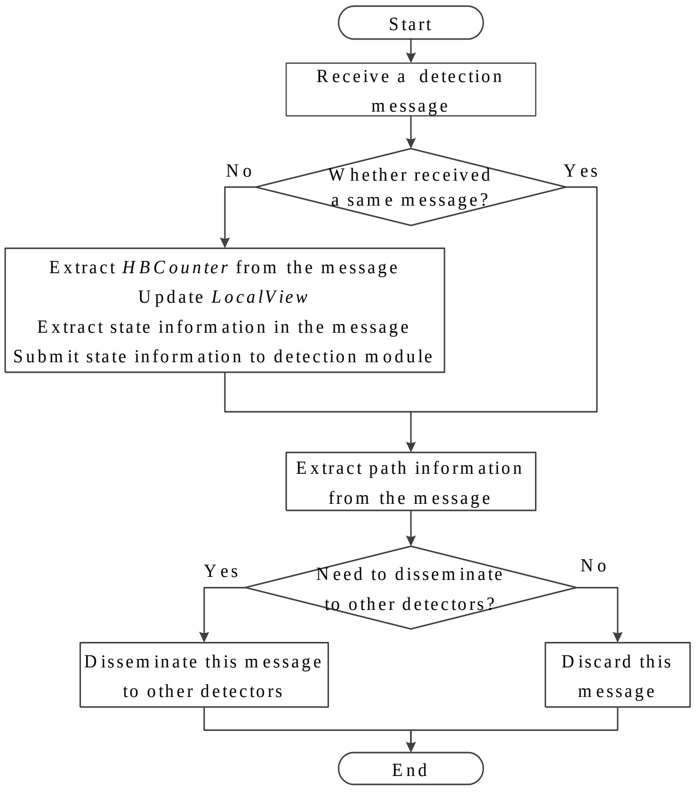

Start
Receive a detection message
No
Whether received a same message?
Yes
Extract HBCounter from the message
Update LocalView
Extract state information in the message
Submit state information to detection module
Extract path information from the message
Yes
Need to disseminate to other detectors?
No
Disseminate this message to other detectors
Discard this message
End

Figure 4. The Flow Chart of D-Gossip

Algorithm 2 (D-Gossip): Message dissemination based on gossip protocol

Assumption: this algorithm concerns a node (denoted as the current node) in a monitoring neighborhood; in its *LocalView*, both the healthy detector set ($D_{\boldsymbol{heal}}$) and suspect detector set ($D_{\boldsymbol{susp}}$) are not empty; every time, the number of objects (*i.e.*, the *fanout*) to which the node disseminate the message is denoted as *f*.

Step 1: It analyzes the received detection message, and judge whether the received message is same as one received before according the message ID. If yes, it jumps to step 5; otherwise, it executes step 2.

Step 2: If the node which transmits this messages to current node exists in the set of D_{heal} or D_{susp}, it updates the node information according *HBCounter* value of the message, and moves the node to D_{heal}.

Step 3: It extracts state information from the detection message, and forwards state information to the fault detection module.

Step 4: It extracts *TTL* (Time To Live) value of the message; if the *TTL* value is greater than 0, it continues to execute step 5; otherwise, it jumps to step 7.

Step 5: It takes the healthy and suspect detectors in the *LocalView* as the set of dissemination objects; then it extracts path information from the message and deletes duplicate nodes in the dissemination set compared with the nodes in the path; if the dissemination set is not empty, it executes step 6; otherwise, it jumps to step 7.

Step 6: It adds the current node to the message path; if the dissemination set is smaller than *f*, it sends the message to all the nodes in the dissemination set; if the dissemination set is larger than *f*, it randomly selects *f* nodes in the dissemination set to transmit the message. The algorithm ends.

Step 7: It discards the message. The algorithm ends.

5.3. Coverage Analysis of D-Gossip Message Dissemination

The proposed D-Gossip algorithm transmits the state information of a node to the entire system. This subsection analyzes whether the message dissemination can cover the entire network. First, the execution of D-Gossip in a single neighborhood is analyzed.

When a detector receives a detection message for the first time, it is said to *be infected*. A round, denoted by *r*, is referred that the detection message is transmitted one time.

Assume that in the *r*-th round, there are λ_r infected nodes. The probability of uninfected nodes to be infected is considered. It is clear that when $r = 0$, $\lambda_0 = 1$.

First, the random dissemination process without dissemination path is considered. When a node receives a detection message, if the *TTL* value of this message is greater than 0, D-Gossip then randomly selects *f* nodes from its *LocalView* as the set of dissemination objects in the next round. In order to facilitate the analysis, this paper assumes that the *LocalView* of a single node contains all the nodes in its own neighborhood, and the number of nodes in its neighborhood is *n*.

For a single node, the simplest method is to randomly select *f* dissemination nodes from its *LocalView*. Therefore, the probability of any neighbor node in the *LocalView* to be infected is $f/(n\text{-}1)$. While the probability of any neighbor node in the *LocalView* not receiving detection message is $1\text{-}f/(n\text{-}1)$. In the *r*-th round, there are λ_r infected nodes. Therefore, the probability that an uninfected node in the neighborhood will still not be infected in the current round should be $\left(1 - f\big/(n-1)\right)^{\lambda_r}$. It can be conclude that, with λ_r nodes having been infected in the former *r* rounds, the probability of a particular uninfected node to be infected is computed as follows:

$$\rho(\lambda_r) = 1 - \left(1 - \frac{f}{n-1}\right)^{\lambda_r} \tag{1}$$

D-Gossip embeds dissemination path into the detection message. Namely, in each dissemination round, it adds the information of current node to the path. Therefore, in the next dissemination round, the nodes having passed by will not be considered. Thus, $\rho(\lambda_r)$ can get improved. The improved probability is denoted by $\rho'(\lambda_r)$ and computed as follows:

$$\rho'(\lambda_r) = 1 - \left(1 - \frac{f}{n - h_i - 1}\right)^{\lambda_r} \tag{2}$$

In the above formula, h_i is called dissemination avoiding factor, which is the number of deleted nodes in the current dissemination round. The introduction of h_i can effectively reduce the number of redundant messages in D-Gossip.

Based on above analysis, the iterative relationship of system infection evolution can be drawn as:

$$\lambda_{r+1} = \lambda_r + (n - \lambda_r)\rho'(\lambda_r) \tag{3}$$

In addition, in the (r+1)-th round, the number of issued messages in the system is $f \cdot \lambda_r$. Then the probability that all nodes are infected in the (r+1)-th round is $\left(\frac{\lambda_r}{n}\right)^{\frac{n}{f}}$.

Let α be the threshold of detection coverage, when:

$$\alpha < \left(\frac{\lambda_r}{n}\right)^{\frac{n}{f}} \tag{4}$$

, the number of rounds needed to satisfy the coverage α of the detection system can be calculated through formulae (1), (2), and (3).

5.4. Network Overhead Analysis of D-Gossip Message Dissemination

Excessive network consumption will pose the network overloaded and congested. This subsection analyzes network overhead of the proposed D-Gossip algorithm.

In order to facilitate the analysis, this paper approximates the h_i value of each node as the mathematical expectation value of all h_is, denoted as h. The approximation does not bring any change to the nature of D-Gossip, but simplifies the discussion below. Therefore, the equation (2) becomes:

$$\rho(\lambda_r) = 1 - \left(1 - \frac{f}{n - h - 1}\right)^{\lambda_r} \tag{5}$$

The following equation is got when substituting (5) into (3).

$$\lambda_{r+1} = n - (n - \lambda_r)\left(1 - \frac{f}{n - h - 1}\right)^{\lambda_r} \tag{6}$$

By equation (6), the following equation can be obtained:

$$\frac{n - \lambda_{r+1}}{n - \lambda_r} = \left(1 - \frac{f}{n - h - 1}\right)^{\lambda_r} \tag{7}$$

Since when n is sufficiently large, the value of $\frac{f}{n-h-1}$ is far less than 1, equation (7) can be approximated as:

$$\frac{n-\lambda_{r+1}}{n-\lambda_r} = e^{\frac{-f\cdot\lambda_r}{n-h-1}} \tag{8}$$

Further, the following equation can be obtained:

$$\lambda_r = \left(\frac{n-h-1}{f}\right)\ln\left(\frac{n-\lambda_r}{n-\lambda_{r+1}}\right) \tag{9}$$

From the dissemination manner of D-Gossip algorithm, it is known that in the (r+1)-th round, the number of issued messages in the system is $f{\cdot}\lambda_r$. Therefore, during the former (r+1) rounds, the total number of messages, θ, can be computed as follows:

$$\begin{aligned}\theta &= \sum_r f\cdot\lambda_r \\ &= \sum f\left(\frac{n-h-1}{f}\right)\ln\left(\frac{n-\lambda_r}{n-\lambda_{r+1}}\right)\end{aligned} \tag{10}$$

Therefore, the total number of messages is:

$$\theta = (n-h-1)\ln(n-1)\,. \tag{11}$$

The above formula shows that the total number of messages has a direct relationship with the size of neighborhood, n. From equation (11), the following conclusion is also obtained.

For the neighborhood-based monitoring method, its network overhead is certainly better than one without neighboring.

The underlying reason is that, since the neighborhood is divided from the monitored system, the number of nodes in a single neighborhood is certainly less than the total number of nodes in the monitored system. Therefore, the above conclusion is obtained according to equation (11).

6. Experiments and Analysis

This paper implements a network monitoring tool based on socket network programming API under Linux operating systems. It simulates D-Gossip message dissemination in the wide area network environment. Message dissemination within a neighborhood employs UDP protocol, while TCP is applied between neighborhoods.

The following experiments mainly focus on statistical analysis of coverage of message dissemination and the amount of messages. The dissemination interval is set to 400ms.

6.1. Coverage Experiment of D-Gossip Message Dissemination

16 nodes are selected to compose a testbed. The operating system in each node is Linux (kernel version is 2.6.32). The network bandwidth is 100M. Multiple processes are running on each node, and each process initiates up to 16 fault detection threads to imitate random dissemination. Each fault detection thread imitates a fault detector. The total number of fault

detectors is denoted as N. Each thread configures a Timer object to control the time interval of send detection messages. The experiments observe the coverage of infected nodes in the system achieved by D-Gossip along with round r increases. The parameter f is set as 2. Under different system scale, Table 1 shows the comparison of the coverage along with round r increases.

Table 1. Comparison of Infection Coverage by Different Rounds (f = 2)

N \ r	3	5	7	9	11	13	15	17
64	0.16	0.55	0.93	0.97	1	-	-	-
128	0.09	0.36	0.59	0.86	0.97	1	-	-
1024	0.01	0.05	0.18	0.52	0.68	0.85	0.94	0.96

The experiment results in Table 1 show that, along with the expansion of the system scale, the dissemination rounds needed to cover the whole system also increase. Taking the threshold α as 0.95, when $N = 64$, and $r \geq 9$, the coverage>α; when $N = 128$, and $r \geq 11$, the coverage>α; while when $N = 1024$, until after the 17th round, the coverage meets threshold requirements. This fully shows that the larger the size of the system to monitor, the more significant the time needing to cover the whole system, which also confirms the correctness of the basic idea in D-Gossip.

According to D-Gossip, each node excludes the infected nodes in the received message, and then randomly selects f nodes in the remaining neighbor nodes to disseminate the message. If there is no node to disseminate, or the *TTL* value of the message is 0, then the dissemination process automatically ends. The experiment is independently run 10 times. The final results of these 10 experiments are calculated as average transmission times and average coverage of message dissemination, as shown in Table 2.

As can be seen in Table 2, if the fanout, f, is larger, the average coverage also increases. But a larger f value will cause high local network traffic during initial dissemination process, which might affect network congestion or cause packet loss in the actual network. This in return will restrict the average dissemination time and average coverage. Therefore, in order to make a trade off between network overhead and coverage convergence efficiency, a fanout value should be appropriately chosen according to current network state and coverage convergence requirement.

Table 2. The Experiments Result of Message Dissemination in a Neighborhood

Number of detectors within a neighborhood	*TTL*	f	The average number of messages within a neighborhood	Average Coverage
32	10	2	103	99.4%
64	10	2	266	98.5%
		3	284	99.4%
128	10	2	614	95.8%
		3	658	98.6%
256	10	2	1418	93.4%
		3	1488	98.3%
		4	1642	99.2%

6.2. Network Overhead Experiment of D-Gossip Message Dissemination

24 nodes are selected to compose a testbed, where 12 nodes locate in a laboratory in University of Electronic Science and Technology, the other 12 nodes locate in a laboratory in Chongqing University. Each node runs 8 fault detection threads. Each fault detection thread imitates a fault detector. Therefore, the total number of fault detectors is 192. The detection coverage threshold α is set as 0.95.

By executing the neighborhood construction algorithm in Section 3.2, and setting the upper threshold of neighborhood size as 80, three neighborhoods are obtained. When the coverage reaches the threshold, D-Gossip ends, and the total number of detection messages (*i.e.*, network overhead) can be calculated. The experiment compares D-Gossip with the purely random dissemination method Renesse, [3] and flooding. The experimental results are shown in Table 3, where each number denotes the total number of detection messages.

In Table 3, R_i represents a dissemination round, C_{avg} represents the average number of detection messages. As can be seen from Table 3, the D-Gossip algorithm reduces 31% network overhead compared with the purely random dissemination (Renesse), which is congruent with the analysis in Section 5.4. Although the flooding method achieves fewer detection messages, it is prone to make local network traffic surge, which results in a greater probability of data loss occurring. Therefore, in each round of the experiment, the flooding method can not achieve complete coverage of the nodes. Especially in crowded public network during the day time, packet loss is even more obvious, which also causes a greater latency messaging and reducing its availability.

Table 3. Network Overhead Comparison of 3 Methods

	R_1	R_2	R_3	R_4	R_5	R_6	C_{avg}
D-Gossip	819	856	923	791	813	756	826
Renesse	1228	1356	1303	1084	1101	1205	1213
Flooding	182	168	171	183	179	182	178

7. Conclusion and Future Work

Based on self-organizing ideas, this paper proposes a neighborhood organization method based the *t* distance between system entities. This method can effectively monitor all nodes in the system, and reduces monitoring delay, thus improving monitoring efficiency. In addition, the paper also proposes an algorithm (*i.e.*, D-Gossip) of message dissemination based on gossip protocol. D-Gossip reduces message dissemination uncertainty of traditional Gossip protocols, and effectively improves the efficiency and coverage of message dissemination. Meanwhile, it reduces the amount of redundant detection messages.

In future we will focus on optimizing the distance between nodes. The introduction of *s* distance will aggregate detection services based on the correlation between services, which will make fault detection more accurate and efficient, and provide more support for the top decision-making applications.

Acknowledgments

We are grateful to the editors and anonymous reviewers for their valuable comments on this paper.

The work of this paper is supported by Research Fund for the Doctoral Program of Higher Education of China (Grant No. 20110191110038) and National Natural Science Foundation of China (Grant No. 61272399).

References

[1] A. Gaba, S. Voulgaris, K. Iwanicki and M. van Steen, "Revisiting Gossip-Based Ad-Hoc Routing", In: Proceedings of 21st International Conference on Computer Communications and Networks (ICCCN), (**2012**), pp. 1-6.

[2] M. Armbrust, A. Fox, and R. Griffith, "A view of cloud computing", Communications of the ACM, vol. 53, no. 4, **(2010)**, pp. 50-58.

[3] R. Renesse R, Y. Minsky and M. Hayden, "A gossip-style failure detection service", In: Proceedings of IFIP International Conference on Distributed Systems Platform and Open Distributed Processing, **(2009)**, pp. 55-70.

[4] P. Th. Eugster and R. Guerraoui, "Probabilistic multicast", In: Proceedings of 32nd International Conferences on Dependable Systems and Networks (DSN), (**2002**), pp. 313-324.

[5] S. Boyd, A. Ghosh, B. Prabhakar and D. Shah, "Randomized gossip algorithms", IEEE Transactions on Information Theory, vol. 52, no. 6, (**2006**), pp. 2508-2530.

[6] D. Kempe and J. Kleinberg, "Protocols and impossibility results for gossip-based communication mechanisms", In: Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science, (**2002**), pp. 471-480.

[7] K. Birman, "The Promise, and Limitations, of Gossip Protocols", ACM SIGOPS Operating Systems Review, vol. 41, no. 5, (**2007**), pp. 8-13.

[8] J. Holliday and R. Steinke, "Epidemic algorithms for replicated databases", IEEE Transactions on Knowledge and Data Engineering, vol. 15, no. 5, (**2003**), pp. 1218-1238.

[9] M. Jelasity and A. Montresor, "Epidemic-style proactive aggregation in large overlay networks", In: Proceedings of the 24th International Conference on Distributed Computing Systems, (**2004**), pp. 102-109.

[10] B. Ghit, F. Pop and V. Cristea, "Epidemic-Style Global Load Monitoring in Large-Scale Overlay Networks", In: Proceedings of International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), (**2010**), pp. 393-398.

[11] W. O. Kermack and A. G. McKendrick, "A contribution to the mathematical theory of epidemics", In: Proceedings of the Royal Society of London, Series A, Containing Papers of a Mathematical and Pysical Caracter, vol. 115, no. 772, (**1927**), pp. 699-721.

[12] A. Demers, M. Gealy and D. Greene, "Epidemic algorithms for replicated database maintenance", In: Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing (PODC), (**1987**), pp. 1-12.

[13] I. Antonios, F. Zhang and L. Lipsky, "A Performance Model of Gossip-Based Update Propagation", In: Proceedings of 9th IEEE International Symposium on Network Computing and Applications (NCA), (**2010**), pp. 125-131.

[14] R. Renesse, K. Birman and W. Vogels, "Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management, and Data Mining", ACM Transactions on Computer Systems, vol. 21, no. 2, (**2003**), pp. 164-206.

[15] J. W. Zhu, P. G. Bridges, A. B. Maccabe, "Lightweight Online Performance Monitoring and Tuning with Embedded Gossip", IEEE Transactions on Parallel and Distributed Systems, vol. 20, no. 7, (**2009**), pp. 1038-1049.

[16] S. Ranganathan, A. D. George, R. W. Todd and M. C. Chidester, "Gossip-Style Failure Detection and Distributed Consensus for Scalable Heterogeneous Clusters", Journal of Cluster Computing, vol. 4, no. 3, (**2001**), pp. 197-209.

[17] N. Hayashibara, A. Cherif and T. Katayama, "Failure Detectors for Large-Scale Distributed Systems", In: Proceedings of 21st IEEE Symposium on Reliable Distributed Systems, (**2002**), pp. 404-409.

[18] Y. Horita, K. Taura and T. Chikayama, "A scalable and efficient self-organizing failure detector for grid applications", In: Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing, (**2005**), pp. 202-210.

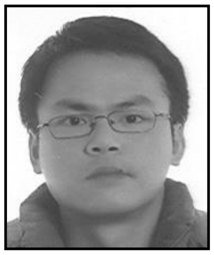
[19] A. J. Ganesh, A. M. Kermarrec and L. Massoulié, "Scamp: peer-to-peer lightweight membership service for large-scale group communication", In: Proceedings of the 3rd International COST264 Workshop on Networked Group Communication, (**2001**), pp. 44-55.

[20] J. S. Ward and A. Barker, "Varanus: In Situ Monitoring for Large Scale Cloud Systems", In: Proceedings of IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom), (**2013**), pp. 341-344.

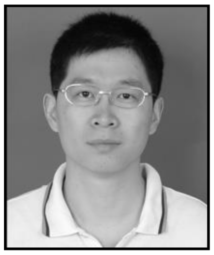
[21] P. Soltero, P. Bridges, D. Arnold and M. Lang, "A gossip-based approach to exascale system services", In: Proceedings of the 3rd International Workshop on Runtime and Operating Systems for Supercomputers (ROSS), (**2013**), Article No. 3.

Authors

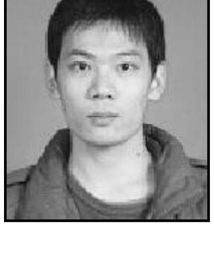
ShuYu Chen, he received his Ph.D. degree in Chongqing University, P. R. China, at 2001. Currently, he is a professor of College of Software Engineering at Chongqing University. His research interests include distributed systems, cloud computing, embedded Linux system. He has published over 120 journal and conference papers in related research areas during recent years.



GuiPing Wang, he received his B.S. degree and M.S. degree in Chongqing University, P. R. China, at 2000 and 2003 respectively. Since July 2003, he acts as a lecturer in Department of Computer Science, Zhejiang University of Finance and Economy. Currently he is a full-time Ph.D. candidate in College of Computer Science, Chongqing University. His research interests include fault diagnosis, dependability analysis and design of distributed systems, cloud computing. As the first author, he has published nearly 20 papers in related research areas during recent years.



Jun Liu, he received his B.S. degree in Southwest University, P. R. China, at 2001, and M.S. degree in Chongqing University, P. R. China, at 2008. He is currently a Ph.D. candidate in College of Computer Science, at Chongqing University. His current interests include flash memory, Linux kernel and big data analytics.



MingWei Lin, he received his B.S. degree in Chongqing University, P. R. China, at 2009. He is currently a Ph.D. candidate in Chongqing University. He is invited as the reviewer by *Journal of Systems and Software*, as well as *Computers and Electrical Engineering*. His current interests include wireless sensor network, flash memory, and Linux kernel.