UDC 621.372

O. B. TKACHOVA, Ph.D., ABDULGHAFOOR RAED YAHYA

METHODS OF CONTRADICTION DETECT IN OPENFLOW PROTOCOL SPECIFICATION

Introduction

Information networks that are based on the concept of software-defined networking (SDN) are becoming increasingly popular today. Flexibility, relatively simple administration and update of components, open network interfaces are the main advantages of SDN. The assurance of high quality of services on-demand in SDN-based networks provides by separating data transmitting roles, improved management mechanisms and strict compliance with requirements for medium. One of factors that affect to the quality of provided services is the efficient exchange of management information between control layer and data plane layer. OpenFlow is the basic protocol that used to control messages exchange between these layers [1].

A rapid development and increasing of services range leads to establish of additional requirements to operation of SDN-based network components. This lead to a permanent modification of OpenFlow protocol, expanding and updating of protocol specification requirements. Adding or updating of specifications requirements in some cases gives the rise of contradictions between them. For example, applying different commands that perform the same actions, different sequences of message processing can be the cause of contradiction in requirements. The lack of strict systematization and formalization of specification requirements, non-formal descriptive of protocol behavior are the reason of complexity in contradictions identification.

The formal methods using in processes of formalization requirements of protocol specifications are giving ability to uniquely specify a set of possible events and processes and describe the causal relationships between them. The interpretation accuracy of protocol requirements greatly depends of expressive power of the method used for formalization.

The methods of graphical and mathematical notation such as UML [2], SDL [3], Backus-Naur form, LOTOS [4], temporal logics [5] are widespread using in formalization process of information systems specification. However, the application of such methods for OpenFlow protocol specification formalization is difficult. The absence of possibility to take into attention timestamps fixation for distributed resources and lack of processes prioritization leads to impossibility of the consistency checking. Thus, there is a need to develop a new method of formal consistency check that allows taking into account timing and processes priority during the protocol operation.

Analysis of the literature [2-6] shows that the apparatus of the algebra of communication of distributed resources (ACSR) enables to formalize the causal relationships between events and processes that take place in the operation of the protocol OpenFlow. The ACSR has a graphical and mathematical interpretation; the analytic features of ACSR can be increase due to this.

Consistency check of protocol OpenFlow requirements is an important task. The solution of this task will help to reduce the number of errors that occur in the case of its operation, and, therefore, ensure the quality of provided services. The solution of the consistency check task can also be implemented to find contradictions in the different versions of the OpenFlow protocol specification.

In the paper a method of the consistency check that based on step-by-step comparison of the protocol chronological sequence are suggested. The logical connectivity “precondition – events – postcondition” is also take into account. The suggested method can detects statements in which there are contradictions between the requirements of the specification. Addition to this method can serve as a check on the consistency of the graph of possible OpenFlow protocol states in accordance with the requirements of the specification. An analysis of the state graph cannot only establish the existence of contradictions and to form a sequence of actions that leading to the emergence of contradictions.

1. Overview of Algebra of Communicating Shared Resources functionality

Algebra of Communicating Shared Resources (ACSR) is a process algebra designed for the formal specification and manipulation of distributed systems with resources and real-time constraints. Application of ACSR as a method of formalizing requirements allows proving the requirements truth and finding inconsistencies in between specification requirements.

In general, the grammar of ACSR can be represented as follows [6]:

$$ACSR = < Proc(X, p), Act, O, C >, \quad (1)$$

where $Proc(X, p)$ a set of all processes that can take part in OpenFlow specification. The process may be defined by a set of variables and their relative priority: $P(x_1, p_1), Q(x_2, p_2), R(x_3, p_3), S(x_4, p_4)$. In this case, variables x_1, x_2, x_3, x_4 determine the process and elements p_1, p_2, p_3, p_4 set the priority of the process; Act is a set of all events; O is a set of operators; C is logical connectives.

The ACSR predicates form the alphabet, which contains all the atomic approval protocol specification. This atomic statements are divided into two groups: processes (*Proc*): $\{P, Q, R, S\} \in Proc$, the processes characterize the performance of an action, for example, search for entries in the forwarding table, and events or actions (*Act*): $\{\alpha, \beta\} \in Act$, the events characterize the result of a specific process, for example, the message is generated..

ACSR comprises a syntaxes and a semantics component [].The behavior of a process is given by a labelled transition system, which is a subset of $Proc \times Act \times Proc$. In common, the structure of any OpenFlow protocol specification statements can be defining by the following sequence of transitions:

$$Act_{in} \longrightarrow Proc \xrightarrow{Act} Proc' \xrightarrow{Act'} Proc'' \xrightarrow{Act''} Act_{fin},$$

where Act_{in} is the initial action, which initiates execution of subsequent processes, Act_{fin} –is the final action, $\{Proc\}$ is the sequence of processes that occur in the operation protocol.

1.1. A methods of contradictions detect that based on phased comparing the chronological sequence of protocol states

The suggested method is basing on finding all specification formalisms that containing the statement for consistency checking.

The method of consistency checking comprises the following steps:

1.The definition of all atomic statements that take place in specification requirements forming.

Let $M(S)$ is a set of atomic statements for some ASCR formalism $F_r(t_0)$. This formalism contains a specification requirement (t_0) that must be checked on contradiction. The predicates $Proc$ and Act, $Proc \in S, Act \in S$ that forming requirement can have different nesting levels. The nesting characterized by the depth of occurrence of the events and processes that make this claim, r. The sequence ordering $(\{Proc\} \& \{Act\})_r$ determines the depth of requirement occurrence.

2. Search the specification requirements that contain a checking statement.

Let $M(S,n)$ the set of all finding formalism that contain the checking statement $F(t)$, $M(S,n) \equiv \Sigma F_r(t)$. In this case, the set $\Sigma F_r(t)$ contains all possible protocol specification requirements that containing the checking statement. Each finding statement can have different nesting level r within the containing formalism $F(t)$.

3. The foundation a logical connectivity "precondition – events – postcondition" for all finding statements.

A formula $A(p)$ is creating for all finding formalisms $F(t)$ that contain the finding statements, $t \in M(S,n)$. $A(p)$ defines direct processes following (after action). $A(p)$ indicates that every states $P: p_0, p_1, p_2, ... p_r$, $P \in Proc \in S$, has a following $2^{|S|}$ states $Q: q_1, q_2, q_3, ... p_r$, $A \in Act \in S$, where $\{q\} \in S, \{p\} \in S$. The logical connectivity between $q \to p$ determine in [6, 7].

A formula $B(p)$ is creating for all finding formalisms $F(t)$ that contain the finding statements, $t \in M(S,n)$. $B(p)$ defines direct processes precedence (before action). $B(p)$ is an inverse to $A(p)$. Evident that states $Q(B): p_1, p_2, p_3, ... p_r$ directly precede the states $Q(A): q_1, q_2, q_3, ... q_r$.

4. The check of consistency of finding statements.

It is assumes that the specification is considered checked statement true: all the action sequences, which are its components, are true. Let found the first statement $F(t_1)$, such that $S_1 \subseteq \sum(S,n)$. Define the boundaries of the statements $M(S,n)$, $A(F_r(t_1))$ and $B(F_r(t_1))$. Where $B(F_r(t)) \xrightarrow{log.link} A(F_r(t))$. Thus, the formation of the total formalism, $\Sigma F_r(t) | t_1 = t_0$ is possible if and only if for all pre- and post-conditions included in $M(S,n)$ and form a logical sequence, definite statement relevant specifications for each elementary conjunction uniform depth r does not result 0. The consistency checking implemented for all formalisms, which contains the desired statements $\sum_{i=0}^{n} F_r(i) \neq 0$. If the result that having a value 0 is found at step r, it means that step p_{r-1} contain inconsistency.

1.2. A methods of contradictions detect of specification requirements that based on analysis of protocol graph

Search and contradictions identification can also done by analysis of reachability tree for graph of protocol states.

$G(S_{r_{max}})$ is a bipartite directed graph of protocol states:

$$G(S_{r_{max}}) = \langle C, T \rangle, \tag{2}$$

where C is a set of vertices of graph, $C \in \{Act\}$ and T is a set of arcs, $T \in \{Proc\}$.

The suggested method comprises the following steps:

1. The vertices set formation.

Let $M(C_0)$ a set of vertices that contains all the possible states of the specification formalisms. Moreover are all the possible requirements of the specification, containing the statement and form a set $M(C_n)$, where n the number of finding statements.

2. Formation of the set transitions.

3. Definition of the transitions sequence that lead to the final state.

As a final state selected last state of checking formalism $G_0(S_{r_{max}})$. The set of active vertices of the graph, leading to the final state $G'(S_{r_{max}})$, It forms the core states of the graph, which includes all the specifications found formalisms $\sum G(S_{r_{max}})$. The following statement feasible at the same time: if $G'(S_{r_{max}})$ the core of the set of possible states of the graph $G(S_{r\,max}) \equiv \sum F_r(t))$,

then for each chain of states $q,q_1,q_2,...q'_r$ from $G'(S_{r_{max}})$ exists subgraph $G(S_\mu)$, where feasible formula $F_r(\mu)$, that contain the chain coinciding with the chain of states $q,q_1,q_2,...q'_r$

5.The check of finding requirements contradiction.

Let there are two subsets $F'(t)$ and $F''(t)$ that belong to $M(F(t))$, where $B(F(t))=F'(t)$ and $A(F(t))=F''(t)$, the $A(F(t))$ defines direct processes following (after action). $B(p)$ defines direct processes precedence (before action).

Search and consistency check is carried out by constructing a tree reachable [] to set the states of the graph that belong to $F'(t)$ and $F''(t)$ consiquently.

The forming $F'(t)$ can be represented as $Q_0 = B(F_0(t))$, $Q_1 = Q_0 \cap B(F_0(t))$, …, $Q_{i+1} = Q_i \cap B(F_i(t))$. The forming $F''(t)$ can be represented as $Q_1 = Q_0 \cap A(F_0(t))$, …, $Q_{i+1} = Q_i \cap A(F_i(t))$.

The existing only one active label in the states of graph is a prerequisite for the consistency check. If the contradiction is not found, it should meet the following statement. Each state from $F_r'(t)$ is reachable. This implies that $F_r'(q) \neq 0$ and kernel of $F_r'(t)$ is reachable for each state.

Thus, the formalism of OpenFlow protocol specification does not contain contradictions, if and only if when there is a finite set of transitions $F(t) \in \Sigma F_r(t)$ leads to final state. Nesting level of each position corresponds to r. If the graph is cyclic, then none of the cycles must not contain empty subcycle.

2.The research result

Analysis of the various versions of the OpenFlow protocol specifications showed that the contradictions within the same specifications often occur in the process of modifying or removing the forwarding tables. The contradiction in the requirements that consist processes of automatically delete table records is the reason of route loss and additional time spent on its recovery.

Flow table modification messages can have the following types [8]:

```
{ OFPFC_ADD, /* Add new flow */
OFPFC_MODIFY, /* Modify all input flows*/
OFPFC_MODIFY_STRICT, /* Modify the FlowTable entries that
strictly corresponding to the standard and priority.*/
OFPFC_DELETE, /* Delete all input flow */
OFPFC_DELETE_STRICT /* Delete the FlowTable entries that
strictly corresponding to the standard and priority.*/ };
```

The lack of strict definitions of functions for listed above messages is a significant disadvantage. Without STRICT appended, the wildcards are active and all flows that match the description are modified or removed.

For non-strict MODIFY and DELETE commands that contain wildcards, a match will occur when a flow entry exactly matches or is more specific than the description in the flow mod command. For example, if a DELETE command says to delete all flows with a destination port of 80, then a flow entry that is all wildcards will not be deleted.

MODIFY and DELETE commands formalized by ACSR can be represent as follows:

$$\begin{aligned} &OFPFC\_DELETE : Receive(OFPFC\_DELETE,1).Find(T_i,entry_i,eng\_port\_80) \xrightarrow{v.entry_i = v.eng\_port\_80} \\ &Delete(entry_i,eng\_port\_80) \xrightarrow{v.entry_j = v.eng\_port\_80} Delete(entry_j,eng\_port\_80)... \xrightarrow{v.entryn \neq v.eng\_port\_80} \\ &Wait(OFPFC\_DELETE,2) \end{aligned} \quad (3)$$

This statement can be has nested level r, $r_{max} = m*n$, where m is a forwarding table amount, n is amounts of entries in each forwarding table.

However, a DELETE command that is all wildcards will delete an entry that matches all port 80 traffic. This same interpretation of mixed wildcard and exact header fields also applies to individual and aggregate flows stats [8].

This statement can be represent as follows:

$$\begin{aligned} &OFPFC\_DELETE : Receive(OFPFC\_DELETE,1).Find(T_i,entry_i,eng\_port\_80) \xrightarrow{v.entry_i=v.eng\_port\_80} \\ &Delete(entry_i,eng\_port\_80) \xrightarrow{v.entry_j=v.eng\_port\_80} Delete(entry_j,eng\_port\_80)... \xrightarrow{v.entryn\neq v.eng\_port\_80} \cdot \\ &Wait(OFPFC\_DELETE,2) \end{aligned} \quad (4)$$

Let (3) is true, then all of its components takes the logical value “1”. Formed a set $A(F(t))$ and $B(F(t))$ processes that lead to their changing is established:

$$M'(t) : Receive(OFPFC\_DELETE,1) \xrightarrow{v.entry_i=v.eng\_port\_80} Delete(entry_j,eng\_port\_80),$$

$$Find(T_i,entry_i,eng\_port\_80) \xrightarrow{v.entry_i=v.eng\_port\_80} Delete(entry_j,eng\_port\_80).$$

$$M''(t)) : Receive(OFPFC\_DELETE,1) \xrightarrow{v.entry_i=v.wildcard} Delete(entry_j,*80),$$

$$Find(T_i,entry_i,wildcard,*80) \xrightarrow{v.entry_i=v.wildcard} Delete(entry_j,*80).$$

When the consistency check revealed that one string of the set $F(t)$ receive the result “0”. That demonstrates the appearance of contradiction in the finding requirements.

To construct a state graph protocol that make up a definite statement, and the next the consistency check of the method of construction of the state graph is necessary to generate two sets $F'(t)$ and $F''(t)$:

$$\begin{aligned} &F'(t) : Receive(OFPFC\_DELETE,1) \cup \\ &Find(T_i,entry_i,eng\_port\_80)(Receive(OFPFC\_DELETE,1)) \cup \\ &\cup(delete(entry_i,eng\_port\_80))_r(Find(T_i,entry_i,eng\_port\_80)_r(Receive(OFPFC\_DELETE,1))) \cup \\ &\cup Wait(OFPFC\_DELETE,2)((delete(entry_i,eng\_port\_80))_r(Find(T_i,entry_i,eng\_port\_80) \\ &(Receive(OFPFC\_DELETE,1)))) \end{aligned}$$

$$\begin{aligned} &F''(t) : Receive(OFPFC\_DELETE,1) \cup \\ &Find(T_i,entry_i,wildcard,*80)(Receive(OFPFC\_DELETE,1)) \cup \\ &\cup(delete(entry_i,*80))_r(Find(T_i,entry_i,wildcard,*80)_r(Receive(OFPFC\_DELETE,1))) \cup \\ &\cup Wait(OFPFC\_DELETE,2)((delete(entry_i,*80))_r \\ &(Find(T_i,entry_i,wildcard,*80)(Receive(OFPFC\_DELETE,1)))). \end{aligned}$$

In this case, the graph $G'(S_{r_{max}})$ involves the sets $F'(t)$ and $F''(t)$ combining. The set combinations are considered as a core of the set of statements involved in protocol specifications (Fig. 1).

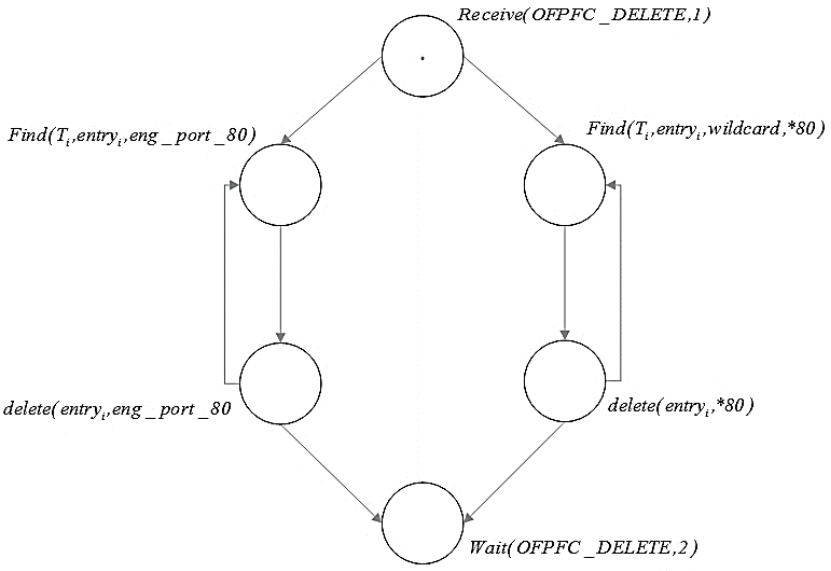

Fig. 1 Graf of protocol states

Reachability tree analysis of given graph $G'(S_{r_{max}})$ is shows that a finite set of transitions $F(t) \in \Sigma F_r(t)$, coming to a final state, does not exist. Thus, the statements of specification contain a contradiction.

Conclusion

Contradictions in requirements of OpenFlow protocol specification may occur in the process of its modifications or additions. The contradictions occurrence caused by absence of formalization and systematization of the process of formalization of the specification. The task of contradictions finding is greatly complicated because the most of protocol specifications are set on a subset of natural language.

Application of different methods of specification requirements formalization give ability to avoid or eliminate the contradictions appearance. The algebra of communication shared resources suggested as a tool for formalizing of protocol specification requirements. Its expressive power allows to describe the possible behavior of the protocol.

The suggested method of contradictions finding based on the formation of the set of atomic statements of specification by ACSR formalisms and phase comparison of the chronological sequence of their occurrence. A method that is based on the construction and analysis of the states protocol graph also suggested in the paper. This method allows to generate a sequence of actions that lead to contradictions.

List of references 1. *Software-Defined* Networking: The New Norm for Networks // Open Networking Foundation, 2012. Available at: https://www.opennetworking. org/images/stories/ downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf 2. *James Rumbaugh, Michael Blaha,* William Premerlani,Frederick Eddy, William Lorensen: “Object-Oriented Modeling And Design”, Prentice Hall, New York, 1998 3. *ITU-T* recommendations. Series Z. Languages and general software aspects for telecommunication systems, Geneva, 2000, 246 p. 4. *Reference* Manual of the LNT to LOTOS Translator. Version 6.3. IRNIA, 2015 – 137. 5 *A. Pnueli*. The Temproal Logic of Programs. In Proc. of Foundations of Computer Science, p. 46-57 6. *Patrice* Brkmond-Grkgoire, Insup Lee A Process Algebra of Communicating Shared Resources with Dense Time and Priorities University of Pennsylvania, Philadelphia, PA, 1994 – 49 p. 7. *G. Engels, J. M. K¨uster, L. Groenewegen, and R. Heckel.* Amethodology for specifying and analyzing consistency of object-oriented behavioral models. In V. Gruhn, editor, Proceedings of the 8th European Software Engineering Conference (ESEC), pages 186–195. ACM Press, 2001. 8. *OpenFlow* Switch Specification (Series) [Electronic resource] // Open Networking Foundation., 2014. Available at: https://www.opennetworking.org/sdn-resources/onf-specifications/openflow

Харьковский национальный
университет радиоэлектроники

Поступила в редколлегию 27.11.2015