Introduction to Intelligent Network Routing
Based on EVALPSN

Kazumi Nakamatsu\(^1\) Jair Minoro Abe\(^2\) and Takashi Watanabe\(^3\)

\(^1\) School of H.S.E., University of Hyogo, Himeji, JAPAN
nakamatsu@shse.u-hyogo.ac.jp
\(^2\) Paulista University, University of Sao Paulo, Sao Paulo, BRAZIL
jairabe@uol.com.br
\(^3\) Faculty of Information, Shizuoka University, Hamamatsu, JAPAN
watanabe@inf.shizuoka.ac.jp

Abstract. The goal of our work is to make up an EVALPSN based multi-path ad-hoc network routing protocol system in which various kinds of protocols can be dealt with uniformly. In this paper, as the first step to our goal we introduce a single-path ad-hoc mobile network routing protocol based on a paraconsistent annotated logic program EVALPSN with a simple example of ad-hoc network routing, which is a translation of the DSR(Dynamic Source Routing) protocol into EVALPSN. Mainly route discovery is focused on in this paper.

Keywords: ad-hoc network, multi-path network routing protocol, paraconsistent annotated logic program, EVALPSN.

1 Introduction

We have already developed a paraconsistent annotated logic program called Extended Vector Annotated Logic Program with Strong Negation (abbr. EVALPSN) in [3,11] that has been applied to various real-time intelligent control and safety verification systems such as pipeline process control in [4]. Moreover, EVALPSN can deal with before-after relation between processes (time intervals) and has been applied to process order control in [8,9]. EVALPSN based intelligent control/verification have the following features: (1) since EVALPSN can deal with deontic notions such as forbiddance, control properties in deontic expression can be easily and directly formalized in EVALPSN; (2) logical verification of operation control can be easily carried out as logic programming; (3) since some restricted fragment of EVALPSN can be implemented on microchips as electronic circuits [5], EVALPSN is a suitable tool providing real-time control. Therefore, EVALPSN can provide a useful platform for all kinds of intelligent information systems in both practical and theoretical senses.

Recently, wireless ad-hoc communication network systems in which nodes can communicate directly each other without via sites have been focused on. In such ad-hoc networks each distributed mobile node has to control communication between nodes autonomously and some ad-hoc network control methods have been proposed. Among them routing protocol is a control method that deal
with two mechanism, route discovery and route maintenance. If we consider data transmission efficiency in ad-hoc mobile networks, multi-path routing protocols that deal with more than two routes for communication should be more appropriate than single-path routing protocols. Various kinds of multi-path routing protocols that are object-orientedly improved versions of the basic one have been proposed and each multi-path routing protocol has different advantages. Therefore, if we utilize various kinds of multi-path routing protocols efficiently, an intelligent platform on which they can be implemented is indispensable. Our near future work is to make up an intelligent multi-path routing system based on EVALPSN in which various kinds of multi-path routing protocols can be treated uniformly. As the first step to our goal, we have implemented the single-path DSR (Dynamic Source Routing) Protocol in EVALPSN, which is introduced in this paper.

The Dynamic Source Routing (DSR) protocol is a simple and efficient routing protocol designed for multi-hop wireless ad-hoc networks of mobile nodes [1, 2]. The DSR protocol consists of two mechanisms, route discovery and route maintenance, and three kinds of messages, route request message, route reply message for route discovery and route error message for route maintenance are used in the DSR. The operations for the three messages at each node can be expressed in rule sentences, and they are easily translated into EVALPSN.

This paper is organized in the following manner: first, EVALPSN is reviewed and the basic ad-hoc network routing protocol DSR is introduced and its node operations are described in if-then rule form; next the node operations are formalized in EVALPSN and a simple example of the EVALPSN based network routing is introduced; last, this paper is concluded with our future work.

We omit the explanation and definition of EVALPSN due to space restriction. The details of EVALPSN can be found in [3,11].

2 Ad-hoc Network Routing Protocol DSR

As preliminary, we provide some definitions. A route is defined as an ordered list of the finite number of nodes such as \(< n_1, n_2, \ldots, n_k >\), where each \( n_i (1 \leq i \leq k) \) represents a node id; the origin node is as a node that has sent the message initially; the object node is as a node that has just received the message; and the source node is as a node that has sent the message immediately before the object node receives the message.

First of all, we introduce a well known single-path routing protocol, DSR (Dynamic Source Routing) protocol [1, 2]. The DSR protocol deals with three kinds of network messages called route request message to discover routes, route reply message to reply for route request if the requested route has been made up, and route error message to report route error for the origin node if the route has been cut. The DSR protocol contains the node operations for the three kinds of network messages, which are summarized in the following seven operation rules. Those operation rules will be translated into EVALPSN in the following section.

[Node Operations in DSR Protocol]
1. **route request message discarding**  If the object node receives a route request message that has been already received, it should be discarded.

2. **route request message replying**  If the object node is the destination of the route request message that should not be discarded, the route request message should not be flooded and the route reply message for the route request should be sent back to the origin node according to the reverse order of the route node list in the route request message;

3. **route request flooding**  otherwise, the route request message should be flooded with adding the object node id to the route node list in the route request message.

4. **route reply message transferring**  If the object node receives a route reply message whose destination node is not the object node, the route reply message should be transferred according to the reverse order of the route node list.

5. **route reply message termination**  If the object node is the destination of a route reply message, the route reply message should be terminated with completing the route discovery.

6. **route error message transferring**  If the object node detects a route error directly or receives a route error message from other nodes, and the object node is not the destination of the route error message, the route error message should be transferred according to the reverse order of the route node list;

7. **route error message termination**  otherwise, the route error message should be terminated with starting another route discovery.

![Network Diagram]

**Fig. 1.** Network Route Message transferring

We describe a simple example of the DSR protocol operations with taking the network in Figure 1.

**Example**
Suppose that a route discovery request to node $N_d$ has been issued at $N_0$; then node $N_0$ floods the route request message to all the neighbor nodes $N_1$, $N_3$ and $N_4$; if node $N_4$ receives the route request message from node $N_0$, node $N_4$ should check whether the same route request message has been already received or not;
then since the route request message has not been received by node $N_4$ before, it
should be flooded to nodes $N_5$, $N_6$ and $N_7$ with adding node $N_4$ id $n_4$ to the end
of the route node list $<n_0,(n_4)>$ in the route request message; on the other
hand, if the same route request message arrived at node $N_4$ via node $N_3$ later
than the previous flooding of the message, it should be discarded; suppose that
node $N_d$ has received the same route request message whose destination is node
$N_d$ from node $N_6$, which has the route node list $<n_0,n_4,n_6>$, then node $N_d$
should send the route reply message with the route node list $<n_0,n_4,n_6,n_d>$
back to the origin node $N_0$ via nodes $N_6$ and $N_4$; if node $N_6$ has received
the route reply message from node $N_d$, it should be transferred to node $N_4$ according
to the reverse order of the route node list.

The route request message flooding is shown by the solid line arrow symbol
$\rightarrow$ and the route reply path is by the chained line arrow $\longrightarrow$ in Figure 1.

3 DSR(Dynamic Source Routing) in EVALPSN

In order to translate the DSR protocol into EVALPSN, three kinds of messages,
route request, route reply and route error should be formalized in EVALP clauses
as follows.

![Fig. 2. The Complete Lattice of Route Message Operations]

**Route request message** contains route request message id $req_d$, the object
node $obn$ where the message has been received, the source node $son$ where the
message has been flooded, the destination node $den$ of the route request message,
and the ordered list $n_{seq}$ of nodes representing the route from the origin node
to the source node. Node operations for route request messages are receiving
(represented by annotation $rec$ in EVALPSN), flooding (by annotation $fld$),
which means sending the route request message to anonymous nodes, replying
(by annotation rep), which means sending the route reply message to the origin
node according to the reverse order of the route node list in the route request
message, and discarding (by annotation dis), which means discarding the route
request message if the same one has been already received.

Here we analyze the node operations represented by the annotations rec,
fld, rep and dis with the independent bi-criteria, message transferability and
knowledge amount. If we take the viewpoint of message transferability, we may
have the order between the node operations, flooding, transferring(replying),
discarding, and operation receiving is neutral for message transferability. On the
other hand, if we take the viewpoint of knowledge amount, since four operations,
flooding, transferring(replying) and discarding, should be executed after operation
receiving, the four operations imply knowledge that the message has been
already received. Therefore, if we distribute those annotations representing the
node operations in the complete lattice of vector annotations, they are located
as shown in Figure 2. The complete lattice is a bi-lattice in which its vertical
criterion is knowledge amount and its horizontal one is message transferability.
For example, we may regard that operation flooding has the highest message
transferability and more knowledge than operation receiving in terms of message
treatment. On the other hand, operation discarding may be regarded that it
has the lowest message transferability.

The route request message is formalized in the EVALP literal,

\[
\text{rreq}(\text{req_id}, \text{obj}, \text{son}, \text{den}, \text{nseq}, t)[\mu_1, \mu],
\]

where \( \mu_1 \in \{ \bot(0,0), \ldots, \text{rec}(1,1), \ldots, \text{dis}(1,3), \ldots, \text{rep}(2,2), \ldots, \text{fld}(3,1), \ldots, \text{ten}(3,3) \} \), and \( \mu \in \mathcal{T}_r \). For example, the EVALP clause \( \text{rreq}(\text{req_id}, \text{obj}, \text{son}, \text{den}, \text{nseq}, t)[\text{fld}, \beta] \) can be intuitively interpreted that the route request message \( \text{req_id} \) flooded from the source node \( \text{son} \) must be flooded from the object node \( \text{obj} \) with node list \( \text{nseq} \) and the destination node \( \text{den} \) at time \( t \).

**Route reply message** contains route reply message id \( \text{rep_id} \), which is the
same as the corresponding route request message id, the object node \( \text{obj} \), the
source node \( \text{son} \), the destination node \( \text{den} \), which is the origin node of the cor-
responding route request message, the ordered list \( \text{nseq} \) of nodes representing
the route. Node operations for route reply messages are receiving(represented by
annotation rec in EVALPSN as well as route request messages), transferring(by
annotation tra), which means sending the route reply message directly to the
next node in the ordered list \( \text{nseq} \) of nodes, and terminating(by annotation ter),
which means the termination operation for route reply messages if the object
node is the destination of the route reply message. The complete lattice structure
of annotations rec, tra and ter for route reply messages are also shown
in Figure 2 as well as those for route request messages. Therefore, route reply
message is formalized in the EVALP literal,

\[
\text{rrep}(\text{rep_id}, \text{obj}, \text{son}, \text{den}, \text{nseq}, t)[\mu_2, \mu],
\]

where \( \mu_2 \in \{ \bot(0,0), \ldots, \text{rec}(1,1), \ldots, \text{ten}(1,3), \ldots, \text{tra}(2,2), \ldots, \text{ten}(3,3) \} \), and
\( \in \mathcal{T}_r \). For example, the EVALP clause \( \text{rrep}(\text{rep_id}, \text{obj}, \text{son}, \text{den}, \text{nseq}, t)[\text{rec}, \alpha] \)
can be intuitively interpreted that the route reply $r_{p_{id}}$ transferred from the source node $son$ has been received with the node list $n_{seq}$ and the destination node $den$ by the object node $obn$ at time $t$.

**Route error message** contains route error message id $err_{id}$, the object node $obn$ where the message has been received, the source node $son$ where the message has been transferred, the destination node $den$ of the route error message, and the ordered list $n_{seq}$ of nodes representing the route. Node operations for route error messages are receiving (represented by annotation $rec$ in EVALPSN), transferring (by annotation $tra$) and terminating (by annotation $ter$) as well as those for route reply message. Therefore, route error message is formalized in the EVALP literals,

$$rerr(err_{id}, obn, son, den, n_{seq}, t) \colon [\mu_2, \mu],$$

where $\mu_2 \in \{ \bot(0, 0), \ldots, rec(1, 1), \ldots, ter(1, 3), \ldots, tra(2, 2), \ldots, T_1(3, 3) \}$, and $\mu \in T$. For example, the EVALP clause $rerr(err_{id}, obn, son, den, n_{seq}, t) : [s, \beta]$ can be intuitively interpreted that the route error message $err_{id}$ transferred from the node $son$ must be sent with the node list $n_{seq}$ and the destination node $den$ from the object node $obn$ at time $t$.

We need to define another EVALP literal to formalize the DSR operations. **Relation between nodes in ordered list of nodes** is formalized in the EVALP literal,

$$rel(n_i, n_j, n_{seq}) : [\mu_3, \mu],$$

where $\mu_3 \in \{ \bot(0, 0), \ldots, pr(2, 0), eq(1, 1), su(0, 2), \ldots, T(1, 1) \}$, and $\mu \in T$, and annotations $pr$, $eq$, and $su$ declare that one node precedes/is equal to/succeeds another one. For example, the EVALP literal, $rel(n_4, n_0, < n_0, n_4 >) : [su, \alpha]$ can be intuitively interpreted that node $N_4(n_4)$ succeeds node $N_0(n_0)$ in the ordered list $n_{seq}$ of nodes.

The node operations described in the previous section are translated into EVALPSN.

1. **route request message discarding**

$$rreq(req_{id}, obn, son, den, n_{seq}, t) : [rec, \alpha] \land$$
$$b(t', t) : [t, \alpha] \rightarrow$$
$$rreq(req_{id}, obn, son, den, n_{seq}, t) : [dis, \beta] \quad (1)$$

where $b(t', t) : [t, \alpha]$ declares that time $t'$ is before time $t$.

2. **route request message replying**

$$rreq(req_{id}, obn, son, obn, n_{seq}, t) : [rec, \alpha] \land$$
$$\sim rreq(req_{a}, obn, son, obn, n_{seq}, t) : [dis, \beta] \rightarrow$$
$$rrep(req_{id}, son, obn, hn, n_{seq} \cup < obn >, t) : [tra, \beta], \quad (2)$$

where $n_{seq} \cup < obn >$ represents the list that the object node $obn$ is added to the node list $n_{seq}$ as its last member, and $hn$ represents the origin node of the route request in the node list $n_{seq}$. 

3. route request message flooding

\[ \text{rcr}(\text{req}_d, \text{obn}, \text{den}, t): [\text{is}, \alpha] \rightarrow \]
\[ \text{rrq}(\text{req}_d, \text{obn}, \text{des}, <n_o, t>): [\text{fld}, \beta] \] (3)

\[ \text{rrq}(\text{req}_d, \text{obn}, \text{son}, \text{den}, \text{nseq}, t): [\text{rec}, \alpha] \land \]
\[ \sim \text{rrq}(\text{req}_d, \text{obn}, \text{son}, \text{den}, \text{nseq}, t): [\text{dis}, \beta] \land \]
\[ \sim \text{rrp}(\text{req}_d, \text{son}, \text{obn}, \text{hn}, \text{nseq} \cup <\text{obn}, t>): [\text{tra}, \beta] \rightarrow \]
\[ \text{rrq}(\text{req}_d, \text{amo}, \text{obn}, \text{den}, \text{nseq} \cup <\text{obn}, t>): [\text{fld}, \beta], \] (4)

where \( \text{rcr}(\text{req}_d, n_o, \text{den}, t): [\text{is}, \alpha] \) declares that the network user of node \( n_o \) has issued the route request \( \text{req}_d \).

4. route reply message transferring

\[ \text{rrp}(\text{req}_d, \text{obn}, \text{son}, \text{den}, \text{nseq}, t): [\text{rec}, \alpha] \land \]
\[ \sim \text{rrp}(\text{req}_d, \text{obn}, \text{son}, \text{den}, \text{nseq}, t): [\text{ter}, \beta] \land \]
\[ \land \text{rel}(\text{nen}, \text{obn}, \text{nseq}): [\text{prec}, \alpha] \rightarrow \text{rrp}(\text{req}_d, \text{nen}, \text{obn}, \text{den}, \text{nseq}, t): [\text{tra}, \beta] \] (5)

5. route reply message termination

\[ \text{rrp}(\text{req}_d, \text{obn}, \text{son}, \text{obn}, \text{nseq}, t): [\text{rec}, \alpha] \rightarrow \]
\[ \text{rrp}(\text{req}_d, \text{obn}, \text{son}, \text{obn}, \text{nseq}, t): [\text{ter}, \beta]. \] (6)

6. route error message transferring

\[ \text{error}(\text{obn}, \text{ern}, t): [\text{d}, \alpha] \land \text{rrp}(\text{req}_d, \text{obn}, \text{ern}, \text{den}, \text{nseq}, t'): [\text{tra}, \alpha] \land \]
\[ \text{bf}(t', t): [\text{t}, \alpha] \land \text{rel}(\text{nen}, \text{obn}, \text{nseq}): [\text{prec}, \alpha] \rightarrow \]
\[ \text{rrerr}(\text{req}_d, \text{nen}, \text{obn}, \text{den}, \text{nseq}, t): [\text{tra}, \beta], \] (7)

where \( \text{error}(\text{obn}, \text{ern}, t): [\text{d}, \alpha] \) declares that a route error has been detected between the object node \( \text{obn} \) and the error node \( \text{ern} \) at time \( t \),

\[ \text{rrerr}(\text{req}_d, \text{obn}, \text{son}, \text{den}, \text{nseq}, t): [\text{rec}, \alpha] \land \]
\[ \text{rrp}(\text{req}_d, \text{obn}, \text{ern}, \text{den}, \text{nseq}, t'): [\text{tra}, \alpha] \land \]
\[ \text{bf}(t', t): [\text{t}, \alpha] \land \text{rel}(\text{nen}, \text{obn}, \text{nseq}): [\text{prec}, \alpha] \rightarrow \]
\[ \text{rrerr}(\text{req}_d, \text{nen}, \text{obn}, \text{den}, \text{nseq}, t): [\text{tra}, \beta]. \] (8)

7. route error message termination

\[ \text{rrerr}(\text{req}_d, \text{obn}, \text{son}, \text{obn}, \text{nseq}, t): [\text{rec}, \alpha] \rightarrow \]
\[ \text{rrerr}(\text{req}_d, \text{obn}, \text{son}, \text{obn}, \text{nseq}, t): [\text{ter}, \beta] \]
\[ (\text{rcr}(\text{req}_d, \text{obn}, \text{den}, t): [\text{is}, \beta]), \] (9)

where the route error message termination may be changed for new route discovery.
4 Example

Here we take the same DSR protocol example shown in Figure 1 and show the EVALPSN network routing based on the DSR protocol.

Stage 1 (time \( t_1 \)) at node \( N_0(n_0) \), suppose that a network user issued the route request \( \text{req}_1 \) with the destination node \( N_d(n_d) \), then we have the fact EVALP clause,

\[
\text{rcr}(\text{req}_1, n_0, n_d, t_1) : [\text{is}, \alpha],
\]

and the obligatory EVALP clause,

\[
\text{rreq}(\text{req}_1, \text{ano}, n_0, n_d, < n_0 >, t_1) : [\text{fld}, \beta]
\]

for flooding the route request message \( \text{req}_1 \) is derived by the EVALP clause (3).

Stage 2 (time \( t_2 \)) at node \( N_d(n_d) \), suppose that the flooded route request message \( \text{req}_1 \) from node \( N_0 \) has been received, then we have the fact EVALP clause,

\[
\text{rreq}(\text{req}_1, n_d, n_d, < n_0 >, t_2) : [\text{rec}, \alpha].
\]

Moreover, since the obligatory EVALP clauses,

\[
\text{rreq}(\text{req}_1, n_d, n_d, < n_0 >, t_2) : [\text{dis}, \beta] \text{ and } \\
\text{rrep}(\text{req}_1, n_0, n_d, < n_0, n_4 >, t_2) : [\text{tra}, \beta]
\]

cannot be derived by the EVALP clauses (1) and (2), we obtain only the obligatory EVALP clause,

\[
\text{rreq}(\text{req}_1, \text{ano}, n_0, n_d, < n_0 >, t_1) : [\text{fld}, \beta]
\]

for flooding the route request message \( \text{req}_1 \) by the EVALP clause (3).

Stage 3 (time \( t_3 \)) at node \( N_4(n_4) \), suppose that the flooded route request message from node \( N_0 \) has been received, then we have the fact EVALP clause,

\[
\text{rreq}(\text{req}_1, n_4, n_3, n_d, < n_0, n_3 >, t_3) : [\text{rec}, \alpha],
\]

and we already have the fact EVALP clause (10), which can be regarded as the record of route request messages received at node \( N_4 \) before, then the obligatory EVALP clause,

\[
\text{rreq}(\text{req}_1, n_4, n_3, n_d, < n_0, n_3 >, t_3) : [\text{dis}, \beta]
\]

for discarding the route request with the same request id \( \text{req}_1 \) is derived by the EVALP clause (1).

Stage 4 (time \( t_4 \)) at node \( N_6(n_6) \), suppose that the flooded route request message from node \( N_4 \) has been received, then the same operations at node \( N_4 \) are carried out and the obligatory EVALP clause,

\[
\text{rreq}(\text{req}_1, \text{ano}, n_4, n_d, < n_0, n_4, n_6 >, t_4) : [\text{fld}, \beta],
\]
for flooding the route request message \( \text{req}_1 \) is derived by the EVALP clause (3).

**Stage 5 (time \( t_5 \))** at node \( N_d(n_d) \), suppose that the flooded route request message \( \text{req}_1 \) from node \( N_6 \) has been received, then we have the fact EVALP clause,

\[
\text{req}(_1,n_d,n_6,n_d, < n_o,n_4,n_6 >,t_5):[\text{rec},\alpha].
\]

Since the obligatory EVALP clause,

\[
\text{req}(_1,n_d,n_6,n_d, < n_o,n_4,n_6 >,t_5):[\text{dis},\beta],
\]
cannot be derived by the EVALP clause (1), the obligatory EVALP clause,

\[
\text{rep}(_1,n_d,n_6,n_0, < n_o,n_4,n_6,n_d >,t_5):[\text{tra},\beta],
\]
for transferring the route reply message for the route request message \( \text{req}_1 \) is derived.

**Stage 6 (time \( t_6 \))** at node \( N_6(n_0) \), suppose that the route reply message \( \text{req}_1 \) transferred from node \( N_4 \) has been received, then we have the fact EVALP clause,

\[
\text{rep}(_1,n_0,n_d,n_0, < n_o,n_4,n_6,n_d >,t_6):[\text{rec},\alpha].
\]

However, the obligatory EVALP clause,

\[
\text{rep}(_1,n_0,n_d,n_0, < n_o,n_4,n_6,n_d >,t_6):[\text{tra},\beta]
\]
cannot be derived by the EVALP clause (6). Therefore, we obtain the obligatory EVALP clause,

\[
\text{rep}(_1,n_4,n_6,n_0, < n_o,n_4,n_6,n_d >,t_6):[\text{tra},\beta],
\]
for transferring the route reply message \( \text{req}_1 \) is derived by the EVALP clause (5).

### 5 Future Work and Conclusion

Our goal of this work is to make up an EVALPSN based intelligent routing system, which can deal with various kinds of routing protocols on the same EVALPSN platform uniformly. As the first step to the goal, we have introduced the most basic EVALPSN single-path routing protocol based on the DSR protocol in this paper, as the next step, we are planning to make up an EVALPSN multi-path routing system in which various kinds of multi-path routing protocols can be dealt with on the same EVALPSN platform uniformly.

As the advantage of the EVALPSN multi-path routing system, we are considering to apply defeasible/plausible reasoning in EVALPSN \([7,10]\) to two kinds of decision making, one is route-discovery in multi-path routing and another one is protocol-selection in the EVALPSN multi-path ad-hoc routing system. It is preferable to discover more than two mutually independent routes as possible in multi-path routing, and EVALPSN defeasible/plausible reasoning can
be applied to discovering such routes. Furthermore, it is also preferable to select the most suitable routing protocol under the network environment such as data traffic congestion in real-time, and the defeasible/plausible reasoning can be also applied to selecting such a suitable protocol. In practice we have already applied EVALPSN defeasible reasoning to traffic signal control aiming at traffic jam reduction in [6], and the research results could be applied to ad-hoc network routing with intelligent data traffic control. ⁴

References


⁴ This work is financially supported by Japanese Scientific Research Grant (C) No. 20500074.