Low-Cost Fault-Tolerance Protocol for Large-Scale Network Monitoring*

JinHo Ahn[1], SungGi Min[2,**], YoungIl Choi[3], and ByungSun Lee[3]

[1] Dept. of Computer Science, College of Information Science, Kyonggi University
San 94-6 Yiuidong, Paldalgu, Suwonsi Kyonggido 442-760, Republic of Korea
jhahn91@hanmail.net

[2] Dept. of Computer Science & Engineering, Korea University
5-1 Anamdong, Sungbukgu, Seoul 136-701, Republic of Korea
sgmin@korea.ac.kr

[3] Network Technology Lab., Electronics and Telecommunications Research Institute
Yusong P.O. Box 106, Taejon 305-600, Republic of Korea
{yichoi,bslee}@etri.re.kr

Abstract. Distributed hierarchical network monitoring model has been proposed to solve scalability problem of centralized model. In this distributed model, a top-level monitoring manager, called main manager, obtains aggregate management information from mid-level managers, named domain managers, forming a hierarchical structure. However, if some of monitoring managers crash, network elements cannot be continuously and correctly monitored until the managers are repaired. To address this important, but previously unresolved issue, this paper presents a new fault-tolerance protocol for domain managers, named $DMFTP$, allowing the managers to efficiently utilize their organization structure. Therefore, this protocol can minimize failure detection overhead and the number of live managers affected by each manager node crash. Also, it tolerates concurrent manager failures and, after the failed managers have been repaired, ensures their immediate and consistent recovery.

1 Introduction

Network monitoring is an ability to collect traffic information from networked systems in a timely manner [4]. The traffic information is generally used in these systems for a variety of purposes such as quality of service, scheduling, capacity planning and traffic flow prediction and so on. As the systems scale up, the importance of the ability significantly increases because it is very difficult to control and manage the systems without any monitoring mechanism.

Monitoring systems should be designed to minimize network traffic incurred by them and latency in extracting the necessary information from the networked systems [5]. These systems can be classified into centralized and distributed(or

* This work was supported by Electronics and Telecommunications Research Institute Grant.

** Corresponding author. Tel.:+82-2-3290-3201; fax:+82-2-953-0771.

P.M.A. Sloot et al. (Eds.): ICCS 2003, LNCS 2659, pp. 504–513, 2003.

decentralized) based on their organization model. In the centralized monitoring model, a single manager collects information from all agents on monitored nodes and processes it. The model can be simply implemented, but become a performance bottleneck of the entire systems. Thus, the model is widely used to manage relatively small-scale networks. In order to pursue increased performance and scalability, the distributed monitoring model can be used for large-scale networked systems. In this model, monitoring managers are largely classified into main manager and domain manager depending on the role of each manager, and hierarchically organized. In other words, a hierarchical structure is formed with the main manager in top, several levels of domain managers in the middle and a collection of agents at the bottom. Thus, administrators can delegate monitoring tasks across a hierarchy of managers [2]. Each lowest-level domain manager collects management information from some agents. Similarly, a set of domain managers act as agents to a higher-level domain manager or to the main manager. The main manager obtains and processes aggregate and pre-filtered information from the domain managers. Several existing network monitoring systems based on this model were proposed to use more than one among some technologies such as SNMP, Distributed Object, Mobile Agent and so on [4].

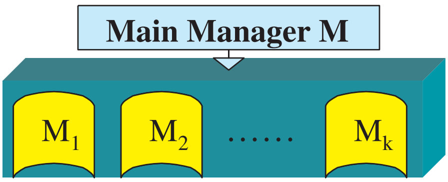


Fig. 1. A cluster of redundant main managers

However, suppose that some of managers crash in this model. In this case, network devices or nodes affected by the failures cannot be continuously and correctly monitored before the failed managers are recovered. To the best of our knowledge, few among previous works consider this important issue. Especially, if the main manager fails, the entire monitoring system may never play its role. To address the issue, our research group has performed developing a scalable and efficient fault-tolerance strategy for the model [1]. In this strategy, main manager fault tolerance is achieved by applying existing scalable replication-based protocols [3] to a cluster of redundant main managers like in Fig. 1 because the main manager should process very large volume of management information and perform significantly complex decision-making process for various applications. But, another fault-tolerance protocol is required for a hierarchy of domain managers due to their more lightweight role compared with that of the main manager. For this purpose, this paper proposes a scalable fault-tolerance protocol for domain managers, called $DMFTP$(Domain Manager Fault-Tolerance Protocol), to efficiently utilize their hierarchical structure based on the assumption that the main manager is failure-free. The protocol results in low failure detection overhead by each domain manager periodically sending a domain manager advertisement message only to its immediate super manager. Thus, when some of domain managers fail even concurrently, the protocol enables their immediate super managers to take over them. This behavior can achieve minimizing the number of live managers affected by the failures. Also, after failed managers have been recovered, it allows them to immediately play their pre-failure roles in order to improve entire monitoring system performance degraded by the failures.

The rest of the paper is organized as follows. In Sect. 2, we describe our designed protocol $DMFTP$ and in Sect. 3, prove its correctness. Finally, Sect. 4 concludes this paper.

2 Domain Manager Fault-Tolerance Protocol (DMFTP)

The proposed protocol $DMFTP$ is composed of three components, failure detection, takeover and recovery. Every manager i should maintain three variables, $MMngr_i$, $Parent_i$ and $SubMngrs_i$, for the protocol. $MMngr_i$ is the main manager's identifier needed when domain manager i recovers after repaired. $Parent_i$ is the immediate super manager's identifier of domain manager i. $SubMngrs_i$ is a tree for saving the identifier and timer of every sub-manager of main or domain manager i. Its node is a tuple (id, $timer$, sub). $timer$ for each sub-manager id is used so that manager i detects whether sub-manager id is alive or failed currently, and is initialized to a. sub for sub-manager id is a sub-tree for all sub-managers of the domain manager id. In the next subsections, basic concepts and algorithms of the three components of $DMFTP$ are described in details. Due to space limitation, there are no formal descriptions of the proposed protocol in this paper. The interested reader can find the descriptions in [1].

2.1 Failure Detection and Takeover

Every domain manager i periodically sends each domain manager advertisement message only to its immediate super manager $Parent_i$. Thus, monitoring the advertisement messages of its sub managers, the super manager $Parent_i$ can detect which managers fail or are alive among them. For example, Fig. 2(a) shows a hierarchy of managers consisting of a main manager M and twelve domain managers and Fig. 2(b), domain manager advertisement message interaction between four managers M, D_1, D_6 and D_{12}. In Fig. 2(b), domain managers D_1, D_6 and D_{12} periodically transmit each a domain manager advertisement message to their immediate super managers M, D_1 and D_6 by invoking procedure RCV_DMMNGRAM(). In this case, each super manager resets $timer$ for each corresponding sub manager to a like in Figs. 3(a)–3(c). Therefore, the protocol $DMFTP$ results in low failure-free overhead incurred by failure detection by efficiently utilizing the hierarchical structure of managers.

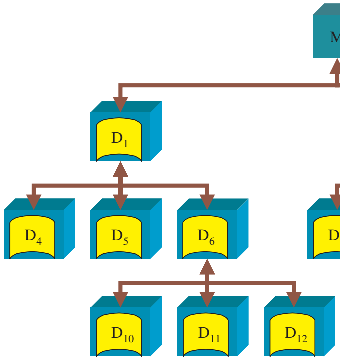
In $DMFTP$, each super manager sm decrements the timer for every sub manager by one every certain time interval. If there are some sub managers which cannot send each a domain manager advertisement message to sm until their timers become zero, sm suspects that the managers failed. In this case, it executes the takeover procedure of $DMFTP$. For example, if domain manager D_6 fails like in Fig. 4, manager D_1 cannot receive any advertisement message from D_6. Thus, the timer for D_6 eventually becomes zero like in Fig. 4(c) and so, D_1 invokes procedure TAKEOVER(). This procedure forces three sub managers of D_6, D_{10}, D_{11} and D_{12}, to call procedure UPDATE_PARENT(). In this case, like in Fig. 4(d), the sub managers set all $Parent_i$ to D_1 as their immediate super manager. Afterwards, they periodically send each an advertisement message to D_1 like in Fig. 4(b).

Also, the protocol $DMFTP$ performs the consistent takeover process even if domain managers fail concurrently. To illustrate this claim, Fig. 5 indicates an example that concurrent failures of two domain managers D_1 and D_6 occur. In this case, main manager M can receive no advertisement message from the domain manager D_1 like in Fig. 5(b). Thus, the timer for D_1 of M is finally expired like in Fig. 5(c) and then M executes procedure TAKEOVER(), which causes sub managers of D_1, D_4, D_5 and D_6, to invoke procedure UPDATE_PARENT(). However, M cannot receive any acknowledgement of the procedure from the failed manager D_6 like in Fig. 5(b). In this case, it sets the timer for D_6 to zero and then forces the sub managers of D_6 to perform procedure UPDATE_PARENT(). After having completed the takeover procedure, M can receive each advertisement message from D_{10}, D_{11} and D_{12} respectively every certain interval like in Fig. 5(b).

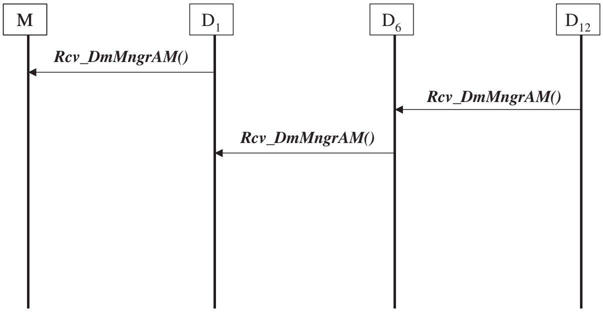
2.2 Recovery

This section describes the recovery algorithms of $DMFTP$ allowing repaired domain managers to play their pre-failure roles. For example, Fig. 6 shows that failed domain manager D_6 in Fig. 5 is recovered by executing $DMFTP$. In this example, after the manager D_6 has been repaired, it performs procedure RECOVERY(). This procedure first forces the main manager M to invoke procedure GET_PARENTS() like in Fig. 6(b) in order to obtain a list of super managers of D_6 known to be alive by M. In this case, the received list is $\{M\}$ because the manager D_1 previously failed. Thus, D_6 requires from M a tree of sub managers it managed before the failure by remotely calling procedure GET_SUBMNGRS() of M. At this point, M resets the timer for D_6 to a like in Fig. 6(c) and then causes three sub managers, D_{10}, D_{11} and D_{12}, to remotely invoke UPDATE_PARENT() so that the sub managers set all $Parent_i$ to D_6 as their immediate super manager. After having completed this procedure, D_6 obtains the tree from M like Fig. 6(d). Hereafter, they periodically send each an advertisement message to D_6 like in Fig. 6(b).

Next, $DMFTP$ can allow each repaired manager to be recovered consistently even if its immediate super manager fails concurrently when the recovery process is started. To illustrate this feature, Fig. 7 shows an example that D_1 crashes as soon as D_6 in Fig. 4 performs procedure RECOVERY(). In this case, D_6 receives a list of super managers of D_6, $\{M, D_1\}$, from M because M has not detected



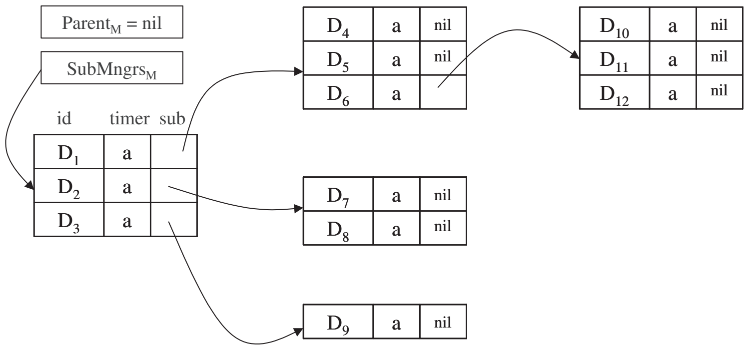
(a) a hierarchy of managers



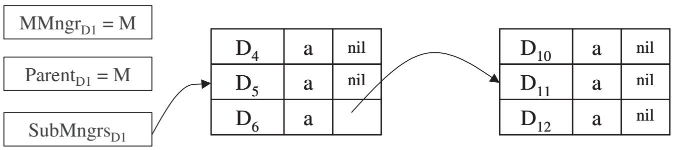
(b) Domain manager advertisement message interaction between four managers

Fig. 2. In case of no manager failure in *DMFTP* (*continued*)

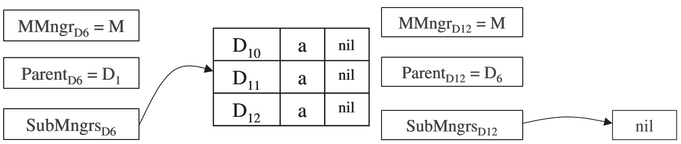
D_1's failure yet. Then, D_6 first invokes procedure GET_SUBMNGRS() of the lowest level super manager in the list, D_1, to obtain a tree of its monitored sub managers. However, like in Fig. 7, D_6 cannot get the tree from the currently crashed manager, D_1. Thus, it requires the tree from the secondly lowest level super manager, M, by calling procedure GET_SUBMNGRS() of M. At this point, M can detect D_1's failure by either D_6 or the timer for D_1. In the first case, it sets the timer for D_1 to zero and then forces the sub managers of D_1 to set all $Parent_i$ to M by invoking their procedure UPDATE_PARENT() respectively. Also, M resets the timer for D_6 to a and then causes D_6's sub managers to set their $Parent_i$ to D_6. Then, D_6 gets the tree from M. Afterwards, D_4, D_5 and D_6 periodically send each an advertisement message to M, and D_{10}, D_{11} and D_{12}, to D_6 like in Fig. 7.



(a) Data structures for the main manager M



(b) Data structures for D_1



(c) Data structures for D_6 (d) Data structures for D_{12}

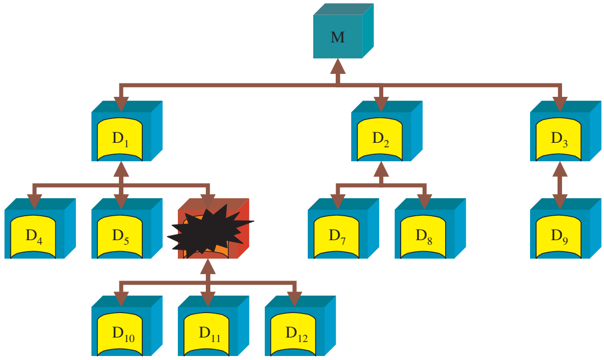
Fig. 3. In case of no manager failure in $DMFTP$

3 Correctness

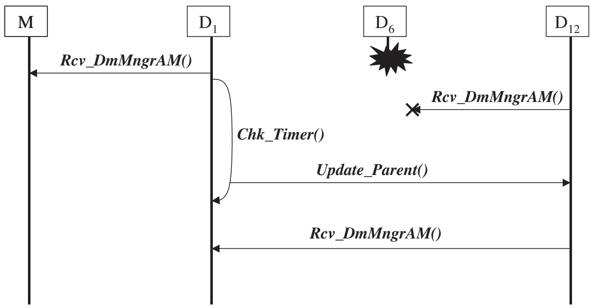
This section shows theorem 1 and 2 to prove the correctness of our takeover algorithm and recovery algorithm in $DMFTP$. Due to space limitation, the proof of the two theorems is omitted. The interested reader can find them in [1].

Theorem 1. Even if multiple domain managers crash concurrently, our takeover algorithm enables another live managers to monitor all the network elements previously managed by the failed ones. □

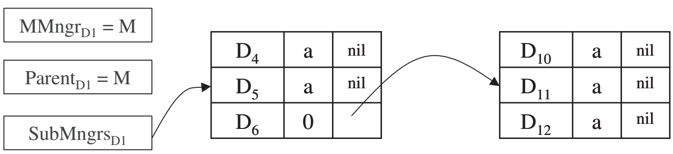
Theorem 2. After every repaired domain manager has completed our recovery algorithm, it can manage its monitored network elements before it has failed. □



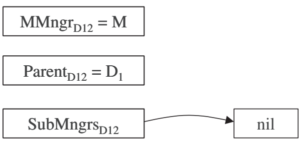
(a) a hierarchy of managers



(b) Domain manager advertisement message interaction between four managers

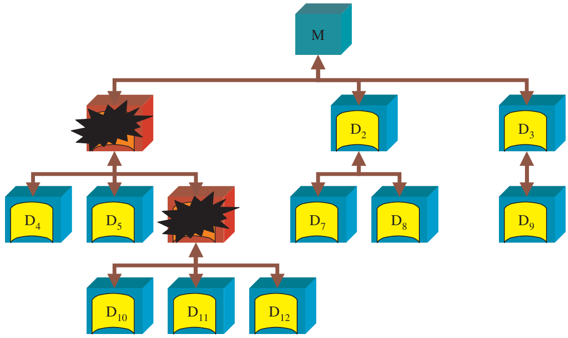


(c) Data structures for D_1

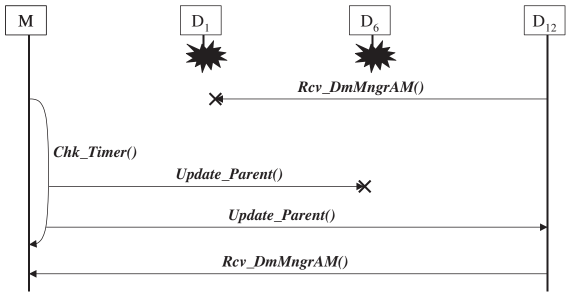


(d) Data structures for D_{12}

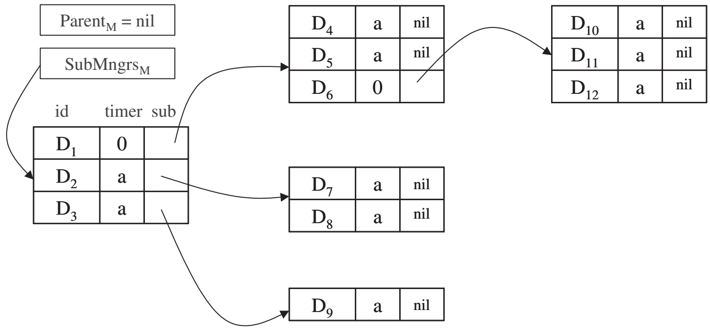
Fig. 4. In case of D_6's failure in $DMFTP$



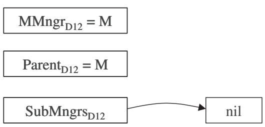
(a) a hierarchy of managers



(b) Domain manager advertisement message interaction between four managers

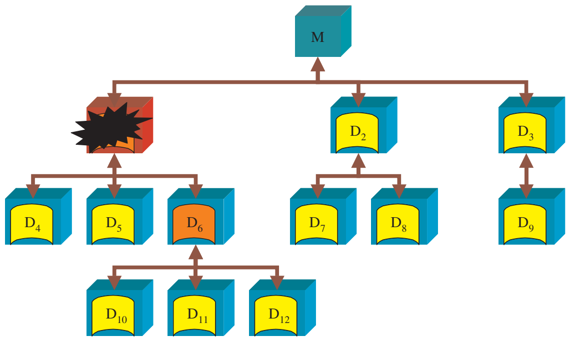


(c) Data structures for M

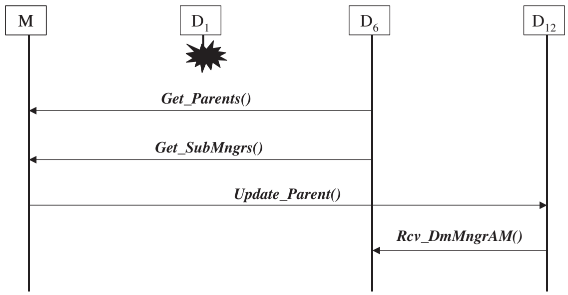


(d) Data structures for D_{12}

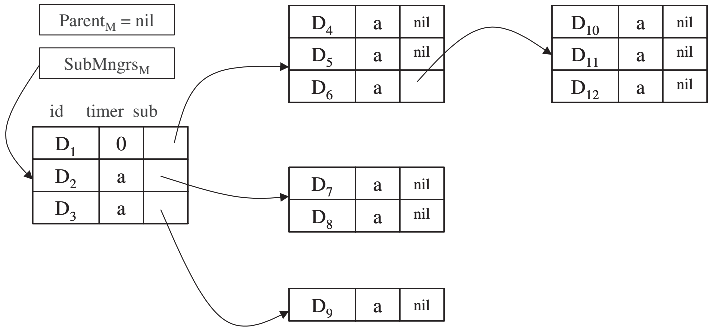
Fig. 5. In case of concurrent failures of two domain managers D_1 and D_6 in $DMFTP$



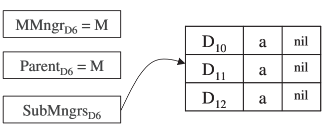
(a) a hierarchy of managers



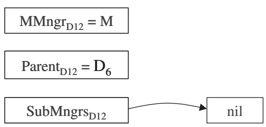
(b) Domain manager advertisement message interaction between four managers



(c) Data structures for M



(d) Data structures for D_6



(e) Data structures for D_{12}

Fig. 6. An example of recovering domain manager D_6 in figure 5

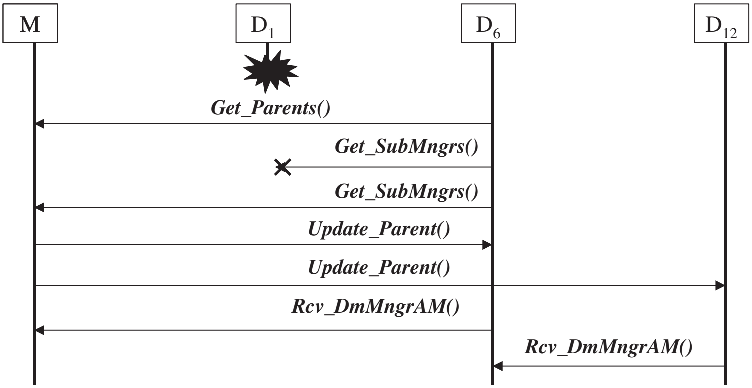


Fig. 7. In case of D_1's failure while recovering D_6 in figure 4

4 Conclusion

In this paper, a low cost fault-tolerance protocol for domain managers, $DMFTP$, was designed based on their hierarchical structure. This protocol can minimize failure detection overhead and the number of live managers affected by each manager node crash. Also, it tolerates concurrent manager failures and, after the failed managers have been repaired, ensures their immediate and consistent recovery.

Integrating $DMFTP$ with existing replication-based protocols [3] for the main manager is a research work requiring further investigation because the role of the main manager fault-tolerance protocols is very important for $DMFTP$ to work effectively [1]. Thus, we are currently examining which replication protocols are appropriate for $DMFTP$ through various experiments.

References

1. J. Ahn, S. Min, Y. Choi and B. Lee. A Novel Fault-Tolerance Strategy for Large-Scale Network Monitoring. *Technical Report KU-CSE-02-049*, Korea University, 2002.
2. G. Goldszmidt and Y. Yemini. Delegated Agents for Network Management. *IEEE Communication Magazine*, 36(3):66–70, March 1998.
3. R. Guerraoui and A. Schiper. Software-Based Replication for Fault Tolerance. *IEEE Computer*, 30(4):68–74, 1997.
4. J. Philippe, M. Flatin and S. Znaty. Two Taxonomies of Distributed Network and System Management Paradigms. *Emerging Trends and Challenges in Network Management*, 2000.
5. R. Subramanyan, J. Miguel-Alonso and J.A.B. Fortes. A scalable SNMP-based distributed monitoring system for heterogeneous network computing. *In Proc. of the 12nd ACM/IEEE International Supercomputing Conference.* Dallas, Texas, Nov. 2000.