Proceedings of the 2014 Federated Conference on Computer Science and Information Systems pp. 1559–1568
DOI: 10.15439/2014F73
ACSIS, Vol. 2

Handling Conflicts to Test Transport Protocol's Parallel Routing on a Vehicle Gateway System

Hassan Mohammad
MBtech Group GmbH & Co. KGaA
Sindelfingen 71063, Germany
Email:hassan.mohammad@mbtech-group.com

Muhammad Shamoon Saleem
Ingolstadt University of Applied Sciences
Ingolstadt 85049, Germany
Email:ia2632@thi.de

***Abstract*—This paper addresses the issue of verifying transport protocol's parallel routing functionality on a vehicle gateway system. The focus of the paper is to construct a conflict-free input parameter model for testing this functionality. The input parameter model shall support the reduction of combinations to be tested and serves as a basis for automatic test case generation from a large space of input parameters. In the proposed approach, defined similarity criteria are used to cluster system input parameters represented as transport protocol routing instances into groups which stimulate similar behavior in the gateway when transport protocol routing is established. Subsequently, the two conflict-handling methods *sub-models* and *avoid* are utilized to prohibit invalid combinations of transport protocol routing instances. The proposed approach is applied on a complex example of real gateway with five buses, 390 transport protocol routing instances and diverse conflicts to illustrate its applicability.**

***Index Terms*—Transport Protocol's Parallel Routing, Input Parameter Model (IPM), Conflict Handling**

I. Introduction

TODAY'S vehicles Electric/Electronic (E/E) systems are designed as distributed systems in order to overcome the increasing complexity and to meet the diversity of requirements such as performance, comfort, safety and costs. In a vehicle distributed system, gateways are indispensable. They enable Electric Control Units (ECUs) within connected networks to interchange information necessary for accomplishing specified functionalities. During information interchange, the gateway routes data between its connected networks although they work on different communication protocols.

In a vehicle gateway system, routing data packets which are larger than a single frame of the corresponding network communication protocol is carried out with the help of transport protocol implementations and such type of routing is called TP routing. Since TP data packets are mostly large in size (flash data for example) and routing of such large data packets requires longer duration, modern gateways support TP parallel routing where multiple TP routing instances are established in parallel over the gateway in order to save time and resources, e.g., flashing multiple ECUs in parallel.

Verifying TP parallel routing of a gateway system is not a trivial problem, since a large number of possible combinations of communicating ECUs can be built for test case selection (the combinatorial explosion problem). Furthermore, in case of established TP parallel routing, different types of conflicts between ECUs need to be handled.

Combination strategies [1] are test case selection methods that focus on solving the combinatorial explosion problem raised while testing the interactions between system input parameter values by defining coverage criteria to satisfy. However, the problem in the case of verifying TP parallel routing on a vehicle gateway system is different than described problem of combination test strategies (see II-B). Hence, new techniques need to be defined.

In combination strategies, Input Parameter Models (IPMs) [2] [3] [4] [5] are essential. They represent the System Under Test (SUT) on an abstract level. Mostly, IPMs contain conflicts which must be resolved. A conflict in an IPM is due to an invalid combination of input parameter values and hence this combination must be omitted or avoided while generating test cases. Diverse conflict handling strategies such as *sub-models* and *avoid* have been suggested in literature [6] to overcome conflicts in IPMs.

This paper suggests an approach to build a conflict-free IPM used to test TP parallel routing on a vehicle gateway system. The IPM is utilized in a recursive way for test case selection, generation and execution in order to overcome the combinatorial explosion problem.

The remainder of this paper is organized as follows. Section II gives background information on terminology and the combinatorial explosion problem of testing TP parallel routing. An IPM along with conflict handling mechanisms are presented in details in section III. In section IV, the suggested IPM along with described conflict handling mechanisms are utilized in a method to reduce test suite size. The complete methodology is applied on a complex example of real gateway in section V. Section VI discusses the approach and section VII concludes the paper.

II. Background

A. Vehicle Bus Communication Systems

Generally, different bus communication systems are utilized in E/E system. In this subsection, two common types of automotive buses are briefly explained.

This work was supported by MBtech Group GmbH & Co. KGaA

978-83-60810-58-3/$25.00

1) CAN Communication Bus: Controller Area Network (CAN) bus system was originally invented to reduce the wiring harness in automotive E/E systems by developing a serial communication protocol. CAN has the capability to connect multiple ECUs directly to one medium, which leads to better management of the system complexity and the reduction of manufacturing costs. CAN protocol is an asynchronous serial protocol that enables real time communication. In CAN, the medium access is based on the concept of Carrier Sense Multiple Access/Collision Detection (CSMA/CD).

2) FlexRay Communication Bus: FlexRay was developed to deliver deterministic, fault-tolerant and high-speed communication bus required for x-by-wire applications such as steer-by-wire and break-by-wire. These applications demand more safety, performance and reliability than provided by CAN. FlexRay unifies time- and dynamic event-triggering mechanisms in one protocol. The communication in FlexRay is based on the Time Devision Multiple Access (TDMA) and Flexible Time Division Multiple Access (FTDMA) schemes of networking.

B. Terminology

The vehicle gateway is part of a distributed network system. It is a special ECU that has multiple communication channels used to communicate with other ECUs in the network in order to route data between them. Communication channels of the gateway are mostly heterogeneous in respect of characteristics and behavior. Generally, a number of ECUs (u) can exchange data over the gateway in predefined fashions. Each fashion is characterized through a set of gateway configuration parameters. These are required by the gateway to establish routing between communicating ECUs connected to different communication channels. For TP routing over the gateway, following definitions are considered:

A *TP_Routing_Fashion* describes a possible routing behavior of TP data and is characterized through a particular set of gateway configuration parameters. An example of a $TP\_Routing\_Fashion_F$ with P parameters shall be described (1) (see [7] for configuration parameters of CAN TP).

$$TP\_Routing\_Fashion_F = \{P_{F_1}, P_{F_2}, ..., P_{F_P}\} \quad (1)$$

A *TP_Connection_Channel* is an instance of a *TP_Routing_Fashion*. It has the same set of gateway configuration parameters and is utilized to route TP data between respective ECUs. A $TP\_Connection\_Channel_x$ in the $TP\_Routing\_Fashion_F$ shall be described (2).

$$TP\_Connection\_Channel_x = \{P_{F_1x}, P_{F_2x}, ..., P_{F_Px}\} \quad (2)$$

The gateway has a number of configured *TP_Connection_Channel*s for connected ECUs utilized to establish TP routing in different scenarios. Examples of TP scenarios are flashing and Onboard Diagnostic (OBD). *TP_Routing_Scenario*s shall be described as a group of s scenarios (3).

$$TP\_Routing\_Scenarios = \{S_1, S_2, ..., S_s\} \quad (3)$$

A *TP_Routing_Instance* is a relationship between a specific *TP_Connection_Channel* and a possible *TP_Routing_Scenario*. An example of a $TP\_Routing\_Instance_x$ shall be described (4).

$$TP\_Routing\_Instance_x = \{P_{F_1x}, P_{F_2x}, ..., P_{F_Px}, S_x\} \quad (4)$$

The gateway can be configured to serve y *TP_Routing_Instance*s in parallel. The number of parallel *TP_Routing_Instance*s *"y"* is a configuration parameter which needs to be verified. In the case of errors, the next determined *"y"* should be verified.

C. The Combinatorial Explosion Problem of Testing TP Parallel Routing

The combinatorial explosion problem mentioned in literature [8] [9] [10] shall be explained as in the following example:

Assume a distributed system consisting of a central unit interacting over communication channels with u units of the network U_1, U_2, ..., U_u. Each unit U_i uses a defined parameter p_i for communication. The parameter p_i shall have v_i possible configuration values. By assuming that configuration values of parameters are independent from each other, the number of possibilities in which the system can be configured would be $v_1 * v_2 * ... * v_u$. If each possible configuration requires c test cases to verify it, the number of test cases for exhaustive test would be $c * v_1 * v_2 * ... * v_u$. In a nontrivial software system, the values of u and v_i are large which leads to a huge number of possible combinations of parameter values.

Related to testing TP parallel routing, the goal of test is to measure the performance of the gateway to handle parallel *TP_Routing_Instance*s. The problem is more complex because:

- System input parameters are *TP_Routing_Instance*s where each consists of a set of configuration parameters.
- The number of system input parameters is not fixed. It can be different for every new release of the system.
- System input parameters include also timing parameters where the interactions are difficult to resolve.
- The number of parallel *TP_Routing_Instance*s *"y"*, which is also a configuration parameter, is used to build possible combinations to be tested. Combinations are any y elements from the system input parameter set. In case of errors, one of the goals is to determine the next *"y"* and verify it (performance measurement).
- In TP parallel routing, each additional instance will consume resources of the system and may lead to errors. Hence, it is not only a specific combination of *TP_Routing_Instance*s which can affect the behavior and may reveal errors, but also the number of included *TP_Routing_Instance*s and their values.

To verify *"y"* parallel routing of *TP_Routing_Instance*s and determine the next *"y"* in case of errors, all possible combinations from 1 to y of *TP_Routing_Instance*s should be included at least once in test cases. This results in a number

of combinations to be tested which can be calculated using equation(5).

$$X = \frac{u!}{y!(u-y)!}.y^s + \frac{u!}{(y-1)!(u-(y-1))!}.(y-1)^s + ... + \frac{u!}{1!(u-1)!}.1 \quad (5)$$

The equation (5) calculates the sum of all possible combinations for a number of ECUs (u) communicating in *TP_Routing_Scenario*s (s), where the selected number of ECUs in each term varies from y to 1.

Even for a simple system, the duration of testing for all these combinations is too high and therefore not acceptable.

III. Input Parameter Model and Conflict Handling

A combination test strategy consists generally of two steps:

1) Building a suitable IPM
2) Selecting combinations of parameter values from IPM to satisfy defined coverage criteria.

In this paper, the proposed IPM shall be used to test TP parallel routing in a recursive approach, i.e., one combination is constructed at a time. Subsequently, a test case is generated for that combination and executed. This procedure is then repeated until satisfying the defined coverage criterion. The decision on using a recursive selection and execution approach has two reasons:

1) Information from executed test cases is collected and can be used to reduce the number of combinations for the successive test cases.
2) Gained information from executed test cases is utilized to determine the worst case combinations.

In order to build a suitable conflict-free IPM, conflicts need to be defined. In a TP parallel routing, three types of conflicts are stated:

1) **Type$_A$-Conflicts**: Conflicts between *TP_Routing_Scenario*s. These are constraints describing *TP_Routing_Scenario*s not practiced in parallel.
2) **Type$_B$-Conflicts**: Conflicts between *TP_Connection_Channel*s. These are constraints describing *TP_Connection_Channel*s not allowed to be combined in parallel.
3) **Type$_C$-Conflicts**: Conflicts because of configuration parameters. These are constraints describing known or desired capabilities of the SUT.

The following example explains raised conflicts in testing TP parallel routing on a vehicle gateway system:

Assume a gateway having six *TP_Routing_Instance*s in *TP_Routing_Fashion*s (F_x and F_y) used to route TP data between connected ECUs as following:

$$TP\_Routing\_Instance_1 = (P_{F_{x11}}, P_{F_{x21}}, P_{F_{x31}}, P_{F_{x41}}, S_1)$$
$$TP\_Routing\_Instance_2 = (P_{F_{x12}}, P_{F_{x22}}, P_{F_{x32}}, P_{F_{x42}}, S_2)$$
$$TP\_Routing\_Instance_3 = (P_{F_{x13}}, P_{F_{x23}}, P_{F_{x33}}, P_{F_{x43}}, S_1)$$
$$TP\_Routing\_Instance_4 = (P_{F_{x14}}, P_{F_{x24}}, P_{F_{x34}}, P_{F_{x44}}, S_2)$$
$$TP\_Routing\_Instance_5 = (P_{F_{y15}}, P_{F_{y25}}, P_{F_{y35}}, S_1)$$
$$TP\_Routing\_Instance_6 = (P_{F_{y16}}, P_{F_{y26}}, P_{F_{y36}}, S_2)$$

Assume also that the gateway supports two *TP_Routing_Instance*s in parallel. Then, following examples explain the three types of conflicts:

- **Type$_A$-Conflicts**: *TP_Routing_Scenario* S_1 and *TP_Routing_Scenario* S_2 are non-combinable. That is, a combination of *TP_Routing_Instance*$_1$ and *TP_Routing_Instance*$_2$ is for example an invalid combination.
- **Type$_B$-Conflicts**: *TP_Connection_Channel*$_2$ and *TP_Connection_Channel*$_4$ are non-combinable, i.e., a combination of *TP_Routing_Instance*$_2$ and *TP_Routing_Instance*$_4$ is an invalid combination.
- **Type$_C$-Conflicts**: Maximum of two *TP_Routing_Instance*s are combinable, i.e., all combinations of more than two *TP_Routing_Instance*s are invalid.

As discussed in the previous section, a *TP_Routing_Instance* of gateway is a relationship between *TP_Connection_Channel* and possible *TP_Routing_Scenario*. *TP_Routing_Instance*s are input parameter values required to build a conflict-free IPM for testing. To achieve building a conflict-free IPM which supports the reduction of combinations in a recursive approach, following steps are required:

1) Collecting and extending *TP_Routing_Instance*s based on similarity criteria.
2) Handling conflicts.

A. Collecting and Extending TP_Routing_Instances based on Similarity Criteria

In collecting *TP_Routing_Instance*s, instances that stimulate similar behavior in the gateway are clustered into groups. Two *TP_Routing_Instance*s are said to be similar if and only if they have the same values for all related parameters such as routing parameters, network relationship parameters and mapped *TP_Routing_Instance*. Creating groups of similar *TP_Routing_Instance*s helps in reducing the number of combinations required for testing. The resulting combination from formulating groups are defined as *"reduced combinations"*.

Following example explains reduction achieved after groups are constructed.

Assume that the SUT has 4 *TP_Routing_Instance*s A, B, C and D (see Fig. 1) and it is configured to support 2 *TP_Routing_Instance*s in parallel. The number of possible combinations of 2 instances out of 4 would be 6 (the order has no effect). If *TP_Routing_Instance*s A,B and C,D are similar to each other, then groups can be constructed based on similarity criteria such that *Group*$_1$ consists of instances A and B, whereas *Group*$_2$ consists of instances C and D. After grouping, the number of combinations could be rather reduced from 6 to 3, because all other possible combinations would resemble a similar behavior, i.e., combinations of instances

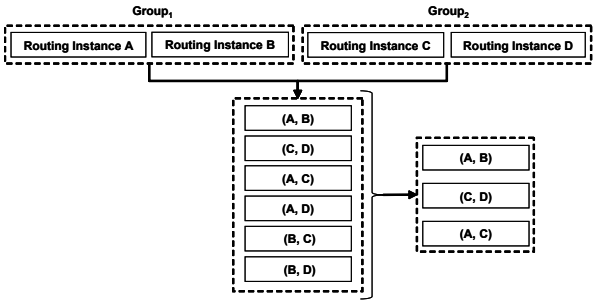

Fig. 1. Advantages of Building Similar Groups

(A, C), (A, D), (B, C) and (B, D) are all similar and can be replaced by only one substitute (A, C). Therefore (A,B), (C,D) and (A,C) are the *"reduced combinations"*.

In extending *TP_Routing_Instance*s, *Similarity Number*s and *Stress Factor*s are assigned to constructed groups. The same *Similarity Number* will be assigned to groups with *TP_Routing_Instance*s having the same routing parameters, the same *TP_Routing_Scenario* and the same characteristics for network relationship. Concerned network characteristics are the protocol type and channel bandwidth. The *Stress Factor* is calculated based on aspects such as processing time, memory usage, channel bandwidth and other channel specific aspects. *Stress Factor*s shall be assigned during test case execution. *Similarity Number*s and *Stress Factor*s are required for reducing combinations during testing. This will be discussed later in details. The basic idea to resolve the combinatorial explosion problem is by using *"reduced combinations"* formulated from constructed groups based on similarity criteria.

B. Conflict Handling

In [2], four different methods for handling conflicts in IPMs were investigated. The result of the study was that the *avoid* method is best suited with respect to size of test suite if it can be utilized. To handle conflicts with the *avoid* method; a procedure is integrated in the test case selection algorithm to prohibit choosing of conflicting combinations. Another method mentioned in the study is the *sub-models* method, in which conflicts are removed by splitting the IPM into multiple smaller conflict-free IPMs used separately to generate test cases. In this paper, a combination of these two conflict handling methods is utilized to handle the defined conflicts for testing TP parallel routing.

1) Type$_A$-Conflict Handling: Since constructed groups possess *TP_Routing_Instance*s similar to each other in all assigned parameters, *TP_Routing_Instance*s of each group will have the same attached *TP_Routing_Scenario*. To handle conflicts existing between *TP_Routing_Scenario*s, the *sub-models* method is utilized. The *TP_Routing_Scenario* is used to split IPM into sub-IPMs which are Type$_A$-Conflict-free. The resulted sub-IPMs shall have no combinable *TP_Routing_Instance*s among each other. This step can be completely achieved before executing any test case, i.e., it is not part of the recursive mechanism. Subsequently, resulted sub-IPMs are processed successively in a recursive methodology.

2) Type$_B$-Conflict Handling: As described before, a *TP_Routing_Instance* is a relationship between a *TP_Connection_Channel* and a *TP_Routing_Scenario*. In order to handle conflicts between *TP_Connection_Channel*s, the *avoid* method is utilized. To implement the *avoid* method, two reserved parameters, i.e., *"Token"* and *"Include Times"*, are attached to individual *TP_Routing_Instance*s. The value of the first parameter *"Token"* is used to decide whether a *TP_Routing_Instance* is allowed to be included in the next combination or not. The *Token* parameter can have one of the following values:

- **0:** Related *TP_Routing_Instance* having no Type$_B$-conflicts and is allowed to be included in a combination.
- **1:** Related *TP_Routing_Instance* having Type$_B$-conflicts and is allowed to be included in the next combination.
- **2:** Related *TP_Routing_Instance* having Type$_B$-conflicts and is not allowed to be included in the next combination.

The second parameter *"Include Times"* is used to hold the number of times a *TP_Routing_Instance* is included in constructed combinations. It is utilized to choose *TP_Routing_Instance*s from the same group that have not yet been included or less included than other instances. Every *TP_Routing_Instance* having Type$_B$-conflict is assigned an *"Include Times"* value "0" that gets incremeted whenever that particular *TP_Routing_Instance* becomes part of a generated combination.

In order to avoid combinations having Type$_B$-conflict, the *Rotate ()* function is called. Each time the function is called, it assigns a suitable value for parameter *Token* based on Type$_B$-conflicts and the value of parameter *Include Times*. After calling the *Rotate ()* function, Type$_B$-conflicting *TP_Routing_Instance*s with the minimum value of parameter *Include Times* will be assigned the value 1 for *Token* and all other Type$_B$-conflicting *TP_Routing_Instance*s will get the value 2. Type$_B$-conflict-free *TP_Routing_Instance*s will be assigned the value 0. The algorithm for generating the final reduced combinations will avoid *TP_Routing_Instance*s with the value 2 for *Token* parameter.

3) Type$_C$-Conflict Handling: Conflicts based on configuration parameters are constraints considered in the test case selection and generation procedure. These constraints shall be corrected if error arises, e.g., correcting the maximum number of parallel routing instances of CAN transport protocol.

IV. Reduction of Test Suite

In order to reduce the size of test suite, a new recursive test mechanism consisting of the following two test phases is proposed:

1) Testing TP parallel routing of Single Network Relationships (SNRs). An individual SNR comprises groups of *TP_Routing_Instance*s responsible for routing TP data between two specific networks of the gateway.

2) Testing TP parallel routing of Mixed Network Relationships (MNRs). An individual MNR comprises groups of *TP_Routing_Instance*s from all SNRs belonging to the same sub-IPM.

In the first test phase, testing of TP parallel routing for individual pairs of connected networks is carried out by means of groups belonging to individual SNRs. The goal of this test is to cover simple interactions (routing) between network pairs. After completing the first test phase, the second test phase is conducted. The goal of the second test phase is to cover complex interactions between multiple interacting networks by means of groups belonging to individual MNRs. Testing SNRs and MNRs separately is very useful for managing the test complexity and reducing the number of combinations. Furthermore, it provides information about possible reason for the occurrence of errors and their relationship with certain pairs of networks or parameter.

A. Coverage Criterion

Coverage criterion is an essential element of combination test strategies. It affects the complexity and the thoroughness of the test. In the proposed approach for testing TP parallel routing, coverage criteria is determined with respect to combinations of *TP_Routing_Instance*s from constructed groups. For example, *1-wise* coverage requires that at least one possible combination with maximum allowed *TP_Routing_Instance*s from every input group is included at least once in a test case. Since all *TP_Routing_Instance*s of a group stimulate a similar behavior in the gateway, any one of such a combination from each group is sufficient enough to satisfy *1-wise* coverage criterion. *N-wise* coverage criterion (exhaustive testing) requires that all possible combinations of *TP_Routing_Instance*s from *N* input groups are included in some test cases, where *N* is the number of input groups.

For an arbitrary number of input groups, the algorithm generates all possible combinations satisfying the *N-wise* interactions between the groups. The maximum number of resulting combinations is calculated as in (6):

$$X = \frac{(n+y-1)!}{y!(n-1)!} \tag{6}$$

Where *n* is the number of groups and *y* is the maximum number of allowed parallel *TP_Routing_Instance*s. Table I explains the algorithm for generating combinations with the help of an example of 4 input groups (G_1,G_2,G_3,G_4) and a maximum of 3 allowed parallel *TP_Routing_Instance*s. Resulting rows in Table. I represent the combinations and the numbers in columns represent the number of *TP_Routing_Instance*s selected from each corresponding group for test case generation. The maximum number of resulting combinations (rows) in this example can be calculated according to (6) and is equal to 20.

While generating test cases, following aspects are considered:

- To guarantee that each *TP_Routing_Instance* is included at least once in some combinations, the *Rotate ()* function is used. Upon calling this function, it generates new *Token* values for *TP_Routing_Instance*s based on the last values of *Include Times* variables and $Type_B$-Conflicts.
- The sum of column elements must be greater than or equal to the number of elements in the corresponding group in order to guarantee that each *TP_Routing_Instance* has been included at least once. If this is not the case, the remaining *TP_Routing_Instance*s from that group shall be tested individually.
- The number in each column must be lesser than or equal to the total number of *TP_Routing_Instance*s in the corresponding group. Otherwise, the combination in the related row is invalid and must be omitted.

TABLE I
COMBINATIONS FROM 4 GROUPS WITH MAXIMUM OF 3 *TP_Routing_Instance*S

G_1	G_2	G_3	G_4
3	0	0	0
2	1	0	0
2	0	1	0
2	0	0	1
1	2	0	0
1	1	1	0
1	1	0	1
1	0	2	0
1	0	1	1
1	0	0	2
0	3	0	0
0	2	1	0
0	2	0	1
0	1	2	0
0	1	1	1
0	1	0	2
0	0	3	0
0	0	2	1
0	0	1	2
0	0	0	3

B. TP Parallel Routing of SNRs

The procedure for testing TP parallel routing of SNRs is depicted in Fig. 2. The procedure accepts constructed $Type_A$-Conflict-free sub-IPMs as input and processes them successively. For each $Type_A$-Conflict-free sub-IPM, the *Selector ()* function extracts groups for the first SNR. In the next step, a combination is built for the current processed SNR with the help of the mechanism explained previously satisfying *N-wise* coverage criterion. After that, a test case will be generated and executed for the build combination. During test case generation, $Type_B$- and $Type_C$-conflicts are handled as explained previously. Result of the test case is analyzed in the following step in which *Stress Factor* is determined and configuration parameters are corrected if a variance has been observed. This procedure is repeated for all combinations until the *N-wise* coverage criteria is satisfied for each SNR. This

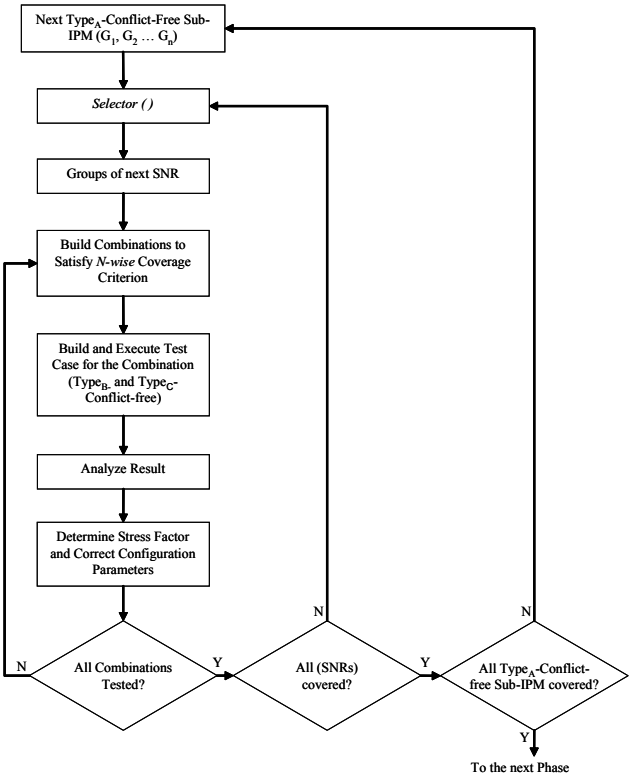

Fig. 2. TP Parallel Routing for SNRs

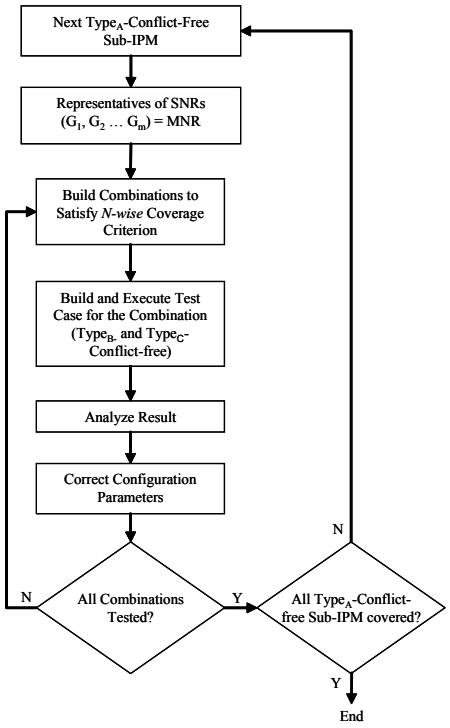

Fig. 3. TP Parallel Routing for MNRs

process is then repeated until all Type$_A$-conflict-free sub-IPM are covered. Processing combinations successively has the advantage to reduce further invalid combinations depending on the corrected configuration parameters. After finishing this phase, each group will be assigned a calculated *Stress Factor*, which will be used by the next phase.

C. TP Parallel Routing of MNRs

The procedure for testing TP parallel routing of MNRs is depicted in Fig. 3. In this procedure, Type$_A$-Conflict-free sub-IPMs are processed successively. Each Type$_A$-conflict-free sub-IPM consists of an arbitrary number of SNRs. A representative group with the best calculated *Stress Factor* for each of these SNR is selected. These representative groups are called MNR and they are the base for testing in this phase. That is, for each processed Type$_A$-Conflict-free sub-IPM, a MNR with selected groups is constructed. After selecting groups of a MNR, groups having the same *Similarity Number*s will be omitted in oder to reduce repititions in combinations. Then, the same procedure as that in previous phase is utilized to build combinations and execute test cases until *N-wise* coverage criterion is satisfied. The procedure will stop once all Type$_A$-Conflict-free sub-IPMs are processed. During recursive testing, constraints are corrected if an error is observed. This helps in further reduction of remaining combinations required to satisfy defined coverage criteria.

V. EXPERIMENT

The central gateway used in this experiment is a special electronic control unit which connects five different networks representing five functional domains of a modern vehicle. Bus systems of the five networks are listed as follows:

- A CAN network for diagnostic functional domain with 500 kilo baud, denoted as bus 1.
- A CAN network for periphery and power train functional domain with 500 kilo baud, denoted as bus 2.
- A CAN network for body functional domain with 250 kilo baud, denoted as bus 3.
- A CAN network for telematics functional domain with 500 kilo baud, denoted as bus 4.
- A FlexRay network for chassis functional domain with 10 MB, denoted as bus 5.

The gateway has 390 *TP_Connection_Channel*s which are separated in:

- 190 *TP_Connection_Channel*s used to transfer data from external device to available ECUs over the gateway.
- 190 *TP_Connection_Channel*s used to transfer data from available ECUs to external device over the gateway.
- 4 *TP_Connection_Channel*s used to transfer data between 4 couples of ECUs in the one direction.
- 4 *TP_Connection_Channel*s used to transfer data between 4 couples of ECUs in the another direction.
- 2 functional *TP_Connection_Channel*s.

Similarity criteria used to form groups are:

- Network relationship of *TP_Connection_Channel*s (source and destination networks).
- Addressing format (normal or mixed).
- Block Size (BS) and separation minimum time (STmin) if they are available.
- Cycle Repetition if it is available.
- ID Type (normal or extended).
- Address Type (physical or functional or extended).
- *TP_Routing_Scenario*

*TP_Routing_Scenario*s determined are:

- Flashing (S_1).
- Uploading (S_2).
- OBD in the one direction (S_3).
- OBD in the other direction (S_4).
- Functional routing (S_5).

Table II, table III, table IV and table V represent the resulting sub-IPMs with their corresponding groups.

Determined conflicts are listed below:

- $Type_A$-Conflicts
  - Flashing and Upload are non-combinable
  - Flashing and OBD in both directions are non-combinable
  - Upload and OBD in both directions are non-combinable
  - OBD in both directions are non-combinable
- $Type_B$-Conflicts
  - *TP_Connection_Channel*s with the same ID but different value for Node Address for Diagnose (NAD) are non-combinable
  - *TP_Connection_Channel*s with the same Slot-ID, Base-Cycle and Cycle-Repetition but different source address are non-combinable
- $Type_C$-Conflicts
  - The maximum configured parallel routing instances for CAN-CAN routing <= 30
  - The maximum configured parallel routing instances for CAN-FlexRay routing <= 8

Transport protocols implemented are based on the ISO standards, ISO 10681 for FlexRay TP [11], and ISO 15765 for CAN TP [7]. Handling $Type_B$- and $Type_C$-Conflicts during test lead to following maximum number of combinations in the first test phase:

- First sub-IPM
  - First SNR: 10 combinations
  - Second SNR: 152 combinations
  - Third SNR: 1 combination
  - Fourth SNR: 31 combinations
- Second sub-IPM
  - First SNR: 10 combinations
  - Second SNR: 78 combinations
  - Third SNR: 1 combination
  - Fourth SNR: 31 combinations
- Third sub-IPM
  - First SNR: 1 combination
  - Second SNR: 1 combination
  - Third SNR: 1 combination
  - Fourth SNR: 1 combination
- Fourth sub-IPM
  - First SNR: 1 combination
  - Second SNR: 1 combination
  - Third SNR: 1 combination
  - Fourth SNR: 1 combination

Table VI represents an example of a resulting combination from the first SNR of the first Sub-IPM. The first SNR of the first Sub-IPM consists of the groups G_1, G_2, G_3, G_4, G_{18} and G_{19} from table II. Following numbers of *TP_Routing_Instance*s have been selected to generate the first combination:

- 5 *TP_Routing_Instance*s from group G_1
- 8 *TP_Routing_Instance*s from group G_2
- 14 *TP_Routing_Instance*s from group G_3
- 3 *TP_Routing_Instance*s from group G_4

VI. Discussion

Depending on the grade of diversity in parameters for *TP_Connection_Channel*s and in connected networks; the number of formulated groups can increase. The idea is to cover the interactions between formulated groups instead of interactions between *TP_Routing_Instance*s. This helps in avoiding similar combinations and contribute to reduce the size of the test suit. If the number *N* of groups is high, either a *2-Wise* or *3-wise* coverage criteria can be used. A drawback of proposed approach is the need of system functionality expertise to define similarity criterion and calculate the Stress Factors of the groups. However this needs to be performed only once. Later on, for each new release of the system, combinations can be generated automatically.

VII. Conclusion and Future Work

In this paper, an approach for building a conflict-free IPM to test TP parallel routing on a vehicle gateway system has been proposed. Firstly, the approach utilizes defined similarity criteria to cluster system input parameters represented as *TP_Routing_Instance*s into groups stimulating similar behavior. Secondly, the two conflict handling methods *sub-models* and *avoid* are used to prohibit invalid combinations of *TP_Routing_Instance*s. In order to reduce the size of the test suite, two phases of testing have been suggested, testing TP parallel routing for SNRs in order to cover simple interactions between network pairs, and testing TP parallel routing for MNRs to cover complex interactions between multiple interacting networks. The proposed approach has been applied on a complex example of a gateway with five buses, 390 *TP_Routing_Instance*s and diverse conflicts. Generating test cases for resulting combinations from the first test phase and conducting the second test phase will be achieved in the future. Furthermore, a restbus simulation is under development to execute the generated test cases for build combinations.

TABLE II
FIRST SUB-IPM'S GROUPS

Groups	Network Relationship	Addressing Format	BS and STmin	Cycle Repetition	ID Type	Address Type	TP Routing Scenario
G_1 (19 elements)	1,2	Mixed	(32,0) (32,0)	-	Normal	Physical	S_1
G_2 (8 elements)	1,2	Normal	(32,0) (8,10)	-	Norma	Physicall	S_1
G_3 (14 elements)	1,2	Normal	(32,0) (32,0)	-	Normal	Physical	S_1
G_4 (8 elements)	1,2	Normal	(8,10) (8,10)	-	Extended	Physical	S_1
G_5 (58 elements)	1,3	Mixed	(32,0) (32,0)	-	Normal	Physical	S_1
G_6 (29 elements)	1,3	Normal	(32,0) (8,10)	-	Normal	Physical	S_1
G_7 (5 elements)	1,3	Mixed	(32,0) (32,20)	-	Normal	Physical	S_1
G_8 (3 elements)	1,3	Normal	(32,0) (32,0)	-	Normal	Physical	S_1
G_9 (8 elements)	1,4	Normal	(32,0) (32,0)	-	Normal	Physical	S_1
G_{10} (2 elements)	1,4	Normal	(4,10) (4,10)	-	Normal	Physical	S_1
G_{11} (13 elements)	1,4	Normal	(32,0) (8,10)	-	Normal	Physical	S_1
G_{12} (5 elements)	1,4	Mixed	(32,0) (32,0)	-	Normal	Physical	S_1
G_{13} (8 elements)	1,5	Normal	(32,0) (-,-)	2	Normal	Physical	S_1
G_{14} (1 elements)	1,5	Normal	(32,0) (-,-)	1	Normal	Physical	S_1
G_{15} (1 elements)	1,5	Mixed	(32,0) (-,-)	4	Normal	Physical	S_1
G_{16} (1 elements)	1,5	Normal	(8,10) (-,-)	1	Extended	Physical	S_1
G_{17} (7 elements)	1,5	Normal	(32,0) (-,-)	4	Normal	Physical	S_1
G_{18} (1 elements)	1,(2,3,4,5)	Normal	(-,-) (-,-)	-	Normal	Functional	S_5
G_{19} (1 elements)	1,(2,5)	Normal	(-,-) (-,-)	-	Extended	Functional	S_5

TABLE III
SECOND SUB-IPM'S GROUPS

Groups	Network Relationship	Addressing Format	BS and STmin	Cycle Repetition	ID Type	Address Type	TP Routing Scenario
G_1 (15 elements)	2,1	Normal	(8,0) (8,0)	-	Normal	Physical	S_2
G_2 (7 elements)	2,1	Normal	(8,10) (8,0)	-	Normal	Physical	S_2
G_3 (19 elements)	2,1	Mixed	(8,0) (8,0)	-	Normal	Physical	S_2
G_4 (8 elements)	2,1	Normal	(8,0) (8,0)	-	Extended	Physical	S_2
G_5 (63 elements)	3,1	Mixed	(8,0) (8,0)	-	Normal	Physical	S_2
G_6 (29 elements)	3,1	Normal	(8,10) (8,0)	-	Normal	Physical	S_2
G_7 (3 elements)	3,1	Normal	(8,0) (8,0)	-	Normal	Physical	S_2
G_8 (8 elements)	4,1	Normal	(8,0) (8,0)	-	Normal	Physical	S_2
G_9 (13 elements)	4,1	Normal	(8,10) (8,0)	-	Normal	Physical	S_2
G_{10} (5 elements)	4,1	Mixed	(8,0) (8,0)	-	Normal	Physical	S_2
G_{11} (2 elements)	4,1	Normal	(4,10) (4,10)	-	Normal	Physical	S_2
G_{12} (8 elements)	5,1	Normal	(-,-) (8,0)	2	Normal	Physical	S_2
G_{13} (1 elements)	5,1	Normal	(-,-) (8,0)	1	Normal	Physical	S_2
G_{14} (7 elements)	5,1	Normal	(-,-) (8,0)	4	Normal	Physical	S_2
G_{15} (1 elements)	5,1	Mixed	(-,-) (8,0)	4	Normal	Physical	S_2
G_{16} (1 elements)	5,1	Normal	(-,-) (8,0)	1	Extended	Physical	S_2
G_{17} (1 elements)	1,(2,3,4,5)	Normal	(-,-) (-,-)	-	Normal	Functional	S_5
G_{18} (1 elements)	1,(2,5)	Normal	(-,-) (-,-)	-	Extended	Functional	S_5

TABLE IV
THIRD SUB-IPM'S GROUPS

Groups	Network Relationship	Addressing Format	BS and STmin	Cycle Repetition	ID Type	Address Type	TP Routing Scenario
G_1 (3 elements)	3,4	Normal	(4,10) (4,10)	-	Normal	Physical	S_3
G_2 (1 elements)	4,5	Normal	(4,20) (-,-)	16	Normal	Physical	S_3
G_3 (1 elements)	1,(2,3,4,5)	Normal	(-,-) (-,-)	-	Normal	Functional	S_5
G_4 (1 elements)	1,(2,5)	Normal	(-,-) (-,-)	-	Extended	Functional	S_5

TABLE V
FORTH SUB-IPM'S GROUPS

Groups	Network Relationship	Addressing Format	BS and STmin	Cycle Repetition	ID Type	Address Type	TP Routing Scenario
G_1 (3 elements)	4,3	Normal	(4,10) (4,10)	-	Normal	Physical	S_4
G_2 (1 elements)	5,4	Normal	(-,-) (8,20)	16	Normal	Physical	S_4
G_3 (1 elements)	1,(2,3,4,5)	Normal	(-,-) (-,-)	-	Normal	Functional	S_5
G_4 (1 elements)	1,(2,5)	Normal	(-,-) (-,-)	-	Extended	Functional	S_5

TABLE VI
AN EXAMPLE OF A RESULTING COMBINATION

Request ID	Response ID	Network Relationship	BS and STmin	Message DLC	NAD	Addressing Format	ID Type	TP Routing Scenario
0x4e9	0x499	1,2	(32,0) (32,0)	8	14	Mixed	Normal	S_1
0x4e8	0x498	1,2	(32,0) (32,0)	8	32	Mixed	Normal	S_1
0x4c4	0x494	1,2	(32,0) (32,0)	8	5	Mixed	Normal	S_1
0x4d0	0x490	1,2	(32,0) (32,0)	8	8	Mixed	Normal	S_1
0x4e7	0x497	1,2	(32,0) (32,0)	8	13	Mixed	Normal	S_1
0x450	0x5d9	1,2	(32,0) (8,10)	8	-	Normal	Normal	S_1
0x456	0x5d5	1,2	(32,0) (8,10)	8	-	Normal	Normal	S_1
0x7e4	0x7ec	1,2	(32,0) (8,10)	8	-	Normal	Normal	S_1
0x625	0x5a5	1,2	(32,0) (8,10)	8	-	Normal	Normal	S_1
0x654	0x5d4	1,2	(32,0) (8,10)	8	-	Normal	Normal	S_1
0x652	0x5d2	1,2	(32,0) (8,10)	8	-	Normal	Normal	S_1
0x624	0x5a4	1,2	(32,0) (8,10)	8	-	Normal	Normal	S_1
0x653	0x5d3	1,2	(32,0) (8,10)	8	-	Normal	Normal	S_1
0x64e	0x5ce	1,2	(32,0) (32,0)	8	-	Normal	Normal	S_1
0x7e5	0x7ed	1,2	(32,0) (32,0)	8	-	Normal	Normal	S_1
0x7e1	0x7ea	1,2	(32,0) (32,0)	8	-	Normal	Normal	S_1
0x665	0x5e5	1,2	(32,0) (32,0)	8	-	Normal	Normal	S_1
0x778	0x788	1,2	(32,0) (32,0)	8	-	Normal	Normal	S_1
0x64d	0x5cd	1,2	(32,0) (32,0)	8	-	Normal	Normal	S_1
0x656	0x5d6	1,2	(32,0) (32,0)	8	-	Normal	Normal	S_1
0x650	0x5d0	1,2	(32,0) (32,0)	8	-	Normal	Normal	S_1
0x7e2	0x7e8	1,2	(32,0) (32,0)	8	-	Normal	Normal	S_1
0x7e6	0x7ee	1,2	(32,0) (32,0)	8	-	Normal	Normal	S_1
0x7d9	0x7e7	1,2	(32,0) (32,0)	8	-	Normal	Normal	S_1
0x7e3	0x7eb	1,2	(32,0) (32,0)	8	-	Normal	Normal	S_1
0x662	0x5e2	1,2	(32,0) (32,0)	8	-	Normal	Normal	S_1
0x65b	0x6db	1,2	(32,0) (32,0)	8	-	Normal	Normal	S_1
0x18da69f1	0x18daf169	1,2	(8,10) (8,10)	8	-	Normal	Extended	S_1
0x18da66f1	0x18daf166	1,2	(8,10) (8,10)	8	-	Normal	Extended	S_1
0x18da43f1	0x18daf143	1,2	(8,10) (8,10)	8	-	Normal	Extended	S_1

REFERENCES

[1] M. Grindal, J. Offutt, and S. F. Andler, "Combination Testing Strategies: A survey," GMU Technical Report ISE-TR-04-05, July 2004.

[2] M.N. Borazjany, L. S. Ghandehari, Y. Lei, R.N. Kacker, and D.R. Kuhn, "An Input Space Modeling Methodology for Combinatorial Testing," Software Testing, Verification and Validation Workshops (ICSTW), 2013 IEEE Sixth International Conference, pp. 372-381, Luxembourg 2013.

[3] M. Grindal and J. Offutt, "Input Parameter Modeling for Combination Strategies," In Proceeding SE'07 Proceedings of the 25th conference on IASTED International Multi-Conference: Software Engineering, pp. 255-260, USA 2007.

[4] M. Grindal, J. Offutt, and J. Mellin, "Handling Constraints in the Input Space when Using Combination Strategies for Software Testing," Technical Report HS-IKI-TR-06-001. School of Humanities and Informatics, University of Skövde 2006.

[5] S. A. Vilkomir, W. T. Swain, and J. H. Poore, "Software Input Space Modeling with Constraints among Parameters," Computer Software and Applications Conference, COMPSAC '09. 33rd Annual IEEE International, pp. 136-141, Seattle. WA. 2009.

[6] M. Grindal, J. Offutt, and J. Mellin, "Managing Conflicts when Using Combination Strategies to Test Software," Software Engineering Conference, ASWEC 2007. 18th Australian, pp. 255-264. Melbourne, Vic. 2007.

[7] "Road vehicles-Diagnostics on Controller Area Networks (CAN)-," ISO 15765:2004(E), Switzerland : ISO.

[8] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton, "The AETG System: An Approach to Testing Based on Combinatorial Design," IEEE Transaction on Software Engineering, vol. 23 ,pp. 437-444, July 1997.

[9] R. Mandl, "Orthogonal Latin Squares: An application of experiment design to compiler testing," Communications of the ACM, 28(10):1054 -1058, October 1985.

[10] C. J. Colbourn, M. B. Cohen, and R. C. Turban, "A Deterministic Density Algorithm for Pairwise Interaction Coverage," In: Proc. of the IASTED Intl. Conference on Software Engineering, pp. 345-352, Austria 2004.

[11] "Road vehicles-Communication on FlexRay-," ISO 10681:2010(E), Switzerland : ISO.