Journal of the Korea Industrial Information Systems Research Vol. 22 No.2, Apr. 2017 :15-25
http://dx.doi.org./10.9723/jksiis.2017.22.2.015 ISSN:1229-3741

Dynamic Dependability Level Switching Strategies by Utilizing Threat Predictions

Lim Sung-Hwa[1)*]

Abstract A System can be more Dependable from some types of Threats if the Dependability Level Against the Threat on the System is Increased. However, The Dependability-performance Tradeoff should be Considered because the Increased Dependability may Degrade the Performance of the System. Therefore, it is Efficient to Temporally Increase the Dependability Level to High only when an Threat is Predicted on the System in a Short time while Maintaining the Level in Low or mid in Normal Situations. In this Paper, we Present a Threat Prevention Strategy for a Networked Node by Dynamically Changing the Dependability Level According to the Threat Situation on its Logically/physically Neighboring Nodes. As case Studies, we Employ our Strategy to an Internet Server Against TCP SYN Flood Attacks and to a Checkpoint and Rollback System Against Transient Faults. Our Performance Analysis Shows that our Strategy can Effectively Relieve the Damage of the Failure without Serious Performance Degradation.

Key Words : Dependability, Fault-tolerance, Security

1. Introduction

In current Internet environments, a lot of computing nodes (i.e., clients or servers) are logically or physically clustered. Moreover, in future Internet environments such as NDN (Named Data Networking) [1], every node may have the both roles of the consumer (i.e., client) and the provider (i.e., server). Therefore, threats such as diffusing faults or malicious attacks on a computing node may also be threats to its physically/logically neighboring nodes [2-3, 14-15]. Reasons of failures are complex and impacts of those tend to be catastrophic. Not only hardware design faults and system-specific weaknesses but also defections on system software or application software may cause augmented chain reactions of common faults on neighboring computing nodes.

Failures on the neighboring nodes can be suppressed by predicting and preventing the same fault when it has occurred on a node. Therefore, we can make a node more dependable from a threat if we increase the dependability level on the node against the threat. However, the increased dependability level brings about somewhat performance degradation. Therefore, it will be efficient to maintain the dependability level in low or mid, and temporally increase it to high in advance only when a threat is predicted in the near future. Then, the challenge is how to predict the threat on the node. We can expect that a threat may occur on a node with a high probability in a short time when one of its

* Corresponding Author : sunghwa@nsu.ac.kr
† Funding for this paper was provided by Namseoul University.
Manuscript received Jan, 24, 2017 / revised Mar, 3 ,2017 / accepted Mar, 13, 2017
1) Department of Multimedia, Namseoul University, the first and the corresponding author

physical or logical neighbor nodes is being failed by the threat. Indeed, if we detect a threat occurring on one of neighbors of the node, we can predict that the same threat will occur on the node with a high probability. At that time, the node, which detects that its neighbor have been failed, can relieve the damage of the failure by temporally increasing their dependability level against the threat.

In this paper, we present a failure prevention policy switching its dependability level dynamically by predicting the threat situation on its neighboring nodes. Moreover, as case studies, we apply our policy to a server system against the TCP SYN flood attack, which is one of the most common DDoS (Distributed Denial of Service) attacks [4] as a security issue, and a checkpoint and rollback system against transient faults as a fault-tolerance issue. Performance analysis shows that our strategy can effectively mitigate the damage of the threats without serious performance.

2. Dependent Fault modeling

Physically or logically related computing nodes may some dependency of common faults. One of our previous works presented the dependent fault modeling [5]. Let k_x be denoted as one of the faults that can incur a failure on node i, and F_i as the failure region which is a set of all faults incurring failures on node i as shown in Equation (1).

$$F_i \supset \{k_a, k_b, \ldots, k_z\} \qquad (1)$$

If two neighboring nodes (i.e., node 1 and node 2) may have common fault elements (i.e. F_C). When a fault element, which has propagating characteristic, in F_C is occurred in node 1, then the fault may be occurred in node 2 with great chance. Fig. 1 shows failure regions of node 1 and node 2, and the common failure region. Let us denote λ_1 and λ_2 as failure rates on node 1 and node 2, respectively. We set the occurrence rate of common faults (i.e., elements in F_C) as λ_C. Then we can express the failure rate of node 1 and node 2 by considering common faults as shown in Equation (2), where D_i is node i's common fault ratio between two nodes.

$$\lambda_C = D_1\lambda_1 = D_2\lambda_2 \qquad (2)$$

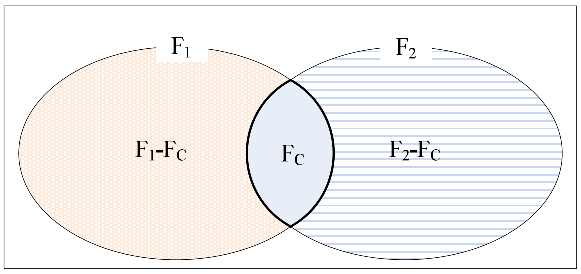


Fig. 1 Failure region model [4]

3. Dynamic Dependability Level Switching Policy

The main idea is to protect physically or logically neighboring nodes against a threat when one or more nodes are detected to being failed by the threat. The method to detect threat situations of other nodes can be performed in various ways, and is quite dependent on their implementation methods. For example, every node may scan the others' states periodically in an attempt to detect a threat situation and determine the types of the threat. On the other way, every node may broadcast an alert signal to its neighboring nodes when it is being failed.

```
Step 1. Scan state of current node to see whether it has any threat
Step 2. Check alert messages sent from neighbour nodes which are to be compromised by propagational threat
Step 3. If any propagational threat is inspected in Step2
    Step 3-1. send an alert message to all neighbor nodes about the propagational threat
    Step 3-2. Increase the fault-tolerance/security level on the node according to the cause of attack
    Step 3-3. After a while, if the threats are dismissed then restore the fault-tolerance /security level on the node to normal
Step 4. Return to Step 1 and repeat
```

Fig. 2 Dynamic Dependability Level Switching Policy

When a node has detected a threat on one of its neighboring nodes, the node temporarily increases its dependability level against the attack by employing proper fault-tolerance or security techniques. The temporarily increased dependability level should be returned to the normal level when the threat is dismissed or a predefined time has passed, because maintaining these levels highly may be the burden to the system. Fig. 2 presents the dynamic security level changing policy, which is partly inspired from the fault-prevention policy, which was previous works of our research group [5-6].

4. Case Study of Security : TCP SYN Attacks[2)]

In this section, we present an attack prevention algorithm for a server node by changing its security level dynamically according to the attack situation on its neighboring nodes as shown in Fig. 3. Moreover, as a case study, we apply our strategy to a server system against the TCP SYN flood attack, which is one of the most common DDoS attacks [4, 16].

```
#define ATTACK_CLEAN 0x0000
#define ATTACK _PROP 0x1000  // Threat is propagational
#define ATTACK _INNER 0x0010 // on the current system

Threat_Alert_N_Avoid( ) {
while (true) {
   nState=Scan_Current_Status( ); /* Scan state of current
                                     node to see whether it
                                     has any attack*/
   nState|=Scan_Alert_Msg( ); /* Check alert messages sent
                                 from neighbour nodes which
                                 are to be compromised by
                                 propagational attacks*/
   if ((nState & ATTACK _PROP) && (L_system==NORMAL)) {
       if (nState & ATTACK _INNER)
          Send_AlertMessage(nState); /* send an alert
                                        message to all neighbor
                                        nodes about the
                                        propagational threat */
       Increase_Level(); /* Increase the security level on the
                            node according to the cause of
                            attack */
       L_system = HIGH;
   }
   if ((nState == ATTACK _CLEAN) && (L_system==HIGH)){
       Restore_Level(); /* Restore the security level on the
                           node to normal */
       L_system = NORMAL;
   }
}
/*********** Parameter Descriptions    *************/
nState : the flag whether this node can continue its
         operation (Default is ATTACK_CLEAN)
L_system : the flag value whether fault tolerance/system
        security level of current system is HIGH or NORMAL.
        Default is NORMAL
```

Fig. 3 Dynamic Security Level Switching Algorithm

Fig. 4 illustrates an example simple scenario when we apply the algorithm shown in Fig 3.

2) A preliminary version of this section is presented in ICITCS 2014 [13]

The three server nodes are guarded by a firewall equipped on the gateway. One of the nodes is being compromise by an attack, and then the node disseminates an alert message enclosing the information about the attack to its neighbors. On receiving the message, the neighboring nodes prepare the attack by switching their security level to the attack higher. After the attack situation is terminated, their security level is returned to the normal level.

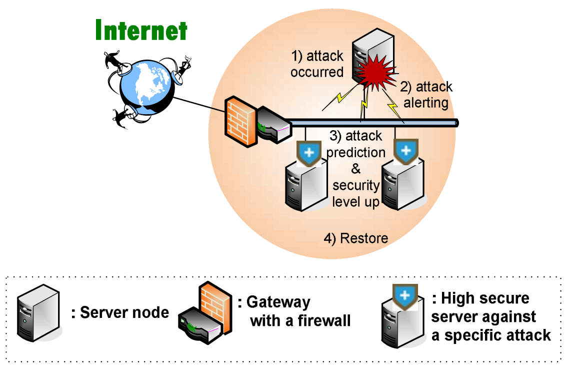

Internet
1) attack occurred
2) attack alerting
3) attack prediction & security level up
4) Restore
: Server node
: Gateway with a firewall
: High secure server against a specific attack

Fig. 4 Example scenario

As a case study, the algorithm shown in Fig. 3 is applied to a specific case, in which we are trying to prevent TCP SYN attacks on a system. TCP SYN Flooding attack, which is one of DDoS attacks, exploits the standard TCP three-way handshaking by sending great number of connection requests with spoofed source addresses to the victim system [4, 7]. This kind of attack results in the victim system refusing other normal incoming connections when the backlog queue which is buffering incoming connection is overloaded. The damage of TCP SYN attack can be relieved by shortening timeout length, by reducing retransmissions, or by increasing the backlog queue size of the victim machine. However, there are tradeoffs in these treatments to relieve the TCP SYN attack. Shortening timeout length or reducing retransmission number may increase dropping rate of normal incoming connections. Increasing backlog queue size may postpone the time the victim system being fully compromised but may consume more system resource. Therefore, it would be highly effective to adjust theses parameters dynamically considering TCP SYN attack situation.

Our suggestion to cope with the TCP SYN flooding attack is as follows. If a nodes is compromised by the TCP SYN flooding through the Internet, its neighboring nodes in the same system may also be attacked in a short time. When a node is compromised by the TCP SYN flooding, it detects the fact that a SYN flooding is being occurred and broadcasts warning message for the TCP SYN flooding to its neighboring nodes in the same system. Basically, we can detect TCP SYN flooding in a node by checking the number of current half-opened TCP connection. Even though, there are many enhanced ways presented to detect TCP SYN attacks [4, 8], finding out the effective detection scheme is out of focus of this paper[3]. Although a compromised node by TCP SYN flooding cannot allow TCP connection requests from others, it can broadcasts TCP SYN flooding warning messages to the neighboring nodes using other communication methods. The neighboring nodes which has received TCP SYN flooding warning messages can be prepared against incoming TCP SYN flooding attacks by increasing backlog queue, shortening half-opened TCP connection keep alive time, or decreasing the number of SYN retry.

The TCP connection is established using three-way handshaking when a connection request is received [10]. Fig.5 shows a Markov model of TCP connection establishment process

3) Any new enhanced detection scheme can be employed to our algorithm.

when n-retransmission is allowed, and Table 1 describes each states.

Table 1 States for Markov model of TCP connection establishment

State	**Description**
0	initial state
S	connection success state
F_i	i-th failure state
E	connection drop state

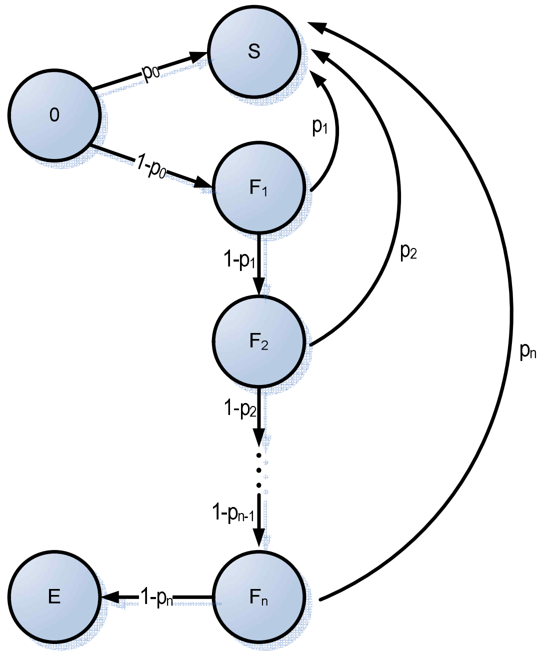


Fig. 5 Markov model of TCP connection establishment

We denote n is the maximum number of retransmission and p_i is the probability that the connection is established at i-th retry. This Markov chain starts from '0' state and turns to 'F_1' states with the probability of $1-p_0$. In general, 'F_{i-1}'state turns to 'F_i' state with the probability of $1-p_{i-1}$, and these transitions can occur until the maximum number of retransmission(n). If the number of retransmission exceeds n, then the state turns to 'E' states and the connection request is dropped. We can compute the summation of probability that the connection is succeeded as (3).

$$
\begin{aligned}
S(n) &= p_0 + (1-p_0)p_1 + (1-p_0)(1-p_1)p_2 \\
&\quad + \cdots + (1-p_0)(1-p_1)\cdots(1-p_{n-1})p_n \\
&= p_0 + \sum_{i=1}^{n}\left[\left\{\prod_{j=0}^{i-1}(1-p_j)\right\}p_i\right]
\end{aligned} \tag{3}
$$

We can express p_i as multiplex of p and $f(x)$ as (4). p is the probability that the connection is succeeded when the timeout is infinite. There are some cases that the connection is due to breakdown of communication line or routers, etc. $f(t)$ is the connection success function of timeout t which tells how the connection success rate is affected as the function of timeout t. Maximum timeout is doubled as retransmission increases followed by the exponential back-off [9]. Therefore, the maximum timeout at 'F_i' state would be $2^i k$ where k is the initial timeout value (e.g., k would be three seconds in some LINUX systems) [9]. In this paper, we assume that time to successful TCP connections follows PDF (probability density function) of normal distribution with mean value μ and standard deviation σ. Therefore, $f(t)$ which is connection success function of timeout t is following CDF (cumulative density function) of normal distribution as depicted in (5).

$$p_i = f(2^i k)p \tag{4}$$

$$f(t) = \frac{1}{\sigma\sqrt{2\pi}}\int_{-\infty}^{t} e^{-\frac{(x-\mu)^2}{2\sigma^2}}dx \tag{5}$$

, where $\mu > 0$

As shown in (6), we can compute the total success number of connection requests during time T when request rate of TCP connection is β and the maximal retransmission number is n.

$$S_{connect}(T,n) = \left\{\begin{array}{l} f(2^0k)p + (1-f(2^0k)p)f(2^1k)p \\ + (1-f(2^0k)p)(1-f(2^1k)p)f(2^2k)p \\ + \cdots + (1-f(2^0k)p)(1-f(2^1k)p) \\ \ldots (1-f(2^{n-1}k)p)f(2^nk)p \end{array}\right\} \times \beta T \qquad (6)$$

$$= \left[f(2^0k)p + \sum_{i=1}^{n}\left\{\prod_{j=0}^{i-1}(1-f(2^jk)p)\right\}f(2^ik)p\right] \times \beta T$$

Upon TCP SYN attack, the system is not compromised if $a_r \times T_{out} < S_Q$, where a_r is the number of SYN attacks per second, T_{out} is the maximal expiration time for non-response connections and S_Q is the size of back log queue. T_{out} is shown as (7) when we assume that exponential back-off retransmission is applied. In (7), k is the initial timeout value, n is the maximal retransmission number, and h is the miscellaneous overhead time such as the delay between two time points a connection is determined to be dropped and really dropped, etc.

$$T_{out} = \left(\sum_{i=0}^{n} 2^i k\right) + h \qquad (7)$$

If $a_r \times T_{out} \geq S_Q$, the system will be compromised after time T_{fail}, which is computed as (8).

$$T_{fail} = \frac{S_Q}{a_r} \qquad (8)$$

We conducted performance analysis to show the effectiveness of our strategy. Table 2 describes system parameters for our performance analysis.

Table 2 System parameters

Parameter	Description	Default
T	Total Time (seconds)	3600
k	initial RTO (seconds)	3
h	time out overhead	0
S_Q	Default size of Backlog queue (number of entries)	1024
β	Request rate of normal connections	0.01
p	probability of one connection is succeed when T_{out} is infinite	0.9
μ	Mean value for the normal distribution function for the connection success function	8
σ	Standard deviation for the normal distribution function for the connection success function	5

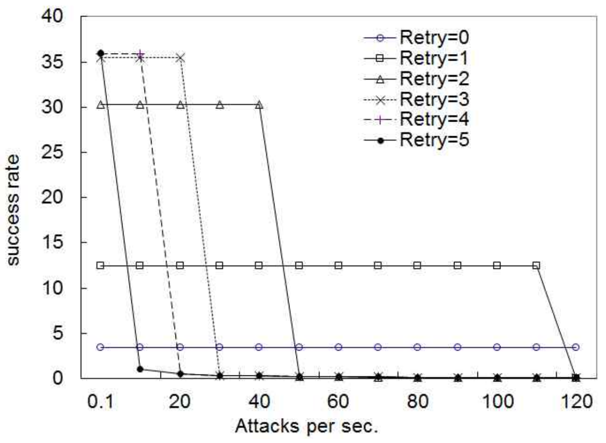
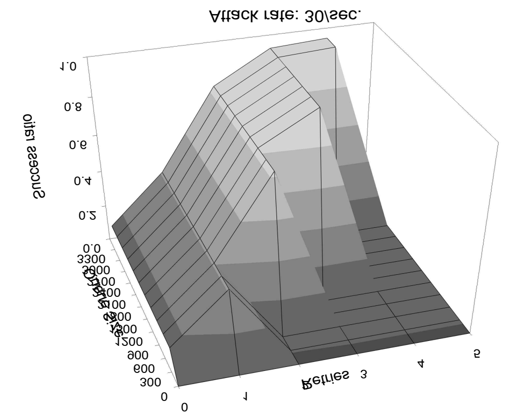


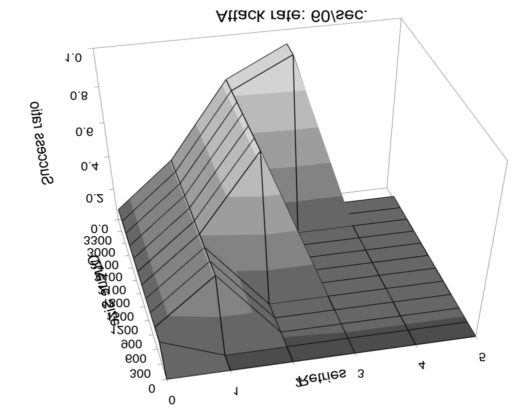
Fig. 6 TCP connection success rate by varying retry numbers in TCP attack situation

Fig. 6 shows TCP connection success rate by varying maximum retransmission numbers and number of attacks per second on TCP

SYN attack situation. TCP connection success rate means how many incoming normal connections are accepted and established during time T.



(a) Attack rate of 30 per second



(b) Attack rate of 60 per second

Fig. 7 TCP connection success rate by varying backlog queue size in TCP attack situation

We assume that T is 3600 seconds, k is 3, h is 0, S_Q is 1024, β is 0.01, P is 0.9, μ is 8, and σ is 5. By decreasing the maximum retransmission number, we can relieve the damage of TCP SYN attack. However, with a decreased maximum retransmission number, some normal connections are refused even if SYN attack is not occurred - the default maximum retransmission in LINUX system is 5.

Fig. 7 shows TCP connection success rate by varying backlog queue sizes (S_Q) and maximum retransmission numbers when the number of TCP SYN attack traffic is 30 and 60 per second, respectively. Increasing backlog queue size could relive the damage of TCP SYN attack at both of two attack rate cases.

As shown in our performance analysis, we can see that dynamical change of retransmission number and backlog queue size in the sense of TCP SYN attack is able to relive the damage of the attack with minimized side effects. When a node has been compromised, the damage of TCP SYN attack on neighboring nodes could be relieved by sensing the attack and dynamically changing effective parameters (e.g., retry number and backlog queue size).

5. Case Study of Fault-Tolerance : Checkpoint and roll-back

In this section, we apply the dynamic dependability policy shown in Fig. 2 to the fault-tolerance case. Checkpoint and rollback recovery is a cost-effective method of providing fault tolerance against transient and intermittent faults. We can reduce execution time of a task by periodically saving process's state on a stable storage as checkpoints and by rolling back to a recent checkpoint a failure. When we define λ is failure rate and C is checkpoint overhead of a system, we can compute the optimal checkpoint interval and expected execution time during one checkpoint interval for a task as Equation (9) and Equation (10), respectively [11-12].

When we define λ is failure rate and C is checkpoint overhead of a system, we can compute the optimal checkpoint interval and expected execution time during one checkpoint interval for a task as Equation (9) and Equation (10), respectively [11-12].

$$T_C = \sqrt{\frac{2C}{\lambda}} \tag{9}$$

$$\Gamma = \lambda^{-1} e^{\lambda R}\left(e^{\lambda (T_C + C)} - 1\right) \tag{10}$$

When a node is failed by a fault, the failure rate of neighboring nodes by the same fault would not be as same as its original failure rate any more. Therefore, the optimal checkpoint interval for neighboring nodes should be re-computed after parts of the nodes are failed.

We define the optimal checkpoint interval of two neighboring nodes (i.e., node 1 and node 2) as Equation (11). We assume that failure rate of node 1 and node 2 are λ_1 and λ_2, respectively.

$$T_{C_1} = \sqrt{\frac{2C}{\lambda_1}}, \quad T_{C_2} = \sqrt{\frac{2C}{\lambda_2}} \tag{11}$$

By considering common fault dependency presented in section 3, the optimal checkpoint interval of each node should be modified as Equation (12) when any neighboring node has been failed. The failure rate of remaining node would be changed to the affected failure rate (λ_{12} or λ_{21}), which can be computed from Equation(1) and Equation (2). $T_{C_{21}}$ is the optimal checkpoint interval of node 2 when node 1 is failed, and $T_{C_{12}}$ is vice versa. As mentioned in Section 3, D_1 and D_2 is the common fault ratio of node 1 and node 2 between two neighboring nodes, respectively. m is averaged impact factor of common faults between node 1 and node 2.

$$T_{C_{21}} = \sqrt{\frac{2C}{\lambda_{21}}} = \sqrt{\frac{2C}{(\lambda_1 - D_1\lambda_1 + mD_1)}} \tag{12}$$
$$T_{C_{12}} = \sqrt{\frac{2C}{\lambda_{12}}} = \sqrt{\frac{2C}{(\lambda_2 - D_2\lambda_2 + mD_2)}}$$

($T_{C_{ij}}$: the optimal checkpoint interval for node j when node i has been failed)

We compared the performance between the conventional checkpoint scheme which has static optimal checkpoint intervals and the proposed checkpoint scheme which has dynamic interval. Assumptions and system parameters are as follows.

- The system should roll back to the last checkpoint when a fault occurs during an execution because one of neighboring nodes has been failed over already.
- We assume that $\lambda_1 = \lambda_2$, $\lambda_{12} = \lambda_{21} = \lambda_d$, and $D_1 = D_2 = D$.
- Intervals among successive failures are assumed to be the exponential distribution.
- Total execution time (T) for a task is 1000 seconds, checkpoint overhead (C) is 1 second, roll back overhead (R) is 1 second.
- Simulation is repeated 10000 times.

Fig. 8 shows the expected execution time for a task of the proposed strategy (*dynamic*) and conventional one (*static*) by varying common fault ratio (D) where failure rate (λ) is 0.001 and common fault impact factor (m) is 0.1. We can see that *dynamic* scheme shows better performance than *static* scheme as D increases.

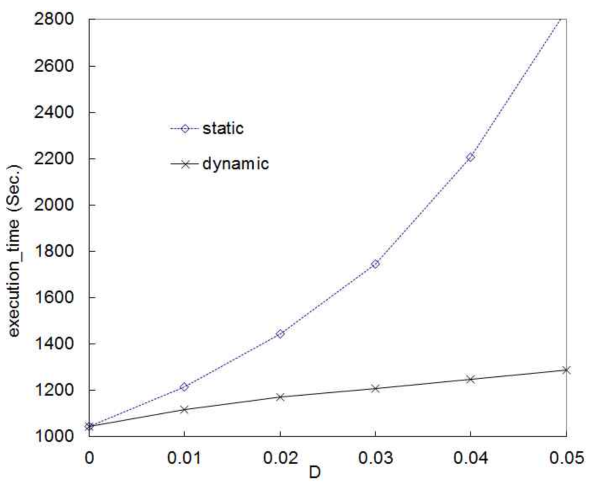


Fig. 8 Expected execution time by varying D ($\lambda = 0.001, C = 1, R = 1, T = 1000, m = 0.1$)

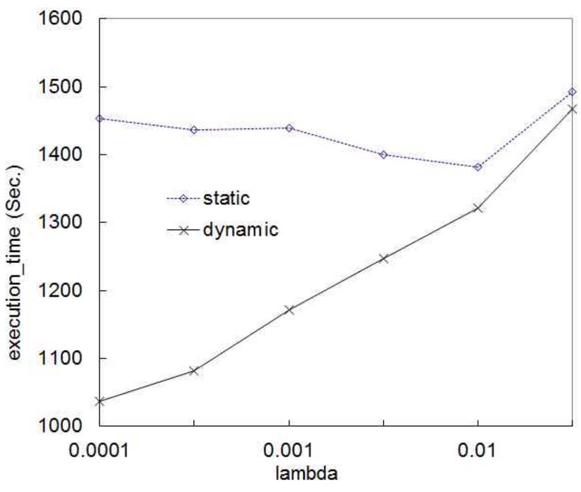


Fig. 9 Expected execution time by varying failure rate (λ) ($D = 0.2, C = 1, R = 1, T = 1000, m = 0.1$)

Fig. 9 shows the expected execution time of the proposed strategy (*dynamic*) and conventional one (*static*) by varying failure rate (λ) where D is 0.2 and $m = 0.1$. We can see that *dynamic* scheme shows better performance than *static* scheme in all cases. It seems odd for the execution time slightly decreases as the initial failure rate increases from $\lambda = 0.0001$ to $\lambda = 0.01$ in *static* scheme. The remaining node is failed by the affected failure rate (λ_d) when a node is failed by common fault. λ_d is far larger than λ when D is above 0. In *static* scheme, the checkpoint interval is determined by the initial failure rate (λ) while it should be computed by the affected failure rate (λ_d). Therefore, the checkpoint interval is computed far larger than its real optimal interval and this increases the expected execution time. The computed checkpoint interval decreases and be getting similar to its optimal checkpoint interval when λ increases. That is because the execution time slightly decreases as the failure rate increases from $\lambda = 0.0001$ to $\lambda = 0.01$ in *static* scheme.

6. Conclusion

In this paper, we present a threat prevention policy for a networked computing node by dynamically changing its dependability level according to the threat situation on its neighboring nodes. As case studies, we apply our policy to a system against TCP SYN flood attacks as a security problem and to a checkpoint and roll back system against transient faults as a fault-tolerance problem. Our performance analysis shows that our strategy can effectively mitigate the damage of the attack or faults without serious performance degradation. In the future, we will apply our policy to various security or fault-tolerance cases and conduct experimental measurements.

References

[1] Zhang L., Afanasyev A., Burke J., Jacobson V., Claffy K., Crowley P., Papadopoulos C.,

Wang L. and Zhang B., "Named Data Networking," ACM SIGCOMM Computer Communication Review, Vol. 44, pp. 66-73, July 2014.

[2] French A., and Shropshire J., "Handheld versus Traditional Computer Security Threats and Practices : A Comparison of End User Perceptions," Journal of Internet Electronic Commerce Research, Vol. 11, No. 2, pp. 153-171, June 2011.

[3] Kim J.-K., Jeon J.-H., and Lim H.-S., "The Effects of Information Security Policies, Security Controls and User's Characteristics on Anti-Virus Security Effectiveness," Journal of Information Systems, Vol. 15, No. 1, pp. 145-168, March 2006.

[4] Vellalacheuvu H. K., and Kumar S., "Effectiveness of Built-in Security Protection of Microsoft's Windows Server 2003 against TCP SYN Based DDoS Attacks," Journal of Information Security, Vol. 2, No. 3, pp. 131-138, July 2011.

[5] Lim S.-H.,. Lee B.-H, and Kim J.-H., "Diversity and Fault Avoidance for Dependable Replication Systems," Information Processing Letters, Vol. 108, pp.33-37, 15 Sep. 2008.

[6] Lim S.-H, "Replication Schemes for Dependable and Effective Mobile Systems," Ph.D. Dissertation, Ajou University. Feb. 2008.

[7] Schuba C., Krsul I., Kuhn M., Spafford E., Sundaram A. and Zamboni D., "Analysis of a Denial of Service Attack on TCP," Proc. of IEEE Symposium on Security and Privacy, May 1997.

[8] Bellaiche M. and Greoire J.-C., "SYN Flooding Attack Detection by TCP Handshake Anomalies," Security and Coomunication Networks, Vol 5, Issue 7, pp. 709-724, July 2012.

[9] Postel J., "Transmission Control Protocol - DARPA Internet Program Protocol Specification," RFC793, USC/Information Sciences Institute, Sept. 1981.

[10] Branden R., "Requirements for Internet Hosts - Communication Layers," RFC 1122, Internet Engineering Task Force, October 1989.

[11] Vaidya N. H., "On Checkpoint Latency," Proc. of the 1995 Pacific Rim International Symposium on Fault-Tolerant Systems, pp. 60-65, Dec. 1995.

[12] Young J. W., "A First Order Approximation to the Optimum Checkpoint Interval," Communications on the ACM, Vol.17, pp.530-531, Sept.1974.

[13] Lim S.-H. and Kim J.-H., "Dynamic Security Level Changing Strategy using Attack Predictions-Case Study of TCP SYN Attacks," International Conference on IT Convergence and Security (ICITCS 2014), Beijing, Oct. 2014.

[14] Yoon S., Lee S. S., and Kim S.-H., "Seamless and Secure Service Framework using Multiple Network Interfaces Terminal in Heterogeneous Environment," Journal of the Korea Society Industrial Information System, Vol. 16, No. 4, pp. 53-65, Dec. 2011.

[15] Kim S.-Y., Lee S. H., and Hwang H.-S., "A Study of Factors Affecting Attitude Towards Using Mobile Cloud Service," Journal of the Korea Society Industrial Information System, Vol. 18, No. 6, pp. 83-94, Dec. 2013.

[16] Chang B.-H., Na J.-C., and Jang J.-S., "Network Security Situational Awareness using Traffic Pattern-Map," Journal of the Korea Society Industrial Information System, Vol. 11, No. 3, pp. 34-39, Sept. 2006.

임 성 화 (Lim Sung-Hwa)

- 정회원
- 아주대학교 정보및컴퓨터공학부 공학사
- 아주대학교 정보통신공학과 공학석사
- 아주대학교 정보통신공학과 공학박사
- 남서울대학교 공과대학 멀티미디어학과 조교수
- 관심분야 : 모바일 시스템, 실시간 시스템, 분산시스템, 결함허용 시스템