ELSEVIER

Information and Software Technology 41 (1999) 745–754

INFORMATION AND SOFTWARE TECHNOLOGY

A protocol synthesis method for fault-tolerant multipath routing

K. Ishida[a,*], Y. Kakuda[a], M. Nakamura[b], T. Kikuno[b], K. Amano[a]

[a]Faculty of Information Sciences, Hiroshima City University, Asaminami-ku, Hiroshima, 731-3194, Japan
[b]Graduate School of Engineering Science, Osaka University, Toyonaka-shi, Osaka, 560-8531, Japan

Abstract

This paper proposes a new synthesis method for generating fault-tolerant multipath routing protocols. The protocol is defined as fault-tolerant if messages can be rerouted by using another path when a communication channel fails. The routing protocols obtained adopt a multipath routing function, augmented with routing table, where each table stores the next nodes for multipath routing, and updates the tables according to the network topology changes. Additionally, the routing protocol can attain flexibility by the multipath routing mechanism in the sense that only a small amount of change is needed for the change of network topology. We also briefly describe an extension of the proposed method for generating multicast routing protocols. © 1999 Elsevier Science B.V. All rights reserved.

Keywords: Protocol engineering; Protocol synthesis; Multipath routing; Fault-tolerant routing; Flexible routing

1. Introduction

Routing in a packet-switched network is to deliver packets through communication paths from a source node to a destination node. The multipath routing is more robust than single path routing because as long as at least one of the multiple paths between source and destination nodes is viable the messages will be delivered. Multipath routing is thus one of the most promising ways to realize the reliable routing services [1,2].

The most fundamental requirement for multipath routing protocols is considered as follows [3]:

Requirement 1
Fundamental capability for multipath routing. As messages are delivered through multiple communication paths, protocol specification for the message delivery must be specified for any node on the communication paths. Next, fault-tolerance and flexibility become important characteristics to ensure quality of communication services. Therefore, the protocol must also satisfy the following two hard requirements.
Requirement 2
Fault-tolerance for a communication channel failure. In order to definitely deliver messages from source node to destination node even when a communication channel fails, the source node must possess a recovery function of rerouting.

Requirement 3
Flexibility for network topology changes. When some nodes and channels are newly added or deleted on the network, modification of the protocol specification must be easily done.

Design of practical multipath routing protocols is complex and difficult due to the complicated requirements listed above. For such a difficult and complex protocol design, the protocol synthesis [4,5] is recognized as one of the most prominent solutions, which automatically derives the protocol specifications without specification errors. In this paper, a synthesis of multipath routing protocols is defined as the generation of a routing protocol specification from a routing service specification, both of which are modeled by Finite State Machines (FSMs). So far, various protocol synthesis methods have been proposed [4–7]. However, none of them were for routing protocols with recovery function of rerouting, although the previous methods generate recovery functions such as retransmission for message loss, check pointing and roll-back recovery for coordination loss [8,9].

This paper proposes a new synthesis method for complex multipath routing protocols, which satisfy Requirements 1, 2 and 3. The proposed method generates multipath routing augmented with routing table. Each table stores the candidates of the next nodes. The table is utilized for determining the next node to which messages are transmitted along a communication path. The synthesized protocol specification has a rerouting function, such that the messages can be rerouted through one of the multiple paths by referring the table. Moreover, the protocol specification can be modified

* Corresponding author.

PII: S0950-5849(99)00034-8

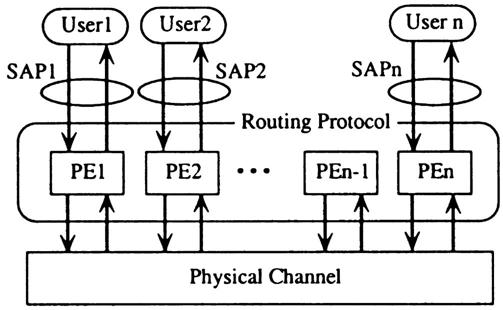

Fig. 1. Communication model.

easily by updating the table, even if network topology changes.

The rest of this paper is organized as follows: Section 2 gives fundamental definitions concerning protocol synthesis. In Section 3, we define synthesis problem for multipath routing and propose a solution to the problem. Then, we prove the correctness of the method in Section 4, and apply our method to a typical example in Section 5. Flexibility for topology change and an extension for multicast routing are discussed in Section 6. Finally, Section 7 concludes this paper.

2. System model

2.1. Communication model

As shown in Fig. 1, a communication service is specified by service primitives exchanged between users in the higher layer, and nodes in the lower layer through service access points (SAPs). A routing protocol can be viewed as a black box from the users' view point. The nodes are also called protocol entities, which are denoted by *PE*s in the following. As the users correspond to protocol entities in the higher layer, it is assumed that one user uses one *PE*. In a routing protocol, each *PE* must deliver a message through existing physical channels.

2.2. Topology graph

Definition 1. A topology graph is defined as an undirected graph $G = (V, E)$, where V represents a set of *PE*s, and $E(\subseteq V \times V)$ represents a set of communication channels (FIFO queues).

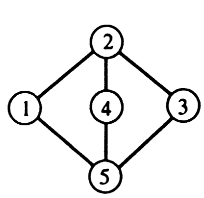

Fig. 2. A topology graph.

For any two nodes, $PE_u, PE_v \in V$ on a topology graph $G = (V, E)$, if there exists an edge $(u,v) \in E$, then node PE_u is called an adjacent node of PE_v. Fig. 2 shows a topology graph. This paper imposes the following restriction to assure the connectivity of the communication path between any pair of *PE*s even if a communication channel fails.

Restriction 1. There are at least two edge-disjoint undirected paths between any two *PE*s in the topology graph.

From Restriction 1, for any pair of nodes $PE_i, PE_j \in V$, a path ρ between PE_i and PE_j must exist. Intuitively, the path can be interpreted as a communication path from protocol entities PE_i to PE_j. Let us consider a case that $user_i$ communicates with $user_j$ via the path. At first, $user_i$ sends the service primitive p to PE_i. Next, PE_i sends a message e to PE_j via the path. Then, PE_j receives e and sends the service primitive q to $user_j$. For this, we call PE_i and PE_j, *S-node* and *D-node* of the path, respectively. For the path, the intermediate nodes between PE_i and PE_j are called *R-nodes* on ρ. Messages are delivered from the *S-node* to the *D-node* via *R-nodes* on the communication path. As a special case, if PE_i is an adjacent node of PE_j and $\rho = (PE_i, PE_j)$, then ρ does not have an *R-node*.

2.3. Service specification

A service specification defines an execution order of service primitives that are exchanged between users and *PE*s through service access points. A service access point (SAP), between $user_i$ and PE_i, is denoted by SAP_i.

Definition 2. A service specification is modeled by an FSM, $S = \langle S_s, \Sigma_s, T_s, \sigma\rangle$, where

- S_s is a non-empty finite set of service states.
- Σ_s is a finite set of service primitives. Each service primitive $p \in \Sigma_s \cup \{\varepsilon\}$ has, as an attribute, an index of SAP through which p passes, and ε is a null primitive that causes no message exchange. When p passes through SAP_i, we define a function $sap(p) = i$, and the primitive is denoted by p_i.
- T_s: $S_s \times \Sigma_s \rightarrow S_s$ is a partial transition function. For simplicity, we use T_s also to represent a set of triplets (u,p,v), such that $v = T_s(u,p)(u,v \in \Sigma_s)$.
- $\sigma \in S_s$ is an initial service state.

A state $u \in S_s$ is called a final state iff there is no outgoing transition (u,p,v) for any p and v. If more than one transition is outgoing from a service state, one such transition is chosen and executed. We call this FSM a service specification S-SPEC. An S-SPEC is represented by a labeled directed graph. For any state which represents a service state $s \in S_s$ in S, we define $OUT(S) = \{i | i = sap(p)\}$, where p is a label attached to an outgoing transition from s.

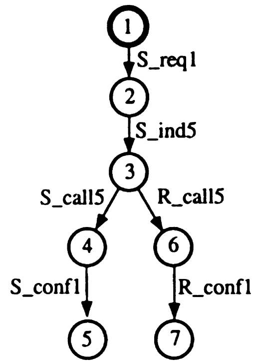

Fig. 3. An example of service specification S-SPEC.

Example 1. An example of the S-SPEC is shown in Fig. 3. In this figure, a circle denotes a service state, and an arrow denotes a transition between states. The state drawn by a bold circle is an initial state. This service specification represents sequences of message delivery from the source node to the destination node and its positive or negative acknowledgement from the destination node to the source node. For example, after $user_1$ sends *S_req1* to PE_1 through SAP_1, $user_5$ receives *S-ind*5 from PE_5 through SAP_5 in this order. In $user_5$ sends *S_call*5 (ACK) through SAP_5, $user_1$ receives *S_conf*1 from PE_1 through SAP_1. In case $user_5$ sends *R_call*5 (NACK) through SAP_5, $user_1$ receives *S_conf*1 from PE_1 through SAP_1.

This paper additionally imposes the following restrictions to assure the correctness of the proposed protocol synthesis method.

Restriction 2. Consider an S-SPEC and any transition sequence (u_1, p_1, u_2), $(u_2, p_2, u_3)\ldots(u_k, p_k, u_{k+1})$ in the S-SPEC, where u_1 is the initial state and u_{k+1} is a final state in the S-SPEC. There exists an execution order of service primitives $p_1, p_2, \ldots, p_k$, such that service primitive $p_i(i \le k)$ must be executed before service primitives $p_{i+1}, p_{i+2}, \ldots, p_k$ for any i.

Restriction 3. In S-SPEC, for any three states u, v, and $w(v \ne w)$, T_s does not include two transitions (u,p,v), (u,p',w) with $sap(p) \ne sap(p')$ and $p \ne p'$.

This restriction implies that service primitives p and p' are not simultaneously exchanged through different SAPs.

2.4. Protocol specification

The protocol specification consists of n-tuples of specifications for PEs. In order to realize loop-free transmission, we assume that after a message is received from the adjacent PE_i, it cannot be transmitted to the same PE_i. Transmission and reception of messages between adjacent nodes are defined as follows:

Definition 3. If a message e is transmitted to PE_j, then it is denoted by a transmission event $!e(j)$. On the contrary, if a message e is received by PE_j, then it is denoted by a reception event $?e(j)$. If the message e is transmitted to (or received by) one of PE_{j1}, $PE_{j2}, \ldots, PE_{jk}$, it is denoted by a transmission event $!e(j_1, j_2, \ldots, j_k)$ (or $?e(j_1, j_2, \ldots, j_k)$), respectively.

We introduce a set of nodes called routing table t-set for each node PE_i. The t-set is used for determining the adjacent node to which the PE_i transmits or receives messages along a communication path. The transmission event $!e(t\text{-}set)$, where t-set $= \{j_1, j_2, \ldots, j_k\}$ implies that message e is transmitted to one of the adjacent nodes PE_{j1}, $PE_{j2}, \ldots, PE_{jk}$. A reception event $?e(j_1, j_2, \ldots, j_k)$ is also denoted by $?e$(t-set) with the t-set. It is assumed that the adjacent nodes are determined by the *S-node* of message m and the identification number of the communication path on which message m is delivered. This is the so-called source-based routing.

Definition 4. A PE specification is modeled by an FSM $P_i = \langle S_{ip},\ \Sigma_{ip},\ T_{ip},\ \sigma_{ip} \rangle$, where

- S_{ip} is a non-empty finite set of protocol states.
- Σ_{ip} is a non-empty finite set of protocol events. $\Sigma_{ip} = \{p | p \in \Sigma_s, sap(p) = i\} \cup \mathrm{MEX}_i \cup \{\mathrm{T.O.}\} \cup \{\varepsilon\}$, where Σ_s is a set of primitives in Definition 2, and MEX_i is a set of events which are transmitted to PE_{i1}, $PE_{i2}, \ldots, PE_{ik}$ or received by $PE_{i1}, PE_{i2}, \ldots, PE_{il}$ and T.O. is a timeout event that occurs when a predetermined time elapses. ε is null primitive that causes no message exchanging.
- $T_{ip} : S_{ip} \times \Sigma_{ip} \to S_{ip}$ is a partial transition function.
- $\sigma \in S_{ip}$ is an initial protocol state.

We call this FSM a PE-SPEC*i*. As with the service specification, a PE specification is also represented by a labeled directed graph. We explain a timeout transition $(u,\mathrm{T.O.},v)$ in T_{ip}. At the time when the state of PE-SPEC*i* moves the state u, counting time starts. Only when a current state of PE-SPEC*i* is state u and the predetermined time elapsed, the state of PE-SPEC*i* moves the state v. A state $u \in S_{ip}$ is called a final state iff there is no outgoing transition (u,p,v) for any p and v. A state $u \in S_{ip}$ is called a receiving state for e iff any (outgoing or incoming) transition from u is a reception event (u,e,v) for any e and v. A transition "p/q" with p, $q \in T_p$ denotes a successive execution of transitions p and q.

Example 2. Fig. 4 shows an example of PE-SPEC$_1$ for PE_1 in Fig. 2. In this figure, a circle denotes a protocol state, and an arrow between states denotes a transition.

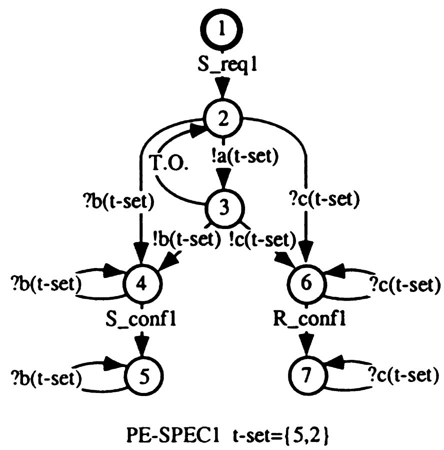

Fig. 4. An example of protocol entity specification.

For example, !a(t-set), where t-set = {5,2} implies two possible message transmissions !a(5) and !a(2).

The following definition requires that messages are exchanged through existing channels.

Definition 5. Consider a topology graph $G = (V,E)$ and a PE specification $P_i = \langle S_{ip}, \Sigma_{ip}, T_{ip}, \sigma_{ip} \rangle$. Transitions $(u,!e(j),v)$ and $(u,?e(j),v)$ in T_{ip} obey channel restriction if the following conditions are satisfied, respectively.

If $(i,j) \notin E'$, then $(u,!e(j),v) \notin T_{ip}$ for any u, v, e, and if $(i,j) \notin E'$, then $(u,?e(j),v) \notin T_{ip}$ for any u, v, e.
If all transitions in T_{ip} obey channel restriction, we say P_i obeys channel restriction. And if all P_is obey channel restriction, we say P obeys channel restriction.

3. Synthesis method for fault-tolerant multipath routing

3.1. Protocol synthesis problem

Protocol synthesis problem for fault-tolerant multipath routing to be solved in this paper is formally defined as follows:

Problem FM. *Input:* A topology graph G with Restriction 1, and a service specification S with Restrictions 2 and 3. *Output:* A multipath routing protocol specification P which satisfies the following Conditions. *Condition 1:* Unspecified receptions never occur in P. *Condition 2:* Even if a message loss occurs, the execution order of service primitives defined by S is kept in P. *Condition 3:* P obeys the channel restriction.

No existence of unspecified receptions in Condition 1, and keeping execution order of service primitives in Condition 2 are ordinary conditions for protocol synthesis. Meanwhile, the channel restriction in Condition 3 and discussions on the failure of communication channel in Condition 2 are unique to our discussion. Requirements 1–3 in Section 1 are taken into consideration as above Conditions 1–3.

3.2. Outline of the synthesis method

For Problem FM, the proposed method to derive a protocol specification from a given service specification consists of the following four steps:

Step 1 Obtain n projected service specifications by applying the projection to the given service specification S-SPEC. In these projected service specifications, the service primitive associated with SAP_i is represented by PS-SPECi, which is obtained from S-SPECs by substituting each transition not associated with SAP_i by ε.

Step 2 Construct PE-SPECi by applying the transition synthesis rules shown in Fig. 5 to PS-SPECi obtained in Step 1. In Fig. 5, OUT is specified in Section 2.3, and E_i denotes a service primitive in the PS-SPEC$_i$. Each pair of transition synthesis rules Ak and Bk (k = 1,2) is applied to each pair of transitions (S_1,E_i,S_2) in PS-SPEC$_i$ and (S_1,ε,S_2) in PS-SPEC$_j$ ($i \neq j$), respectively. Message e is

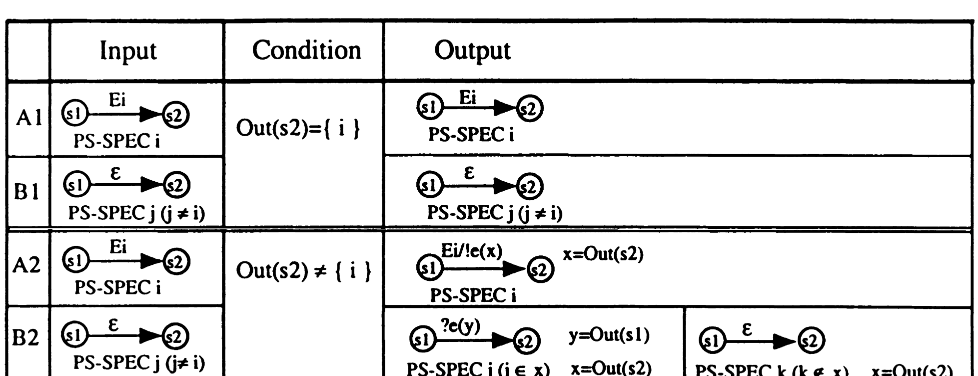

Fig. 5. Transition synthesis rules.

uniquely generated for each service primitive E_i in the rules A2 and B2.

Step 3 Incorporate the capability of multipath routing into the PE specifications PE-SPEC*i* constructed in Step 2, such that the resultant specification obeys channel restriction. Next, remove ε transitions from PE-SPEC*i* by the algorithm presented in Ref. [10].

Step 4 Incorporate the recovery function of rerouting into the protocol specification constructed in Step 3 using timeout event.

3.3. Detail of proposed synthesis method

As Steps 1 and 2 are almost the same as those in Ref. [4], we will explain only details of Steps 3 and 4.

3.3.1. Step 3

Step 3 incorporates the capability of multipath routing into n PE specifications PE-SPECs that obey the channel restriction. This incorporation is executed by applying the following TE procedure. Note that the protocol specification obtained in Step 2 possesses the same graph structures as the service specification, because the transition synthesis rule adds (removes) neither states nor transitions, respectively. That is, each PE-SPEC*i* has the same number of states and transitions. Hence, for a transition (u,E_i,v) in S-SPEC, there exists n transitions such that $(u,E_i/!e(j),v) \in T_{ip}$, $(u,?e(i),v) \in T_{jp}$, and $n-2$ transitions $(u,\varepsilon,v) \in T_{kp}$ $(k \neq i,j)$. TE procedure is applied to such n transitions.

TE procedure

Input PE-SPECs obtained in Step 2, and topology graph $G = (V,E)$.

Output PE-SPECs with the capability of multipath routing that obeys channel restriction.

Procedure

For each n transition $(u,E_i/!e(j),v) \in T_{ip}, (u,?e(i),v) \in T_{jp}$, and $n-2$ transitions $(u,\varepsilon,v) \in T_{kp}(k \neq i,j, 1 \leq k \leq n)$, execute TE-Step 1 through TE-Step 4. Then, remove all ε transitions using the algorithm presented in Ref. [10].

TE-Step 1

Search all loop-free paths $\rho_1,\rho_2,\ldots,\rho_m$ from PE_i to PE_j based on G.

TE-Step 2

If $(i,j) \notin E$, remove transitions $(u,E_i/e(j),v)$ and $(u,?e(i),v)$ from T_{ip} and T_{jp}, respectively.

TE-Step 3

For each $\rho \in \{\rho_1,\rho_2,\ldots,\rho_m\} - \{\rho_d\}$, where ρ_d is a path from PE_i to PE_j having no *R-nodes*, execute TE-Substeps 3.1 and 3.2.

TE-Substep 3.1

For each $T_{kp}(k \neq i,j)$, such that PE_k is an *R-node* on ρ, remove (u,ε,v) from T_{kp}.

TE-Substep 3.2

Add several transitions for each PE based on ρ as follows:

1. For an adjacent node PE_x of PE_i on ρ (PE_i is a *S-node* on ρ), add a transition $(u,E_i/?e(x),v)$ to T_{ip}.
2. For an adjacent node PE_y of PE_j on ρ (PE_j is a *D-node* on ρ), add a transition $(u,?e(y),v)$ to T_{jp}.
3. For each T_{kp} $(k \neq i,j)$ such that PE_k is an *R-node* on ρ, and PE_z and PE_w are a pair of adjacent nodes of PE_k, add a transition $(u,?e(z)/!e(w),u)$ to T_{kp}, where PE_z is on the sub-path of ρ from PE_i to PE_k, and PE_w is on the sub-path of ρ from PE_k to PE_j.

TE-Step 4 Introduce t-set into PE-SPEC as follows:

1. In the T_{ip}, remove transitions $(u,E_i/!e(x_1),v),\ldots,(u,E_i/!e(x_m),v)$. Next, create a new state u' in S_{ip}, then add two transitions (u,E_i,u') and $(u',!e(t\text{-}set),v)$. Finally, let t-set $= \{x_1,\ldots,x_m\}$.
2. In the T_{jp}, remove transitions $(u,?e(y_1),v),\ldots,(u,?e(y_m),v)$ and add transition $(u,?e(t\text{-}set),v)$. Next, let t-set $= \{y_1,\ldots,y_m\}$.
3. In the T_{kp} such that P_{Ek} is an R-node on a path $\rho \in \{\rho_1,\ldots,\rho_m\}$, remove all transitions $(u,?e(z_1)/!e(w_1),u),\ldots,(u,?e(z_{m'})/!e(w_{m'}),u)$ and add transitions $(u,?e(t\text{-}set1)/!e(t\text{-}set2),v)$. Then, let $t\text{-}set1 = \{z_1,\ldots,z_{m'}\}$ and $t\text{-}set2 = \{w_1,\ldots,w_{m'}\}$. Here, $1 \leq m' \leq m$.

At TE-Step 1, loop-free communication paths from the *S-node* to the *D-node* for the message are searched as much as possible by the conventional path enumeration method [11]. At TE-Step 2, transitions that violate channel restriction are removed. If ρ_d exists in G, it is clear that the transitions $(u,E_i/!e(j),v) \in T_{ip}$ and $(u,?e(i),v) \in T_{jp}$ obey the channel restriction. It is not necessary to execute TE-Step 2 and TE-Substeps 3.1 and 3.2 for ρ_d. In TE-Step 4, modification of the *S-node* is done to avoid transmission of E_i more than once.

3.3.2. Step 4

In this step, we incorporate the function of rerouting into PE specifications PE-SPECs obtained in Step 3. When a communication channel fails, a PE at the source node finds the failure by the timeout event, and retransmits messages for rerouting.

When a channel failure occurs on a path and a transmitted message is lost, reception events are not executable in *PE*s on the path. This is because the source node waits continuously for receiving an acknowledgement of the transmitted message. Hence, we add transitions for retransmission of the message to the source node. However, as a side effect, unspecified receptions may occur. Therefore, we add supplementary transitions so that unspecified receptions are avoided.

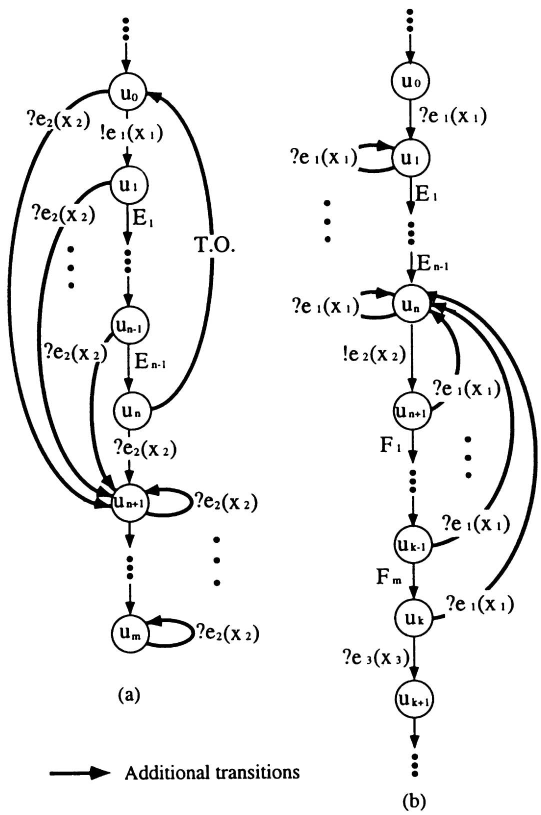

Fig. 6. Addition of transitions in Step 4.

The procedure of Step 4 consists of the following steps S1, S2 and S3: For each transition sequence from an initial state u_{init} to a final state in each PE-SPEC, we apply steps S1–S3 in this order.

S1: First, we search a transition (u_0,E,u_1) such that E is either a transmission event or a reception event and all transitions from u_{init} to u_0 are service primitives.
S2: Here, we consider two cases: $E = !e_1(x_1)$ and $E = ?e_1(x_1)$.

(Case of $E = !e_1(x_1)$)
Assume that the event $(u_n, ?e_2(x_2), u_{n+1})$ is the first reception event in the transition sequence from u_0. Then, we add the transitions (1), (2) and (3) to PE-SPECi as shown in Fig. 6(a).

1. (u_n,T.O.,u_0).
2. $(u_0, ?e_2(x_2), u_{n+1})$, $(u_1, ?e_2(x_2), u_{n+1}), \ldots,$ $(u_{n-1}, ?e_2(x_2), u_{n+1})$.
3. (v,$?e_2(x_2)$,v) for each state v included in the transition sequence from u_n to the final state u_m.

After this, we regard u_{n+1} as u_{init}.
(Case of $E = ?e_1(x_1)$)
Assume that the event (u_k,$?e_3(x_3)$,u_{k+1}) is the first reception event in the execution sequence from u_1. Then, we perform the following (see Fig. 6(b)).
Assume that the event $(u_n, !e_2(x_2), u_{n+1})$ is the first transmission event in the transition sequence from state u_1 to u_k. (If the transition is not found, we regard u_n as u_k.)
Next, we add the following transitions (1) and (2) to PE-SPECi. (If $u_n = u_k$, we only add a transition (a).)

1. $(u_1, ?e_1(x_1), u_1), (u_2, ?e_1(x_1), u_2), \ldots, (u_n, ?e_1(x_1), u_n)$.
2. $(u_{n+1}, ?e_1(x_1), u_n), (u_{n+2}, ?e_1(x_1), u_n), \ldots, (u_k, ?e_1(x_1), u_n)$.

After this, we regard u_k as u_{init}.

S3: We recursively execute S1 and S2 from the new u_{init}.

4. Correctness of the synthesis method

In this section, we discuss the correctness of the proposed method. For this purpose, we must prove that the obtained protocol specification satisfies Requirements 1 and 2. However, correctness for Requirement 1 can be proved easily because the method is almost the same as the previous method [4]. Then, we omit the proof for Requirement 1 in

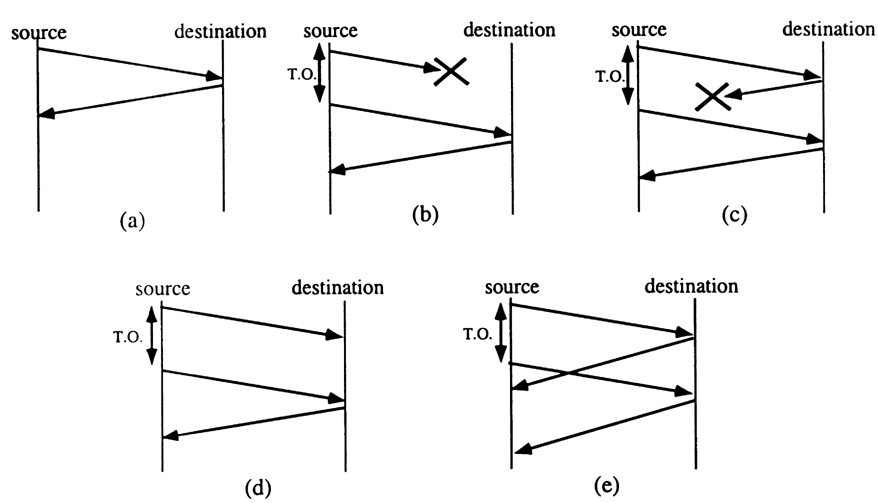

Fig. 7. Sequences for five cases.

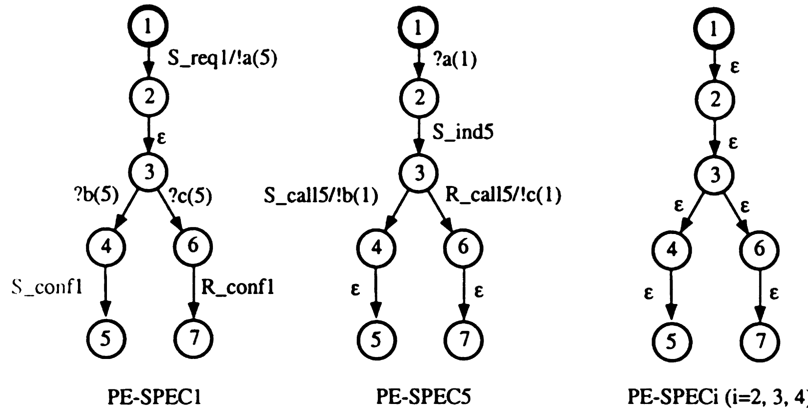

Fig. 8. Protocol specification PE-SPEC*i* after Step 2.

this paper. For the proof for Requirement 2, we must show that the obtained protocol specification satisfies Conditions 1, 2 and 3, even when additional transitions in Step 4 are executed. As it is obvious that protocol specification satisfies Condition 3, we discuss Conditions 1 and 2 here.

Cases shown in Fig. 6(a) and (b) are considered for the source node and destination node, respectively. The timeout events in the source node can occur when a predetermined time elapses. If a message from the source node to the destination node is lost or an acknowledgement message from the destination node is lost, the timeout event can surely occur and the message retransmitted. If the predetermined time for timeout is smaller than the sum of the time for delivery of the message and the time for delivery of the acknowledgement message, then the timeout event possibly occurs and unnecessary retransmission may be executed.

Depending on the loss of the message and its acknowledgement message and on the inappropriate predetermined time for the timeout, all possible cases are divided into the following five cases as shown in Fig. 7:

Case 1 (see Fig. 7(a))
The message from the source node was delivered to the destination node, and the acknowledgement message was also delivered without message loss. Thus, both the transitions from u_0 to u_{n+1} through $E_1,\dots,E_{n-1}$ in the source node (shown in Fig. 6(a)) and the transitions from u_0 to u_k through $E_1,\dots,E_{n-1}$, $F_1,\dots,F_m$ in the destination node (shown in Fig. 6(b)) are executed. This case is included in the proof for Requirement 1.
Case 2 (see Fig. 7(b))
After the timeout event occurs due to the loss of the message from the source node, the message is retransmitted. As shown in Fig. 6(a), in the source node after the transitions from u_0 to u_n through $E_1,\dots,E_{n-1}$ are executed, the timeout event occurs and the transitions from u_0 to u_{n+1} through $E_1,\dots,E_{n-1}$ are executed. That is, message e_1 is retransmitted in the source node. Execution of transitions in the destination node is the same as that in Case 1.
Case 3 (see Fig. 7(c))
After the timeout event occurs due to the loss of the acknowledgement message from the destination node, the message is retransmitted from the source node. As shown in Fig. 6(a), the execution of transitions in the source node is the same as that in Case 2. On the contrary, as shown in Fig. 6(b), in the destination node after transitions from u_0 to u_k through $E_1,\dots,E_{n-1}$, $F_1,\dots,F_m$ are executed, the event $?e_1(x_1)$ occurs and then the transitions from u_n to u_k through $E_1,\dots,E_{n-1}$, $F_1,\dots,F_m$ are executed.
Case 4 (see Fig. 7(d))
There is no loss of message and its acknowledgement message. However, due to inappropriate predetermined time for the timeout event, the message is retransmitted. After the transmitted and retransmitted messages are received, the acknowledgement message is transmitted. In Fig. 6, transitions for avoiding unspecified reception are added for the source and destination nodes.
In the source node (as shown in Fig. 6(a)), after transitions from u_0 to u_n through $E_1,\dots,E_{n-1}$ are executed, the timeout event and the reception event $?e_2(x_2)$ occur in this order. Message e_2 can be received at a state between u_0 and u_n. In the destination node (as shown in Fig. 6(b)), after message e_1 is received by executing the reception event $?e_1(x_1)$, the same message e_1 is received at a state between u_0 and u_n. Then, transitions from u_n to u_k are executed through $F_1,\dots,F_{\mathrm{m}}$.
Case 5 (see Fig. 7(e))
The situation is the same as that in Case 4 with respect to no loss of the message and its acknowledgement, and inappropriate predetermined time for the timeout event. However, the acknowledgement message is transmitted twice from the destination node, as the acknowledgement is transmitted before the retransmitted message is received. In Fig. 6, transitions for avoiding unspecified messages are also added for the source and destination nodes.
In the source node (as shown in Fig. 6(a)), the reception event $?e_2(x_2)$ is executed at a state between u_{n+1} and u_m after execution of transitions mentioned in Case 4. In the destination node (as shown in Fig. 6(b)), after transitions

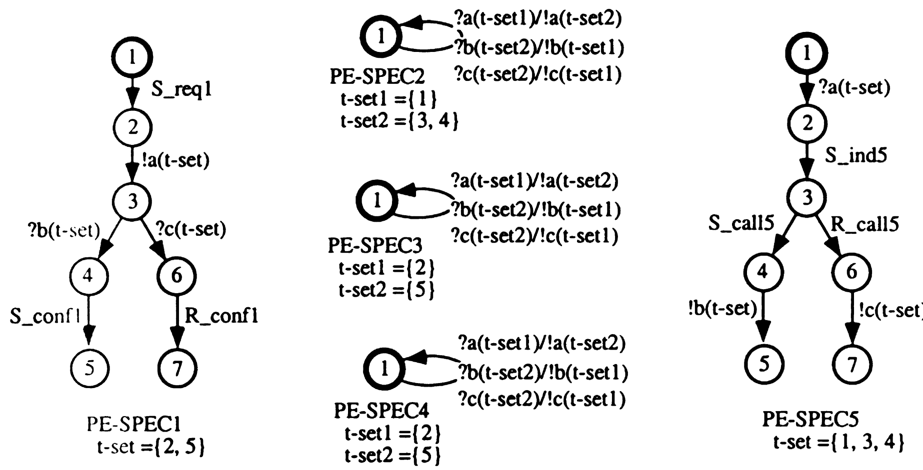

Fig. 9. Protocol specification PE-SPEC*i* after Step 3.

from u_0 to u_{n+1} are executed, the reception event $?e_1(x_1)$ occurs at a state between u_n and u_k. After this, transitions from u_n to u_{k+1} are executed through $F_1,\ldots,F_{n-1}$.

In Case 1, the execution order of primitives is executed in such an order that primitives in $F_1,\ldots,F_m$ are executed after all primitives in $E_1,\ldots,E_{n-1}$ are executed. This is required in the given service specification. Although redundant primitives are executed in Cases 2–5, the above execution order of primitives in Case 1 is also kept in Cases 2–5.

Based on the above observation, we can say that Conditions 1 and 2 are satisfied in the synthesized protocol specification.

5. Example

5.1. Explanation of the synthesis method by example

We apply our synthesis method to a typical example. Consider a service specification S-SPECs shown in Fig. 3 and a topology graph G shown in Fig. 2.

At Step 1, service primitives are projected to PS-SPEC1, PS-SPEC2, PS-SPEC3, PS-SPEC4, and PS-SPEC5. At Step 2, PE specifications PE-SPECs are obtained from PS-SPECs. The synthesis rules satisfy Condition 1. For example, consider transition (1,S_req1, 2) in PS-SPEC1 and transition $(1,\varepsilon,2)$ in PS-SPEC*i* ($i = 2, 3, 4, 5$). For this case, as $OUT(2) = \{5\} \neq \{1\}$ (see Fig. 3), the transition synthesis rule A2 and B2 are applied (see Fig. 5). As a result, two transitions (1,S_req1, 2) in PS-SPEC1 and $(1,\varepsilon,2)$ in PS-SPEC5 are changed to (1,S_req1/!a(5),2) and (1,?a(1),2), respectively. However, $(1,\varepsilon,2)$ in PS-SPEC*i* ($i = 2, 3, 4$) remains unchanged as $\{2,3,4\} \neq OUT(2)$ for PS-SPEC*i* ($i = 2, 3, 4$). Fig. 8 shows the result of Step 2.

Step 3 constructs PE-SPECs with multipath transmission that obey the channel restriction. For example, consider transmission (1,S_req1/!a(5),2) in PE-SPEC1, (1,?a(1),2) in PE-SPEC5, $(1,\varepsilon,2)$ in PS-SPEC*i* ($i = 2, 3, 4$) in Fig. 8. There are three paths from node 1 (PE_1) to node 5 (PE_5) in G (see Fig. 2), ρ_1: $PE_1 \rightarrow PE_5$, ρ_2: $PE_1 \rightarrow PE_2, \rightarrow PE_3, \rightarrow PE_5$ ρ_3: $PE_1 \rightarrow PE_2, \rightarrow PE_4, \rightarrow PE_5$. Next, with respect to ρ_1, transitions (1,S_req1/!a(5),2), (1,?a(1),2) are unchanged in PS-SPEC*i* ($i = 1, 5$), as ρ_1 has no *R-node*. For ρ_2, transitions (1,S_req1/!a(2),2), (1,?a(1)/!a(3),1), (1,?a(2)/!a(5),1), (1,?a(3),2) are added to PS-SPEC*i* ($i = 1, 2, 3, 5$) by TE-Substep 3.2, the transitions in PS-SPEC_1 are modified to transitions (1,S_req1,1′), respectively. Similarly, for ρ_3, transitions (1,S_req1/!a(2),2), (1,?a(1)/!a(4),1), (1,?a(2)/!a(5),1), (1,?a(4),2) are added to PS-SPEC*i* ($i = 1, 2, 4, 5$) by TE-Substep 3.2, the transitions in PS-SPEC1 are modified to transitions (1,S_req1,1′), respectively. Then, the transitions in PE-SPEC1 are modified to (1′,!a(t-set),2), where t-set = {2,5}. Similarly, the transitions in PE-SPEC5 are modified to (1,?a(t-set),2), where t-set = {1,3,4}. And the transitions in PE-SPEC2 are modified to (1,?a(t-set1)/!a(t-set2),1), where t-set1 = {1} and t-set2 = {3,4}. Other transitions in PE-SPEC3 and PE-SPEC4 are similarly modified, and all ε transitions are removed. Fig. 9 shows a protocol specification PE-SPECs after Step 3.

In Step 4, timeout events and some other transitions are added. For example, consider execution sequence S_req1, !a(t-set), ?b(t-set), S_conf1 in PE-SPEC1. As transition (3,?b(t-set),4) exists after transition (2,!a(t-set),3) on the transition sequence, four transitions (3,T.O.,2), (2,?b(t-set),4), (4,?b(t-set),4), (5,?b(t-set),5) are added in PE-SPEC1. Fig. 10 shows the final protocol specification after Step 4, where each number in circle is renumbered.

5.2. Fault-tolerance of synthesized routing protocol

In this subsection, we discuss whether the protocol specification obtained by our method realizes both Requirements 1 and 2 or not. Consider the protocol specification shown in

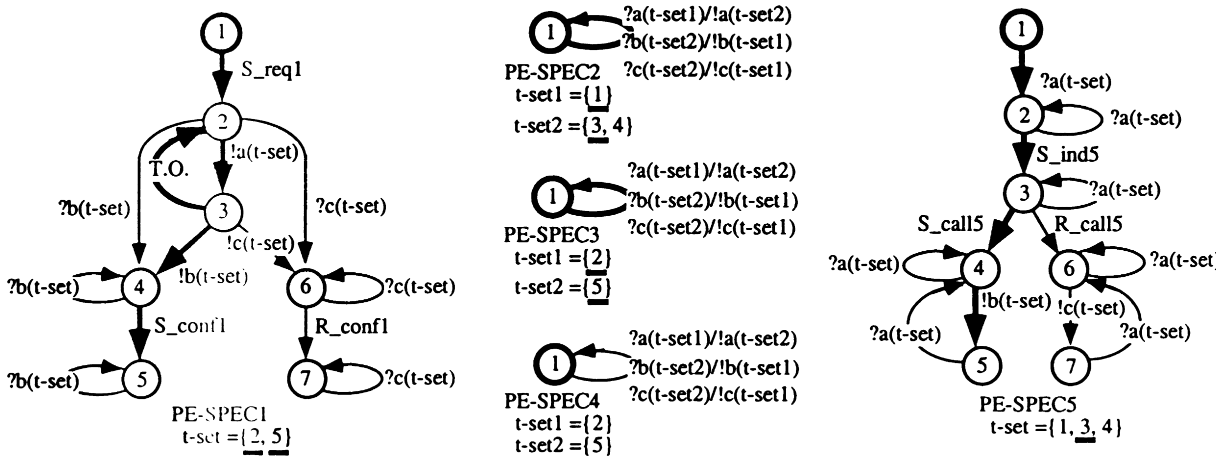

Fig. 10. Final protocol specification after Step 4.

Fig. 10, and assume that the channel between PE_1 and PE_5 in Fig. 2 fails.

First PE_1 delivers directly message *a* through the channel between PE_1 and PE_5. However, this message is lost because of the channel failure. Then PE_1 can know that the message is lost by the timeout event. Next, the PE_1 retransmits the message via another path. Let us consider the following path: $PE_1 \rightarrow PE_2 \rightarrow PE_3 \rightarrow PE_5$. This retransmission is realized by executing transitions !*a*(t-set) in PE-SPEC1, ?*a*(t-set1)/!*a*(t-set2) in PE-SPEC2, ?*a*(t-set1)/!*a*(t-set2) in PE-SPEC3, and ?*a*(t-set) in PE-SPEC5. Here, t-set and t-set2 are interpreted as follows. Based on the source based routing policy in Section 2.4, e.g. !*a*(2) is actually executed for !*a*(t-set) in PE-SPEC1, and !*a*(3) is actually executed for !*a*(t-set2) in PE-SPEC2.

It is clear that for the channel failure between PE_1 and PE_5, the execution order of service primitives is kept in the transmission sequence denoted by bold arrows in Fig. 10.

6. Discussion

6.1. Flexibility for topology change

In this subsection, we describe only a procedure that realizes Requirement 3. At first, we explain the case of $G + \Delta G$ (i.e. some *PE*s are added). We suppose that *PE*s and their associated channels are incrementally added. In order to satisfy Requirement 3, we define a new problem FT as follows:

Problem FT. *Input:* (1) A protocol specification, which is obtained from Protocol Synthesis Problem (by applying our proposed method to a topology graph G with Restriction 1, and a service specification S with Restrictions 2 and 3. (2) Topology change ΔG due to addition or deletion of a node and its associated channels. We assume that the updated graph $G + \Delta G$ or $G - \Delta G$ still satisfies Restriction 1.

Output: A protocol specification, which is obtained by applying our synthesis method to the topology graph $G + \Delta G$ or $G - \Delta G$ and a service specification S.

In this problem FT, we discuss the process of realizing or obtaining the output, and show that the proposed method attains it effectively using *t-set*.

Assume that PE_i is a newly added node and $PE_{j_1}, PE_{j_2}, \ldots, PE_{j_k}$ are connected to PE_i as shown in Fig. 11. These are the elements of ΔG. The added PE_i can be *R-node*. Then, let PE_{j_p} and PE_{j_q} ($1 \le p \le k$, $1 \le q \le k$, $p \ne q$) are arbitrary two *PE*s among $PE_{j_1}, PE_{j_2}, \ldots, PE_{j_k}$. There exist loop-free directed paths from *S-node* to *D-node* through PE_{j_p} and PE_{j_q} in G. For the *PE*s on these paths, transitions for message delivery were specified in PE-SPECs. By adding PE_i and channels (i,j_p), (i,j_q), new loop-free directed paths, each of which consists of a subpath from *S-node* to PE_{j_p}, a subpath (j_p,i), (i,j_q), and a subpath from PE_{j_q} to *D-node*, appear in $G + \Delta G$. As, for the *PE*s on the subpath from *S-node* to PE_{j_p} and the subpath from PE_{j_q} to *D-node*, transitions for message delivery have been specified in PE-SPECs, modification of PE-SPEC is unnecessary for the *PE*s except for PE_{j_p} and PE_{j_q}. We have only to change, for each message e, !e($t\text{-}set_p$) in PE-SPECj_p and ?e($t\text{-}set_q$) in PE-SPECj_q into !e($t\text{-}set'_p$) where $t\text{-}set'_p = t\text{-}set_p \cup \{i\}$ and ?e($t\text{-}set'_q$) where $t\text{-}set'_q = t\text{-}set_q \cup \{i\}$, respectively. Similarly, if transition ?e($t\text{-}set_{i1}$)/!e($t\text{-}set_{i2}$) does not exist in PE-SPECi, then transition ?e($t\text{-}set_{i1}$)/!e($t\text{-}set_{i2}$) where $t\text{-}set_{i1} = \{j_p\}$ and $t\text{-}set_{i2} = \{j_q\}$ are added. Otherwise, the transition is updated to ?e($t\text{-}set_{i1} \cup \{j_p\}$)/!$e$($t\text{-}set_{i2} \cup \{j_q\}$).

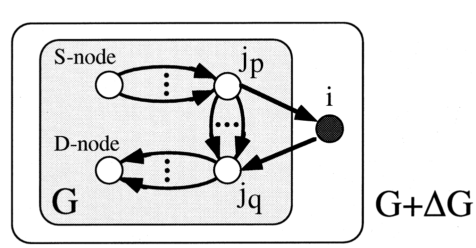

Fig. 11. Concept of *PE* added.

Next, we explain the case of $G - \Delta G$. As deletion of *S-node* or *D-node* makes message delivery impossible, we assume that the deleted *PE* is *R-node*. Assume that in Fig. 11 PE_i is a deleted node and that PE_i was connected to $PE_{j_1}, PE_{j_2}, \ldots, PE_{j_k}$. By eliminating PE_i and channels (i,j_p), (i,j_q), directed paths (j_p,i), (i,j_q) are deleted for any p and q $(1 \le p \le k,\ 1 \le q \le k,\ p \ne q)$. In this case, we have only to delete from PE-SPECj_p index $i \in t\text{-}set_p$ in !e($t\text{-}set_p$), and we also delete from PE-SPECj_q index $i \in t\text{-}set_q$ in ?e($t\text{-}set_q$). Then, we delete ?$e(t\text{-}set_{i1})$/!$e(t\text{-}set_{i2})$ in PE-SPECi.

6.2. An extension for multicast routing

Multicast communication services will be one of the most promising future applications, which includes real-time flows, in both the B-ISDN and the Internet [2].

We have proposed an extension of the synthesis method presented in this paper for multicast routing protocols, which are fault-tolerant [12]. The multicast routing protocol is defined to be fault-tolerant if messages can be retransmitted when a message loss occurs. The method generates a multicast routing protocol in consideration of behavior of a copy node. In the protocol, against a message loss, not a source node but a copy node can retransmit the message to destinations. Therefore, the retransmission can be fast.

In order to describe the behavior of a copy node efficiently, we consider two kinds of service specifications. One is a set of service specification between an *S-node* and *Copy nodes*. Another is a set of service specifications between a *Copy node* and *D-nodes*. The synthesis method presented in Ref. [12] is applied to both the specifications. Moreover, to handle the synchronization of messages in the copy node efficiently, a *fork state* and a *join state* are introduced into the protocol specification. As a result, several component pieces of protocol specifications are obtained. Finally, a final protocol specification is constructed from these pieces.

7. Conclusion

In this paper, we have proposed a new synthesis method which generates a fault-tolerant and flexible multipath routing protocol from a given service specification. The proposed method enables derivation of such a fault-tolerant protocol specification such that messages are rerouted at the source node and delivered to the destination node even when a communication path fails. Hence, the proposed design method enables the efficient production of reliable fault-tolerant routing protocol specification at a lower cost.

Further, for the given network changes, only PE specifications corresponding to the changes need to be modified in the obtained protocol. Therefore, only a small amount of change is needed for the change of network topology. This is useful for routing protocol for a network with large number of nodes. We also briefly describe an extension of the proposed method for generating multicast routing protocols.

References

[1] S. Dolev, J.L. Welch, Crash resilient communication in dynamic networks, IEEE Transactions on Computers 46 (1) (1997) 14–26.

[2] C. Huitema, Ipv6 The New Internet Protocol, 2, Prentice Hall, Englewood Cliffs, NJ, 1997.

[3] Y. Hatanaka, M. Nakamura, Y. Kakuda, T. Kikuno, A synthesis method for fault-tolerant and flexible multipath routing protocols, in: Proc. IEEE International Conference on Engineering of Complex Computer Systems, 1997, pp. 96–105.

[4] Y. Kakuda, M. Nakamura, T. Kikuno, Automated synthesis of protocol specifications with parallely executable multiple primitives, IEICE Transactions on Fundamentals E77-A (10) (1994) 1634–1645.

[5] R.L. Probert, K. Saleh, Synthesis of communication protocols survey and assessment, IEEE Transactions on Computers 40 (4) (1991) 468–476.

[6] M.T. Liu, Protocol Engineering, *Advances in Computers 29*, Academic, New York, 1989 pp. 79–195.

[7] G. Singh, M. Sammeta, On the construction of multiphase communication protocols, in: Proc. International Conference on Network Protocols, 1994, pp. 151–158.

[8] P.M. Chu, M.T. Liu, Protocol synthesis in a state transition model, in: Proc. International Computer Software and Applications Conference, 1988, pp. 505–512.

[9] K. Saleh, R.L. Probert, Automatic synthesis of protocol specifications from service specifications, Proc. Intl Phoenix Conference on Computers and Communications, 1991, pp. 615–621.

[10] J.E. Hopcroft, J.D. Ullman, Introduction to Automata Theory, Language, and Computation, Addison-Wesley, Reading, MA, 1979.

[11] J.V. Leewen, HandBook of Theoretical Science, Elsevier, Amsterdam, 1990.

[12] K. Ishida, K. Amano, A synthesis method for fault-tolerant multicast routing protocol, *Parallel and Distributed Processing 1388*, Springer, Berlin, 1998 pp. 1121–1130.