A Truant Failure Detection Algorithm for Multi-Policy Distributed Systems

Yoshifumi Manabe Shigemi Aoyagi

NTT Basic Research Laboratories
3-1 Morinosato-Wakamiya, Atsugi-shi, Kanagawa 243-01 Japan

Abstract

In recent autonomous decentralized systems, every node might not execute the same algorithm because it might have its own local policy and follow that policy rather than the common principle. In this paper, we model these systems as a multi-policy distributed system. We introduce a new type of failure, a truant failure, on multi-policy distributed systems, which is considered to be the simplest local policy. A truant failure node does nothing for the other nodes' requests selected by its local policy. This paper shows a condition to be able to detect a truant failure and presents a distributed truant failure detection algorithm for that case.

1 Introduction

In a distributed system developed within one organization (i.e., one university or one company) it can be assumed that all of its subsystems obey the same rules. Most distributed algorithms, therefore, assume that every node executes the same program [14]. Though there are some cases where special nodes execute different programs, the execution is contingent upon all the other nodes' permission.

This assumption does not hold for current autonomous decentralized systems where subsystems are spread out among various organizations. Each subsystem obeys its organization's rule and might not obey the common rule. There is no control over the diversification.

For example, consider a routing problem in an internet [11]. The unit of routing is called an AS (Autonomous System). The ASs are connected by a network. Each AS must maintain its routing table. The table consists of pairs of a destination AS and the route to it. No AS knows the whole topology of the network. The ASs obtain the routing table as follows. First, each AS sets the route from itself to neighboring ASs. It then sends the routing information to its neighbors and receives routing information from them. From the received routing information, it can set the route from itself to ASs whose distance is less than two. The AS continues sending and receiving routing information between neighboring ASs and obtains routes to distant ASs. This procedure is the common rule of maintaining a routing table.

In maintaining a routing table, policy routing allows each AS to decide routes and control the forwarding of routing information according to its local policy [2]. For example, even if the path from AS X to AS Y via AS Z is longer than the one via AS W, X can select the path via Z as the route to Y because of X's preference unknown to the others. In addition, if X does not want to send some routing information to neighbor Z, it can stop forwarding information. Some protocols for policy routing have been proposed. BGP (Border Gateway Protocol) is one of such protocol [10].

Now let us consider the following case where BGP is used. If all neighboring ASs of AS Z stop forwarding routing information to Z, Z cannot get enough information to send messages to any other AS. BGP does not define how each AS selects its routing policy, so there is no way to detect or avoid this undesirable situation. This problem cannot be considered in the context of current distributed algorithms. As a basis for solving it, a new theoretical model of such distributed systems is therefore necessary.

This paper proposes a multi-policy distributed system as a model of distributed systems in which each subsystem works on different execution principles. It can model the above route preference policy and forward stopping policy in BGP.

Each node can have various kinds of local policies. As one of the most fundamental local policy, we consider a truant failure. A truant failure node does noth-

0-8186-7087-8/95 $4.00

ing for the other nodes' requests selected by its local policy. This paper shows a condition to be able to detect a truant failure and presents a distributed truant failure detection algorithm for that case.

Section 2 presents the model of multi-policy distributed systems. In Section 3, the truant failure detection algorithm is shown. Section 4 mentions issues for further study.

2 Multi-policy distributed system model

A multi-policy distributed system is defined as a tuple (N, P, LP), where N is the network topology, P is the set of distributed problems that must be solved on the network, and LP is the policy for each process. As will be shown below, the definition of N and each problem in P is the same as that for distributed algorithms [14]. N is a tuple (G, K), where $G = (V, E)$ is a undirected graph and K is network information. The nodes in V represent processors and the edges in E represent bidirectional communication lines. Communication between nodes is done by message passing over the edges. Each edge is assumed to be error-free, and FIFO (First-In, First-Out). The message transmission delay on an edge is arbitrary but finite, that is, the communication is asynchronous. Node v_i has a list of its adjacent edge set $adj\_list(v_i)$, though v_i might not know the node connected by each edge. Network information K is the set of knowledge each node has of network topology. For example, $K = \{G$ is a ring.$\}$.

A distributed problem p must be solved on N. Some (arbitrary, but not empty) set of nodes receives a locally originated request. The request means "Start solving p". The nodes begin to solve p. The other nodes begin to solve p by receiving a message from another node. The nodes execute the same algorithm. When the execution is terminated, p is solved.

In this paper, we assume that there is a set of problems (P) to solve. As shown in Theorem 1, if there is only one problem to solve and a node executes by its local policy, it is impossible for other nodes to detect the fact. Note that several problems might be solved simultaneously. For a message m, let $p(m)$ be the problem for which m is sent to solve. The initiation of problem solving is the same as in conventional distributed algorithms. Each node executes as follows:

1. There is an algorithm $A = (AS, AM, AI)$ to solve the problem set P. AS is a function $S \times M \rightarrow S$, where S is the set of internal states and M is the set of the tuple (c, m) of a message and the edge it passes. AM is a function $S \times M \rightarrow 2^M$. $AI \in S$ is the initial state. AS defines the internal state transition and AM defines the message sending on each node. A is common to all nodes.

   Note that A might be parameterized by the identifier of each node and thus the algorithm might differ among nodes.

2. Node $v_i \in V$ has its local policy $LP_i = (LPS_i, LPM_i, LPI_i)$. LPS_i, LPM_i are functions from $S \times M \times S \times 2^M \rightarrow \{true, false\}$. $LPI_i \in S$ is the initial state of v_i. LPI_i might be different from AI.

   If v_i receives a (locally originated or neighbor-originated) message m when v_i's current state is s_i, the next state s_i' is defined as

   $$s_i' = \begin{cases} s_A & \text{if } LPS_i(s_i, (c, m), s_A, M_A) = true \\ s_i & \text{if } LPS_i(s_i, (c, m), s_A, M_A) = false \end{cases}$$

   where $s_A = AS(s_i, (c, m))$ and $M_A = AM(s_i, (c, m))$. The sending message set M' is defined as

   $$M' = \begin{cases} M_A & \text{if } LPM_i(s_i, (c, m), s_A, M_A) = true \\ \phi & \text{if } LPM_i(s_i, (c, m), s_A, M_A) = false \end{cases}$$

   Here, ϕ means that v_i sends no messages.

When v_i receives a message m, v_i changes its state only if LPS_i allows the state transition. Similarly, v_i sends messages only if LPM_i allows the message transmission.

A node with its own local policy can also be considered a node with a failure. A node without a failure can be considered as $LPS_i = LPM_i = true$ and $LPI_i = AI$. In conventional distributed algorithms, the most common failures are crash failure, omission failure [4], and Byzantine failure [6]. A static crash failure node can be considered a node that satisfies $LPS_i = LPM_i = false$. Thus, static crash failure can be considered as a special case of multi-policy distributed systems.

Nodes with omission failure do nothing for some messages. The difference between omission failure and local policy is that a node with a local policy acts deterministically, i.e., it does nothing for fixed situations. On the other hand, an omission failure node acts non-determistically in that it may do nothing in any situations. Thus, a local policy can be considered as a special case of omission failure.

The nodes with Byzantine failure try to prevent other nodes from solving P. They might send false

messages intentionally. Byzantine failure is theoretically the worst kind. Since an algorithm that can tolerate Byzantine failures can tolerate any other type of failures, Byzantine failure has been widely discussed. However, algorithms that tolerate omission or Byzantine failure do not exist for most cases, especially when the communication is asynchronous [5][6].

Our model of multi-policy distributed systems therefore does not consider omission failure or Byzantine failure nodes. Our model is intended as a basis for discussing more realistic cases.

Let us consider the routing example again. The policy routing can be expressed by this model. Assume that A is the conventional routing algorithm where each node uses the shortest path to another node and sends all of its routing information to all neighboring nodes. BGP allows each node to select the route, even if it is longer than the shortest path. This policy can be expressed by LPS_i as follows:
$LPS_i(s_i,(c,m),s_A,M_A) = false$ if $p(m) =$'*routing*' and s_A sets a non-preferable route as the new route.

BGP also allows each node to stop forwarding routing information. This policy can be expressed by LPM_i as follows:
$LPM_i(s_i,(c,m),s_A,M_A) = false$ if $p(m) =$' *routing*' and M_A contains a non-preferable routing information.

Many kinds of other local policies can be considered. Some examples are:

- Truant: A node does nothing for problems that it does not want to do.
- No spontaneous execution: A node does nothing spontaneously.
- No update: A node continues using old data even if other nodes indicate it is old since it thinks the correction might cause worse side effects.

With the above local policies, the nodes might not be able to solve P. Thus it is necessary to detect whether the local policies obstruct the solving of P and how the obstruction can be avoided. For the first step of the detection problem, we consider the case of the truant policy which is considered as the most simple local policy.

3 An algorithm to detect truant failure

This section discusses a truant failure detection problem. Some definitions are given for preparation. First we define a truant failure.

Definition 1
The local policy $LP_i = (LPS_i, LPM_i, LPI_i)$ *of a truant failure node* v_i *is* $LPS_i(s_i,(c,m),s_A,M_A) = LPM_i(s_i,(c,m),s_A,M_A) = F_i(p(m))$, $LPI_i = AI$, *where* $F_i : P \rightarrow \{true, false\}$ *is* v_i*'s truant function.* ■

Truant failure nodes execute A for problem p that satisfies $F_i(p) = true$ and do nothing for problem p that satisfies $F_i(p) = false$. Generally, there is no algorithm for detecting a truant failure node. This is set forth the following theorem.

Theorem 1 *If there is a node* v_i *that satisfies* $F_i(p) = false$ *for all* $p \in P$, *there is no algorithm to detect that a node is a truant failure node.* ■

The proof for Theorem 1 is similar to that of the reaching agreement in asynchronous systems [5].
(Proof) Consider the following two cases.
(1) $F_i(p) = false$ for all $p \in P$.
(2) $F_i(p) = true$ for all $p \in P$ and the communication delay between v_i and its neighboring nodes is very long.
There is no way for v_i's neighboring nodes to distinguish these two cases within a finite time. ■

If $F_i(p) = false$ for all $p \in P$, there is no value for v_i to be a member of a distributed system. Thus, such a node does not exist in most cases.

On the other hand, if there is another problem $p' \in P$ that satisfies $F_i(p') = true$ and its existence is known to the other nodes (the fact $F_i(p') = true$ can be easily detected by receiving a message of p' from v_i), there can be a case for the other nodes to detect whether $F_i(p) = false$ by using messages for p'. Theorem 2 gives a sufficient condition to be able to detect it.

For preparation, we give some definitions and assumptions.

Assumption 1 *Each message* m *includes the value of* $p(m)$. *No two messages for different problems are merged into one message.* ■

Assumption 2 *At any time, any node can begin to solve any problem. Algorithm A terminates its execution after broadcasting that problem solving is finished.* ■

Assumption 1 is common in networking. For example, an IP datagram in TCP/IP protocols has a field

that carries protocol information [3]. Assumption 2 is also common in some cases. For example, in the BGP protocol a node begins sending routing information when it detects a change in adjacent edges. This change might happen at any time, so a node can begin execution at any time. Some algorithms send a 'terminate' message to all nodes when they finish execution (for example, election algorithms [8]). The overhead for broadcasting termination is at most $O(|E|)$ and that is not very large in many cases. Thus, the above assumptions hold for many distributed systems.

Assumption 3 *Every message has a field 'start'. If node v_i begins solving p spontaneously, v_i sets the 'start' field to true for every message until it receives a message to solve p from another node. v_i sets the 'start' field to false for the other messages.* ■

This assumption does not increase the message size by much (one bit is sufficient).

Definition 2 *A function $st : M \rightarrow \{true, false\}$ is defined as the content of the 'start' field of the message.* ■

Definition 3 *Let $c = (u, v)$ be a unidirectional communication link from u to v. It is one of the bidirectional communication line represented by $(u, v) \in E$. In the following, a unidirectional communication link is written as a link. For a link $c = (u, v)$, $sr(c)$ means its source node u and $ds(c)$ means its destination node v.*

Communication sequence $cs = \{(c_1, M_1), (c_2, M_2), \ldots, (c_k, M_k)\}$ is a set of pairs of a link and a message sequence that passes the link.

For a message sequence M and message m, $m \in M$ iff m appears in M.

For two communication sequence $cs = \{(c_1, M_1), (c_2, M_2), \ldots, (c_k, M_k)\}$ and $cs' = \{(c_1, M_1'), (c_2, M_2'), \ldots, (c_k, M_k')\}$, the concatenation $cs \cdot cs'$ is defined as $\{(c_1, M_1 \cdot M_1'), (c_2, M_2 \cdot M_2'), \ldots, (c_k, M_k \cdot M_k')\}$.

For two communication sequence cs and cs', $cs \prec cs'$ iff for all $i (1 \leq i \leq k)$, M_i is a subsequence of M_i' or $M_i = M_i'$.

If a $cs = \{(c_1, M_1), (c_2, M_2), \ldots, (c_k, M_k)\}$ satisfies $ds(c_i) = v (1 \leq i \leq k)$, it is called an input sequence for v and denoted as is_v. If a cs satisfies $sr(c_i) = v (1 \leq i \leq k)$, it is called an output sequence for v and denoted as os_v. ■

When an input sequence is_v is sent to a node v, the output sequence in response to is_v depends on v's internal state and the arrival order of messages from different links because of the communication delay. It is impossible for the other nodes to know the message arrival order or the internal state of a node. Thus, even if v executes A and A is deterministic, the other nodes cannot predict the output sequence from v uniquely. The output from v can therefore be expected to be a set of communication sequences, which are defined as a possible output sequence below. When a communication sequence is observed in response to an input sequence, the nodes might send additional messages to test node v. The additional test messages are defined as a function from a set of v's possible output sequence to a set of v's input sequence. This function is defined as an input sequence function below.

When detecting a truant failure, the nodes must not decide the next input sequence after they receive all output messages. Consider the case where cs_1 and cs_2 are included in the possible output sequence and $cs_1 \prec cs_2$. If the nodes are waiting for cs_2 and cs_1 actually appears, the nodes wait forever. Thus, we define an output sequence as a subset of a possible output sequence.

Definition 4 *The set of the communication sequences that may occur in response to input sequence is_v when v is executing A is called a possible output sequence set for is_v and denoted as $\tilde{OS}(is_v)$. The output sequence in response to input sequence is_v is defined as $\{cs \in \tilde{OS}(is_v) |$ there is no $cs' \in \tilde{OS}(is_v)$ such that $cs' \prec cs\}$ and denoted as $OS(is_v)$.*

An input sequence function IS_v is a function from the set of v's output sequences to the set of v's input sequences.

An input sequence function from $OS(is_v)$ to a set of input sequences (next input to v) is denoted as $IS_v^{(2)}$.

The possible output sequence set in response to $is_v \cdot is_v^{(2)}$ is denoted as $\tilde{OS}_v^{(2)}$ and the output sequence is denoted as $OS_v^{(2)}$. $IS_v^{(i)}, OS_v^{(i)} (i = 3, 4, \ldots)$ is defined similarly. Also, let $IS_v^{(1)} = is_v, OS_v^{(1)} = OS(is_v)$. ■

Next, for a set of messages, let us define the set of problems for which the messages are used.

Definition 5 *For a set of communication sequence CS, let $P(CS) = \{p(m) | m \in M$ for some $(c, M) \in CS\}$.* ■

Lastly, the causal relation between events is defined.

Definition 6 *[9]*

- *When an node executes event b after a, $a \rightarrow b$.*
- *For a message m, if its send event is $s(m)$ and its receive event is $r(m)$, $s(m) \rightarrow r(m)$.*
- *When $a \rightarrow b$ and $b \rightarrow c$, $a \rightarrow c$.* ■

Using the above definitions, a condition for detecting truant failure is set forth the following theorem.

Theorem 2 *A sufficient condition for the nodes $V - \{v\}$ to be able to decide whether $F_v(p) = true$ is that $G - \{v\}$ is connected, each node knows its neighboring node, and there are an input sequence $IS_v^{(0)}$ and a sequence of input sequence functions $IS_v^{(1)}, \ldots IS_v^{(k)}$ that satisfy the following:*

- *For any message $m \in IS_v^{(0)}$, $p(m) = p$.*
- *For any $p' \in P(IS_v^{(1)}, IS_v^{(2)}, \ldots, IS_v^{(k)})$ and for all $v_i \in V$ (includes v), $F_i(p') = true$ and that fact is known to all nodes.*
- *Let $OS_v^{(0)} = \{cs_1, cs_2, \ldots, cs_a\}$ and $OS_v^{(k)} = \{cs'_1, cs'_2, \ldots, cs'_b\}$.*

  For any pair of cs_i and $cs'_j (1 \leq i \leq a, 1 \leq j \leq b)$, there is a tuple $(c_{ij}, m_{ij}, m'_{ij})$ that satisfies the following:

  - *$m_{ij} \in M$ for $(c_{ij}, M) \in cs_i$.*
  - *$m'_{ij} \in M'$ for $(c_{ij}, M') \in cs'_j$.*
  - *For any pair (c_x, m_x) that satisfies $m_x \in M_x$ for $(c_x, M_x) \in IS_v^{(0)}$ and $r(m_x) \rightarrow s(m_{ij})$, there is a message m'_x which satisfies $m'_x \in M'_x$ for $(c_x, M'_x) \in IS_v^{(1)}$ and $r(m'_x) \rightarrow s(m'_{ij})$.* ■

Theorem 2 can be proved by a decision algorithm DA. DA is outlined as follows.

The nodes send messages of problem p to node v first. Next, the nodes send messages of problem p' that satisfies $F_v(p') = true$ to node v on the links p's messages are sent. If a node receives problem p's message from v, the nodes can decide that $F_v(p) = true$. On the other hand, if the nodes receive the messages of p' from v without receiving messages of p, the nodes can decide that $F_v(p) = false$ since the communication is FIFO. The messages of p' wipe out the messages of p on the links. That is, DA executes a channel flushing [1].

This algorithm can be considered a generalization of the ICMP echo (ping) protocol of the TCP/IP protocol suite [3]. In the ICMP echo protocol, when an alive node v receives an ICMP echo request from node w, v must send an ICMP echo reply message to w. Consider in the algorithm A to solve a problem p, v must send a reply when w sends a message to v. When w sends a message on the problem p and gets no reply from v, w sends an ICMP echo request. When w receives the reply message, it can decide that v is alive, v is reachable from w, and thus v does not process the message on problem p.

There may be other types of problems in which v must communicate to other nodes before replying to w. For these types of problems, w cannot test whether v processes correctly by ICMP echo, since ICMP echo cannot test the activeness and reachability of the other nodes with which v is communicating. Thus it is impossible for w to distinguish the two cases when v does not process w's request or v does so correctly and the message communication delay between v and other nodes is large. The theorem above gives a sufficient condition that w can test v's correctness for such types of problems.

(Proof of Theorem 2) The steps in the DA are as follows.

1. (Initiation) A node w that initiates the test of v sends $("INIT", v, p, IS_v^{(1)}, IS_v^{(2)}, \ldots, IS_v^{(k)})$ to all neighboring nodes but v.
2. (Initiation forwarding) Using the leader election algorithm [7] outlined below on the network $G - \{v\}$, $V - \{v\}$ selects one node as the leader (the actual leader election algorithm is more complex in order to minimize message transmission).
   - Each node is a candidate in the initial state. It sends a "query" message with its identifier to its neighbors.
   - When a candidate receives a "query" message with an identifier larger than its own identifier, it enters a lost state and replies a "lost" message to the sender. After that, lost nodes just forward incoming messages.
   - If a candidate receives "lost" from all nodes it has sent "query" to, it sends "query" again to the other candidate nodes.
   - Lastly, one candidate receives only "query" messages of its own and it is the leader.

   During election, the message $("INIT", v, p, IS_v^{(1)}, IS_v^{(2)}, \ldots, IS_v^{(k)})$ is piggybacked on the election messages and broadcasted to $V - \{v\}$. Note that the rule for sending "lost" message is modified as will be shown later.

3. (Stopping normal execution) When a node receives "$INIT''$", it stops the initialization for solving $P(IS_v^{(1)}, IS_v^{(2)}, \ldots, IS_v^{(k)}) \cup \{p\}$. When some of these problems are being solved, the execution continues as usual.

4. (Execution of election) When a node receives "query" and it must reply "lost", the reply is deferred until the time the execution of solving problems in $P(IS_v^{(1)}, IS_v^{(2)}, \ldots, IS_v^{(k)}) \cup \{p\}$ are terminated. No deadlock results from this waiting since the termination of execution of a problem is broadcasted to the nodes (from the assumption). Let w be the elected node.

5. (Sending messages of p) Let $IS_v^{(0)} = (c_1, M_1), (c_2, M_2), \ldots, (c_n, M_n)$. Node $sr(c_i)$ sends M_i to v $(i = 1, \ldots, n)$.

6. (Sending messages of p') Let $IS_v^{(1)} = (c_1^{(1)}, M_1^{(1)}), (c_2^{(1)}, M_2^{(1)}), \ldots, (c_n^{(1)}, M_n^{(1)})$. Node $sr(c_i^{(1)})$ sends $M_i^{(1)}$ to v $(i = 1, \ldots, n)$.

7. (Output message collection and detection) Execute the following procedure for $j = 2, 3, \ldots, k-1$.
   - If a node adjacent to v receives a message m that satisfies $p(m) = p$, it broadcasts the information $F_v(p) = true$ and terminates. Otherwise, it forwards all messages received from v (except for the ones with $bg(m) = true$) to w.
   - w collects the messages forwarded by v's neighbors, obtains $OS_v^{(j-1)}$, and calculates $IS(OS_v^{(j-1)})$. $IS(OS_v^{(j-1)})$ is sent to v's neighbors.
   - Let $IS(OS_v^{(j-1)}) = (c_1^{(j)}, M_1^{(j)}), (c_2^{(j)}, M_2^{(j)}), \ldots, (c_n^{(j)}, M_n^{(j)})$. Node $sr(c_i^{(j)})$ sends $M_i^{(j)}$ to v $(i = 1, \ldots, n)$.

8. If a node adjacent to v receives a message m that satisfies $p(m) = p$, it broadcasts the information $F_v(p) = true$ and terminates. Otherwise, it forwards all received messages from v (except for the ones with $bg(m) = true$) to w.

9. If w receives all the messages in $cs \in OS_v^{(k)}$, w broadcasts the information $F_v(p) = false$ and terminates. (end of algorithm)

The validity of the algorithm is shown below.

(Case 1: $F_v(p) = true$)

Since $IS_v^{(0)}$ is the set of problem p's messages, v begins to solve p by receiving messages in $IS_v^{(0)}$, or v might begin to solve p spontaneously before receiving the messages. For the latter case, a neighboring node receives a message m that satisfies $p(m) = p$.

For the former case, assume that v sends a communication sequence $cs_i \in OS_v^{(0)}$ in response to $IS_v^{(0)}$. In that case, false detection of $F_v(p) = false$ by receiving a communication sequence $cs'_j \in OS_v^{(k)}$ cannot occur. The reason is as follows. From the condition set forth in Theorem 2, there is a pair of messages m_{ij} (included in cs_i) and m'_{ij} (included in cs'_j) received from the same link c_{ij} that satisfies the following. For any input message m_x in $IS_v^{(0)}$ sent via c_x that satisfies $r(m_x) \rightarrow s(m_{ij})$, there is a message m'_x in $IS_v^{(1)}$ sent via c_x which satisfies $r(m'_x) \rightarrow s(m'_{ij})$ (that is, for any message m_x that has a causality to m_{ij}, there is an input message that has a causality to m'_{ij} and passes the same link as that of m_x). Assume that m'_{ij} is received. In that case, a message that has a causality to m'_{ij}, m'_x, must have been received by v. On the other hand, since the communication lines are FIFO, problem p's message m_x that has sent the same line as the one for m'_x, must have been received by v before receiving m'_x. Since $F_v(p) = true$, v processed these messages correctly and sent m_{ij}. Since communication lines are FIFO, m_{ij} must have sent and received before m'_{ij}. Therefore, false detection cannot occur.

(Case 2: $F_v(p) = false$)

v correctly processes all messages in $IS_v^{(i)} (1 \leq i \leq k)$. Thus, a communication sequence $cs \in OS_v^{(k)}$ is sent to neighbors. Thus, the nodes can detect $F_v(p) = false$. ■

(Example) Consider distributed the search problem [13] and the broadcast problem [12]. In the distributed search problem, a node is the initiator and it gets information on all nodes in G by sending "search" messages and receiving their reply messages. When a non-initiator node receives a "search" message from a communication line c, it sends a "search" message to at least one communication line that is not c (the reply for this "search" message will be sent back afterwards).

In the broadcast problem, a node is the initiator and it sends information to all nodes. When a non-initiator node receives a "broadcast" message for the first time from communication line c, it sends "broadcast" messages to all communication lines but c.

Assume that the neighbor nodes of v are $u_1, \ldots, u_n$ and $c_i = (v, u_i), c'_i = (u_i, v)$ $(i = 1, \ldots, n)$.

First, consider the case where $F_v(broadcast) = true$ and deciding whether $F_v(search) = true$. Using Theorem 2, let $IS_v^{(0)} = (c'_n, "search")$,

$k = 1$, and $IS_v^{(1)} = (c'_n, \text{"}broadcast\text{"})$. Since $OS_v^{(0)} = \{\{(c_1, \text{"}search\text{"})\}, \{(c_2, \text{"}search\text{"})\}, \ldots, \{(c_{n-1}, \text{"}search\text{"})\}\}$ and $OS_v^{(1)} = \{(c_1, \text{"}broadcast\text{"}), (c_2, \text{"}broadcast\text{"}), \ldots, (c_{n-1}, \text{"}broadcast\text{"})\}$, these satisfy the condition in Theorem 2. Therefore, to decide whether $F_v(search) = true$, after the search message, a broadcast message is sent to v. If broadcast messages are received by all neighbors and no search messages are received, the nodes can decide that $F_v(search) = false$.

Next, consider the case where $F_v(search) = true$ and deciding whether $F_v(broadcast) = true$. Using Theorem 2, let $IS_v^{(0)} = (c'_n, \text{"}broadcast\text{"})$, $k = 1$, and $IS_v^{(1)} = (c'_n, \text{"}search\text{"})$. Since $OS_v^{(0)} = \{(c_1, \text{"}broadcast\text{"}), (c_2, \text{"}broadcast\text{"}), \ldots, (c_{n-1}, \text{"}broadcast\text{"})\}$ and $OS_v^{(1)} = \{\{(c_1, \text{"}search\text{"})\}, \{(c_2, \text{"}search\text{"})\}, \ldots, \{(c_{n-1}, \text{"}search\text{"})\}\}$, these satisfy the condition in Theorem 2. Therefore, to decide whether $F_v(broadcast) = true$, after the broadcast message, send a search message to v. If a search message is received by a neighbor before receiving a broadcast message, the nodes can decide that $F_v(broadcast) = false$. ■

The messages of the broadcast problem can be used to detect the truant failure for the search problem. The messages of the search problem can be used to detect the truant failure for the broadcast problem.

4 Conclusion

This paper has presented a multi-policy distributed system as a new model of autonomous decentralized systems. We introduced a new type of failure on the multi-policy distributed systems, truant failure, and presented an algorithm for truant failure detection.

Algorithms for detecting other undesirable local policies and for eliminating the effects of such policies are subjects for further study.

Acknowledgments

The authors would like to thank Dr. Rikio Onai for his encouragement and suggestions.

References

[1] Ahuja, M.: "An Implementation of F-channels," IEEE Trans. on Parallel and Distributed Systems, Vol. 2, No. 6, pp. 658–667 (June 1993).

[2] Clark, D.: "Policy Routing in Internetworks," Internetworking: Research and Experience, Vol. 1, pp. 35–52 (1990).

[3] Comer, D.: "Internetworking with TCP/IP," Prentice-hall (1988).

[4] Cristian, F., Aghill, H., Strong, R., and Dolev, D.: "Atomic Broadcast: From Simple Message Diffusion to Byzantine Agreement," FTCS (1985).

[5] Fisher, M. J., Lynch, N. A., and Paterson, M. S.: "Impossibility of Distributed Consensus with One Faulty Process," J. of ACM, Vol. 32, No. 2, pp. 374–382 (Apr. 1985).

[6] Fisher, M. J.: "The Consensus Problem in Unreliable Distributed Systems (A Brief Survey)," Proc. Int. Foundation of Computation Theory Conf. pp. 128–140 (Aug. 1983).

[7] Gallager, R. G., Humblet, P. A., and Spira, P. M.: "A Distributed Algorithm for Minimum-Weight Spanning Trees," ACM TOPLAS, Vol. 5, No. 1, pp. 66–77 (Jan. 1983).

[8] Garcia-Molina, H.: "Elections in distributed computing systems," IEEE Transactions on Computers C-31, 1, pp. 48–59(Jan. 1982).

[9] Lamport, L.: "Time, Clocks, and the Ordering of Events in a Distributed System," Comm. ACM, Vol. 21, No. 7, pp. 558–565 (July 1978).

[10] Lougheed, K. and Rekhter, Y.: "A Border Gateway Protocol 3 (BGP-3)," RFC 1267 (Oct. 1991).

[11] Lynch, D. C. and Rose, M. T.: "Internet System Handbook," Addison-Wesley (1993).

[12] Segall, A.: "Distributed Network Protocols," IEEE Trans. on Information Theory, Vol. 29, No. 1 (Jan. 1983).

[13] Sharma, M. B., Iyengar, S. S., and Mandyam, N. K.: "An Efficient Distributed Depth-first-search Algorithm," Information Processing Letters, Vol. 32, No. 4, pp. 183–186 (Sep. 1989).

[14] Tel, G.: "Topics in Distributed Algorithms," Cambridge University Press (1991).