Implementing a virtual network on the SDN data plane

Igor Burdonov
Software Engineering department
Ivannikov Institute for System Programming of RAS
Moscow, Russia
igor@ispras.ru

Nina Yevtushenko
Software engineering deparment
Ivannikov Institute for System Programming
Moscow, Russia
evtushenko@ispras.ru

Alexandr Kossachev
Software Engineering department
Ivannikov Institute for System Programming of RAS
Moscow, Russia
kos@ispras.ru

***Abstract*—The paper investigates the implementation of virtual networks on the SDN data plane, modeled by a graph of physical connections between network nodes. A virtual network is defined as a set of ordered host pairs (sender, receiver), and it is implemented by a set of host-host paths that uniquely determine the switch settings. It is shown that any set of host pairs can be implemented on a connected graph without the occurrence of an infinite transfer of packets in a loop and without duplicate paths when the host receives the same packet several times. However, undesired paths may occur when a host receives a packet that is not intended for this host. On the other hand, it is shown that in some cases, implementation without undesired paths inevitably leads to duplication or looping of packets. The question is posed: on which graph can any set of host pairs be implemented without looping, duplication and undesired paths? A sufficient condition is proposed and a hypothesis is put that this condition is also the necessary condition.**

Keywords— Software Defined Networking (SDN), data plane, Network connectivity topology

I. Introduction

Software Defined Networking (SDN) is one of the main technologies for network virtualization [1][2][3][4][5]. On the data plane, packets are transmitted between hosts through intermediate switches. This is modeled by a graph of physical connections (links) often referred to as RNCT (Resource network connectivity topology), the vertices of which are hosts and switches, and the edges correspond to physical connections between them. The switches are configured by SDN controller (-s), setting up a set of flow rules for each switch. The rule determines which neighboring vertices of the graph the packet received by the switch is forwarded to, depending on which neighbor the packet came from and the parameter vector in the packet header [6]. Thus, the configuration of the network switches determines the set of paths from host to host, through which packets will be forwarded.

There are tasks of two levels. 1) How to implement a given set of host-host paths through appropriate switch settings? 2) How to implement a given set of pairs (host, host) through appropriate host-host paths in the graph of physical connections?

It is known that when solving the 1st problem there are three effects. 1a) Cycles may occur in which packets will be transmitted endlessly and, moreover, endlessly cloned. 1b) Undesired paths may appear. These problems were investigated in [4] [5]. 1c) Duplicate paths may appear, due to which the host destination receives the same packet more than once.

The 2nd task is reduced to the 1st task by choosing a suitable set of paths, avoiding the above effects if possible. The following questions are then raised. 2a) Is it possible to implement a given set of host pairs on a graph, i.e. to choose a suitable set of paths without the indicated effects? 2b) Is it possible to implement any set of host pairs without such effects on a given graph of physical connections?

As an answer to question 2a), this paper shows that any set of host pairs can be implemented on a connected graph without cycles and duplication, however, undesired paths may occur, i.e. paths connecting unintended host pairs. On the other hand, the article demonstrates that in some cases, the implementation of a given set of host pairs without undesired paths inevitably leads to path duplication or loops. The article establishes a sufficient condition for the positive answer to question 2b) and puts the hypothesis that this sufficient condition is also the necessary condition.

II. Preliminaries

A physical connection graph (hereinafter simply a graph), often referred to as RNCT (Resource network connectivity topology), is a connected undirected graph $G = \{V, E\}$ without multiple edges and loops, where V is the set of switches and hosts, $E \subseteq V \times V$ is the set of edges modeling physical connections between the switches and hosts. Since the edge connecting the vertices a and b is undirected and there are no multiple edges, it can be denoted by both ab and ba. Since there are no loops, there are no edges of the form aa in E. Since there are no multiple edges, a path as a sequence of adjacent edges is uniquely determined by the sequence of vertices $a_1 \ldots a_n$ through which it passes. A path starting at vertex a and ending at vertex b is called an ab-path. If the path passes along the edge ab from a to b, then we say that it passes through the arc ab. If a and b are hosts, an ab-path in which all vertices except the first vertex a and the last vertex b are switches, is called a *complete path*. A

path is called *vertex-simple* (*edge-simple*) if each vertex (arc) occurs at most once. The vertices of the graph will be denoted by lowercase letters *a*, *b*, *c*,…*x*, *y*, *z*, the paths by bold lowercase letters ***p***, ***q***, ***r***,…, and the set of paths by capital letters *P*, *Q*, *R*, …

We will assume that each host *x* is connected to exactly one switch [3]. Therefore, the host is the terminal vertex of the graph, i.e. a vertex of degree 1. If the switch *a* has degree 1 and is connected to vertex *b*, then any complete path passing through *a* has the form ...*bab*...; removing all *bab* cycles from it, we get a path that does not pass through *a*. This means that such a switch is "superfluous", and it is enough to consider graphs in which terminal vertices are only hosts. The sets of hosts and switches are denoted by *H* and *S*, respectively; $H \cup S = V$, $H \cap S = \varnothing$.

In general case, the rule of the switch *b* has the form σabc, where *a* and *c* are neighbors of *b*, and σ is the vector of packet header parameters that can be used in the rules. Such a rule means that switch *b*, having received a packet with vector σ from neighbor *a*, forwards it to neighbor *c*. It is assumed that the switch does not change σ. Thus, for the vector σ complete paths of the form $a_1 \ldots a_n$ are considered where in the switch a_i there is a rule $\sigma a_{i-1} a_i a_{i+1}$, $i = 2..n-1$. If there are two rules σabc and $\sigma abc`$, where $c \neq c`$, then it is said that the packet is cloned, i.e. is sent to both neighbors *c* and *c`*.

The given set *P* of complete paths uniquely determines the minimal set of switch rules that induces all paths from *P*. However, this does not mean that only paths of *P* are induced. We say that two paths are *merging* paths on the arc *ab* at vertex *a* if they have an intermediate common arc *ab* with different direct predecessor arcs *ca* and *c`a* where $c \neq c`$, and are *separating paths* after the arc *de* at the vertex *e* if they have an intermediate common arc *de* with different direct successor arcs *eg* and *eg`* where $g \neq g`$.

There is a cycle in the path if a complete *xy*-path passes through an arc twice, i.e. the path has the form $\boldsymbol{p}a\boldsymbol{q}\boldsymbol{e}\boldsymbol{r}(a\boldsymbol{q}\boldsymbol{e}\boldsymbol{r})^{*}a\boldsymbol{q}\boldsymbol{e}\boldsymbol{s}$, where the segment ***p*** starts at the host $x \neq a$, the segments ***p*** and ***r*** do not end at the same vertex, after these segments the switch *a* follows, the segment ***aqer*** passes one or more times, after the switch *e* segments ***r*** and ***s*** do not start at one vertex, and segment ***s*** ends at the host $y \neq e$. Moving along the path, we see that at the vertex *a* the path merges with itself, then at the vertex *e* it separates with itself, and then this merging and separating occurs again. If the path goes through the cycle *k* times, then it will be $k + 1$ times both separating after merging, and merging after separating. Packets will not only endlessly traverse the ***aqer*** cycle, but also endlessly clone at vertex *e*, so host *y* will receive an infinite number of clones of the same packet.

A path that does not merge with itself is an edge-simple path. For the absence of cycles, it is necessary that all paths of the set *P* be edge-simple. But that is not sufficient. If two edge-simple complete paths from *P* after merging on the arc *ab* are separated (after this or another arc), i.e. those are $x\boldsymbol{p}ab\boldsymbol{q}y$ and $x`\boldsymbol{p}`ab\boldsymbol{q}`y`$ with different start and end hosts $x \neq x`$ and $y \neq y`$, then new paths $x\boldsymbol{p}ab\boldsymbol{q}`y`$ and $x`\boldsymbol{p}`ab\boldsymbol{q}y$ are also induced. This operation of inducing new paths is called the *arc closure*, and the result of the arc closure of all pairs of paths from *P* is denoted by $P{\downarrow}{\uparrow}$ [4] [5]. Obviously, $P \subseteq P{\downarrow}{\uparrow}$. If $P \neq P{\downarrow}{\uparrow}$, i.e. *P* is not arc closed, then undesired paths occur. In particular, non-edge-simple paths and, therefore, cycles may occur. The appearance of cycles in the arc closure of the set of complete paths always indicates the infinity of this arc closure and, thus, the presence of duplication. There are no cycles in a finite arc closed set of complete edge-simple paths.

For a set of complete paths *P*, by $H(P) \subseteq H \times H$ we denote the set of pairs *xy* for which there is an *xy*-path in *P*. A set of host pairs $D \subseteq H \times H$ that does not contain pairs of the form *xx* will be called *normal*. We say that a normal set *D* (*non-strictly*) is implemented by an arc closed set of complete paths *P* if $D \subseteq H(P)$, *D* is *strictly* implemented if $D = H(P)$, *D* is implemented *without cycles* if *P* is finite, *D* is implemented *without duplication* if *P* contains exactly one *xy*-path for each pair $xy \in D$.

If the source address is included into the parameter vector σ, then the rules for parameter vectors with different source addresses work independently. For each source host *x* in the graph, the tree I_x of shortest paths leading from *x* to all other hosts can be selected. For any normal set *D* of host pairs and any host *x*, a subset D_x is selected, where the first element of the pair is host *x*, and the subtree $I_x(D)$ is selected, in which leaf vertices are destination hosts *y* such that $xy \in D_x$. In the outgoing tree, all paths are edge-simple (even vertex-simple, that is, without vertex repetition), and there is no merging, thereby there is no separating after merging. Therefore, $I_x(D)$ is arc closed and, obviously, strictly implements D_x without cycles and duplication; moreover, the shortest complete paths are used. Thus, in this case there is no problem with the strict implementation without loops and duplication of any normal set of host pairs. Moreover, the implementation of any such set turns out to be a subset of the same set of paths, namely, the union of I_x trees over all source hosts *x*. A similar procedure with a similar result is applicable when the destination address is included into the parameter vector σ. In this case, the incoming O_x tree is built for each destination host *x*.

Below we consider the case when the source address and the destination address are not included into the parameter vector σ. The remaining parameters do not affect packet transmission with the given vector σ, so we will omit σ in the designation of the rule and write *abc* instead of σabc. In other words, the switch rules (for a given vector σ) determine to which vertex the packet should be sent, only depending on the neighbor from which the packet was received. In this case, the maximum number of rules by which the switch operates depends only on the number of its neighbors and does not depend on the number of hosts in the network.

III. Non-strict / strict implementation of versus cycles and duplication

In this section, we examine the relationship between the non-strict and strict implementation of the set of host pairs with the presence or absence of cycles and duplication.

Proposition 1. On a connected graph *G*, any normal set *D* of host pairs is non-strictly implemented without cycles and duplication.

Proof. In G, choose an arbitrary spanning tree T. Since a host has degree 1 in G, all the hosts are leaves of T. Let P be the set of all shortest complete paths in the tree T. Obviously, all paths from P are vertex-simple and, therefore, edge-simple; moreover, there are no duplicate paths and P is finite and arc closed. If we leave only xy-paths in the set P such that $xy \in D$, then for the resulting set $P(D)$ we have $P(D)\downarrow\uparrow \subseteq P$. Thus, in $P(D)\downarrow\uparrow$ all the paths are also edge-simple, there are no duplicate paths and $P(D)\downarrow\uparrow$ is finite and arc closed. By construction, $D = H(P(D)) \subseteq H(P(D)\downarrow\uparrow)$. Therefore, $P(D)\downarrow\uparrow$ non-strictly implements D without cycles and duplication.

□

For a complete path $\boldsymbol{p}$, let $\boldsymbol{p}^\circ$ denote the path that is obtained from $\boldsymbol{p}$ by using, as far as possible, the following operation to delete cycles: the path $\boldsymbol{p} = \boldsymbol{qaras}$ turns into the path $\boldsymbol{p}^\circ = \boldsymbol{qas}$. Note that the result of the operation "°", generally speaking, is ambiguous. For the sake of simplicity, we assume that the operation of deleting a cycle is applied only when each vertex of the prefix $\boldsymbol{q}$ occurs only once in $\boldsymbol{p}$ and $\boldsymbol{q}$ and $\boldsymbol{r}$ do not contain vertex a. In other words, if the vertex a occurs first among the vertices that have several occurrences in $\boldsymbol{p}$, then a cycle is removed by deleting $\boldsymbol{ra}$ in $\boldsymbol{p}$, where $\boldsymbol{r}$ does not contain $\boldsymbol{a}$. For example, for $\boldsymbol{p} = xacbcaby$, $\boldsymbol{p}^\circ = xaby$ (not $xacby$) is obtained. The procedure ° terminates when each vertex occurs at most once in $\boldsymbol{p}^\circ$. Given a set of complete paths P, P° is the set of paths obtained by deleting cycles from all paths P, i.e. $P^\circ = \{ \boldsymbol{p}^\circ \mid \boldsymbol{p} \in P \}$.

Proposition 2. Let P be the set of complete paths. Then P° consists of vertex-simple paths and connects the same pairs of hosts that the set P: $H(P^\circ) = H(P)$. If P is arc closed, then the arc closure $P^\circ\downarrow\uparrow$ connects the same pairs of hosts: $H(P^\circ\downarrow\uparrow) = H(P^\circ)$, i.e. the arc closure $P^\circ\downarrow\uparrow$ adds only duplicate paths to the set P°. If P is finite and arc closed, then P° is finite and arc closed.

Proof. Obviously, removing all cycles from a path makes the path vertex-simple. Therefore, P° consists of vertex-simple paths. The operation of deleting one cycle from one path does not change $H(P)$. Therefore, the chain of such operations also does not change $H(P)$. Therefore, P° connects the same host pairs that the set P: $H(P^\circ) = H(P)$.

Let us prove that if the set P is arc closed, then $H(P^\circ\downarrow\uparrow) = H(P^\circ)$. Indeed, let the set P° contain xy-path $\boldsymbol{p}$ and $x`y`$-path $\boldsymbol{q}$, which are obtained by removing cycles from xy-path $\boldsymbol{p}`$ and $x`y`$-path $\boldsymbol{q}`$, respectively, which are elements of the set P. If the path $\boldsymbol{p}$ and $\boldsymbol{q}$ have a common arc, then this arc is also common for the paths $\boldsymbol{p}`$ and $\boldsymbol{q}`$. Since P is arc closed, it also contains an $xy`$-path and an $x`y$-path. Thus, after deleting the cycles in P° there will also be an $xy`$-path and an $x`y$-path. Therefore, $H(P^\circ\downarrow\uparrow) = H(P`)$.

Let P be finite and arc closed. Then, obviously, P° is also finite. Let us prove that P° is arc closed. Let P° contain paths $\boldsymbol{pabq}$ and $\boldsymbol{p_1ab q_1}$ with a common arc. These paths are obtained by deleting the cycles from the paths $\boldsymbol{r}$ and $\boldsymbol{r_1}$, respectively, which are elements of the set P. Since, when deleting the cycles, any nonempty sequence between any two occurrences of the vertices a and b can be completely deleted only together with the removal of the occurrence a and/or b, the paths $\boldsymbol{r}$ and $\boldsymbol{r_1}$ can be represented as $\boldsymbol{p}`\boldsymbol{ab}\boldsymbol{q}`$ and $\boldsymbol{p}`_1\boldsymbol{ab}\boldsymbol{q}`_1$, respectively, where $\boldsymbol{p} = \boldsymbol{p}`^\circ$, $\boldsymbol{p_1} = \boldsymbol{p}`_1{}^\circ$, $\boldsymbol{q} = \boldsymbol{q}`^\circ$, $\boldsymbol{q_1} = \boldsymbol{q}`_1{}^\circ$. Since P is arc closed, P contains the paths $\boldsymbol{p}`\boldsymbol{ab}\boldsymbol{q_1}`$ and $\boldsymbol{p}`_1\boldsymbol{ab}\boldsymbol{q}`$. And then P° contains the paths $\boldsymbol{pabq_1} = \boldsymbol{p}`^\circ\boldsymbol{ab}\boldsymbol{q_1}`^\circ = (\boldsymbol{p}`\boldsymbol{ab}\boldsymbol{q_1}`)^\circ$ and $\boldsymbol{p_1abq} = \boldsymbol{p}`_1{}^\circ\boldsymbol{ab}\boldsymbol{q}`^\circ = (\boldsymbol{p}`_1\boldsymbol{ab}\boldsymbol{q}`)^\circ$. Therefore, the set P° is arc closed.

□

Note that all conditions on the set P of complete paths in Proposition 2 are necessary conditions. If the set P is not arc closed, then the arc closure $P^\circ\downarrow\uparrow$ can connect additional pairs of hosts: $H(P^\circ\downarrow\uparrow) \supseteq H(P^\circ)$, i.e. arc closure $P^\circ\downarrow\uparrow$ can add to the set P° not only duplicate paths. If the set P is finite, but not arc closed, then the set P° is finite, but not necessary is arc closed. In both cases, the set $P = \{ xaby, x`aby` \}$ can serve as an example, where $x \neq x`$ and $y \neq y`$: $P^\circ = P$, $P^\circ\downarrow\uparrow = \{ xaby, x`aby`, xaby`, x`aby \}$. If the set P is arc closed, but infinite, then the set P° not necessary is arc closed. As an example the set $P = Q\downarrow\uparrow$ can be considered where $Q = \{ xabcdy, x`cdaby` \}$, $x \neq x`$ and $y \neq y`$: the set P contains the path $xabcdaby`$ that is not edge-simple, $P^\circ = Q$ and $P^\circ\downarrow\uparrow = P$.

It follows from Proposition 2 that any set D of host pairs that is strictly implemented without cycles can be strictly implemented by a set of vertex-simple paths. It is sufficient to take the set P° of vertex-simple paths instead of a finite arc closed set P of complete paths that strictly implements D.

Proposition 3. A strict implementation is not always possible without duplication: there is a graph on which some normal set of host pairs is strictly implemented only with duplication.

Proof. Consider the example in Figure 1. The set D is strictly implemented by a finite arc closed set of paths P that contains duplicate paths $x_0a_0a_1b_1b_0y_0$ and $x_0a_0a_2b_2b_0y_0$. Let us assume that the arc closed set of paths $P`$ without duplicate paths strictly implements the set $D$. Since $D$ contains 7 pairs of hosts, $P`$ strictly implements D and there are no duplicate paths in $P`$, $P`$ must contain exactly 7 paths. Then, by Proposition 2, we can choose the set $P`$ consisting only vertex-simple paths. In order to reach host y_j from host x_i, the path must go either via the arc a_1b_1 or via the arc a_2b_2. Let m_i paths, $i = 1, 2$, pass via the arc a_ib_i, and these paths start at n_i hosts and end at k_i hosts. Then $n_1 + n_2 = 3$, $k_1 + k_2 = 3$, $m_1 + m_2 = 7$. The set $P`$ is arc closed, which implies $n_1k_1 = m_1$, $n_2k_2 = m_2$. Hence $n_1k_1 + (3 - n_1)(3 - k_1) = 7$, which implies $2n_1k_1 = 3(n_1 + k_1) - 2$. In this example, $k_1 \leq 3$ and $n_1 \leq 3$ which implies: $3k_1 = 2$ for $n_1 = 0$, $k_1 = -1$ for $n_1 = 1$, $k_1 = 4$ for $n_1 = 2$, $3k_1 = 7$ for $n_1 = 3$. Each of these equations has no solution for non-negative integers or contradicts the condition $k_1 \leq 3$. We came to a contradiction, therefore, our assumption is not true, and the proposition is proved.

□

Proposition 4. A strict implementation is not always possible without cycles: there is a graph on which some normal set of host pairs is strictly implemented, but only by an infinite arc closed path set.

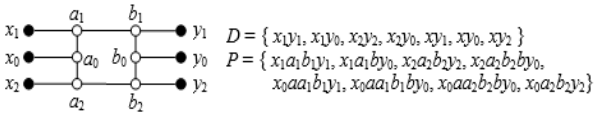

Fig. 1. The set D is strictly implemented only with duplication.

Proof. Consider the example in Figure 2. The set D is strictly implemented by arc closure $P{\downarrow}{\uparrow}$ of the set of paths P. But in P there are paths $x_1a_1b_1c_1c_2a_2b_2y_2$ and $x_2a_2b_2d_1d_2a_1b_1y_1$, which in $P{\downarrow}{\uparrow}$ induce a non-edge-simple path $x_1a_1b_1c_1c_2a_2b_2d_1d_2a_1b_1y_1$ that goes twice via the arc a_1b_1, and therefore $P{\downarrow}{\uparrow}$ is infinite. Let the arc closed set of paths $P`$ strictly implement the set $D$ and $P`$ is finite. Then, by Proposition 2, we can choose the set $P`$ consisting of vertex-simple paths.

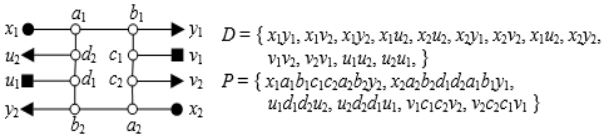

Fig. 2. The set D is strictly implemented only with cycles.

1. Let there be x_2y_1-path $\boldsymbol{p_1}$ passing via the arc c_2c_1.

1.1. Let there be a v_2v_1-path $\boldsymbol{p_2}$ passing via the arc c_2c_1. Then x_2y_1-path $\boldsymbol{p_1}$ and v_2v_1-path $\boldsymbol{p_2}$ both pass via the arc c_2c_1 and, therefore, induce an x_2v_1-path but $x_2v_1 \notin D$.

1.2. Therefore, any v_2v_1-path $\boldsymbol{p_3}$ does not pass via the arc c_2c_1, and then it passes via the arcs a_2b_2 and a_1b_1.

1.2.1. Let there be an x_1y_1-path $\boldsymbol{p_4}$ passing via the arc a_1b_1. Then v_2v_1-path $\boldsymbol{p_3}$ and x_1y_1-path $\boldsymbol{p_4}$ both pass via the arc a_1b_1 and, therefore, induce a v_2y_1-path but $v_2y_1 \notin D$.

1.2.2. Therefore, any x_1y_1-path $\boldsymbol{p_5}$ does not pass via the arc a_1b_1, and then it passes via the arc c_2c_1.

1.2.2.1. Let there be an x_2y_2-path $\boldsymbol{p_6}$ passing via the arc a_2b_2. Then v_2v_1-path $\boldsymbol{p_3}$ and x_2y_2-path $\boldsymbol{p_6}$ both pass via the arc a_2b_2 and, therefore, induce an x_2v_1-path but $x_2v_1 \notin D$.

1.2.2.2. Therefore, any x_2y_2-path does not pass via the arc a_2b_2. Such a path is unique among vertex-simple paths: $\boldsymbol{p_7} = x_2a_2c_2c_1b_1a_1d_2d_1b_2y_2$. Similarly, any x_1y_1-path does not go via the arc a_1b_1. Such a path is unique among vertex-simple paths: $\boldsymbol{p_5} = x_1a_1d_2d_1b_2a_2c_2c_1b_1y_1$. The paths $\boldsymbol{p_7}$ and $\boldsymbol{p_5}$ have a common arc a_2c_2, therefore the path $x_1a_1d_2d_1b_2a_2c_2c_1b_1a_1d_2d_1b_2y_2$ is induced, which passes twice through the arc a_1d_2, i.e. this path is not edge-simple.

2. Thus, any x_2y_1-path does not pass via the arc c_2c_1. Such a path is unique among vertex-simple paths: $\boldsymbol{p_8} = x_2a_2b_2d_1d_2a_1b_1y_1$. Due to symmetry, it is similarly proved that any x_1y_2-path does not pass via the arc d_2d_1. Such a path is unique among vertex-simple paths: $\boldsymbol{p_9} = x_1a_1b_1c_1c_2a_2b_2y_2$.

The paths $\boldsymbol{p_8}$ and $\boldsymbol{p_9}$ have a common arc a_1b_1, so the path $\boldsymbol{p}_{10} = x_2a_2b_2d_1d_2a_1b_1c_1c_2a_2b_2y_2$ is induced. This path passes twice through the arc a_2b_2, i.e. this path is not edge-simple. We came to a contradiction and, therefore, our assumption is not true, and the proposition is proved.

□

IV. A SUFFICIENT CONDITION FOR THE STRICT IMPLEMENTATION OF ANY SET OF HOST PAIRS WITHOUT CYCLES AND DUPLICATION

In this section, we investigate sufficient conditions on a graph that allow us to strictly implement any set of host pairs without cycles and duplication. If for two paths there is a merge on the arc ab and there is a separation after the arc cd, then we say that the separation occurs after the merge, if at least one of these paths first passes the arc ab and then the arc cd. Accordingly, merging occurs after separation, if at least one of these paths first passes the arc cd and then the arc ab. Note that in the linear order of the vertices of one path, the separation after the arc cd can occur after merging on the arc ab, and in the linear order of the vertices of the other path, on the contrary, the merging on the arc ab can occur after the separation after the arc cd, as demonstrated by the example of two paths: *xabefcdy* and *ufcdabev*, where different letters indicate different vertices.

Proposition 5. Given a finite set of complete paths, if there is no separation after merging, then there are no cycles but the converse is not always true.

Proof. The sufficiency follows from the fact that the arc closure can induce new paths only in the case of the separation after merging. Also, the cycle is induced by a non-edge-simple path, in which there is a separation after the merging, as indicated in Section 2. But the presence of the separation after merging does not necessarily mean the arc non-closure or the presence of cycles, as demonstrated by the following example of a finite arc closed set of complete paths without cycles $P = \{ xaby, xaby`, x`aby, x`aby` \}$, where the hosts x, y, $x`$ and $y`$ are pairwise different.

□

Proposition 6. Given an arc closed set P of complete paths, the absence of merging after separation is the necessary and sufficient condition for the absence of duplication.

Proof. Let there be two different xy-paths. Since each host is connected to exactly one switch, the maximum common prefix of these paths and the maximum common postfix of these paths each has length at least 1, and the prefix can be represented as $\boldsymbol{xpa}$, and the postfix as $\boldsymbol{bqy}$. Then, obviously, a is traversed in each of the paths earlier than b, and the paths are of the form $\boldsymbol{xpa}rb\boldsymbol{qy}$ and $\boldsymbol{xpa}r`b\boldsymbol{qy}$. Thus, these paths separate at the vertex $a$, and then merge at the vertex $b$. From this follows the sufficiency of the condition. Let us prove the necessity of the condition. Let there be a merge after the separation for two complete paths: $xy$-path and $x`y`$-path, with the xy-path first passing through the vertex a at which the paths are separated, and then the vertex b where the paths merge. Two cases are possible.

1) Figure 3 (a). The $x`y`$-path goes through the vertices a and b in the same order: ab. Then the paths have the form $\boldsymbol{xpaqbry}$ and $\boldsymbol{x`p`aq`br`y`}$, where the segments $\boldsymbol{xp}$ and $\boldsymbol{x`p`}$ end at the same vertex $c$, the segments $\boldsymbol{q}$ and $\boldsymbol{q`}$ start and end at different

vertices, the segments ***r****y* and ***r***`*y*` start at the same vertex *d*. The arc *ca* is common for these paths; therefore, the path *x****p****a****q***`*b****r***`*y*` is in the arc closure. Compare this path with the path *x****p****a****q****b****r****y*. The arc *db* is common for these paths, so the path *x****p****a****q***`*b****r****y* is in the arc closure. Since the segments ***q*** and ***q***` start and end at different vertices, the paths *x****p****a****q****b****r****y* and *x****p****a****q***`*b****r****y* are different, therefore, these are duplicate paths.

2) Figure 3 (b). The *x*`*y*`-path goes through the vertices *a* and *b* in the reverse order: *ba*. Then the paths have the form *x****p****a****q****b****r****y* and *x*`***p***`*b****q***`*a****r***`*y*`, where the segments *x****p*** and *b****q***` end at the same vertex *c*, the segments ***q*** and ***r***`*y*` start at different vertices, the segments ***q*** and *x*`***p***` end at different vertices, the segments ***r****y* and ***q***`*a* start at the same vertex *d*. The arc *bd* is common for these paths, so the path *x****p****a****q****b****q***`*a****r***`*y*` is in the arc closure. Compare this path with the path *x****p****a****q****b****r****y*. The arc *ca* is common for these paths, so the path *x****p****a****q****b****q***`*a****q****b****r****y* is in the arc closure. Compare this path with the *x****p****a****q****b****r****y* path. Since the segment ***q***`*a****q****b* is not empty, the paths *x****p****a****q****b****r****y* and *x****p****a****q****b****q***`*a****q****b****r****y* are different, therefore, these are duplicate paths.

□

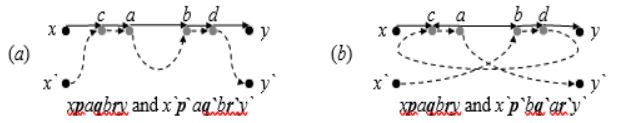

Fig. 3. Merging after separation.

A graph in which any normal set of host pairs can be strictly implemented without cycles is called *almost good*. A graph in which any normal set of host pairs can be strictly implemented without cycles and without duplication is called *good*.

A graph in which any normal set of host pairs can be strictly implemented without cycles is called *almost good*. A graph in which any normal set of host pairs can be strictly implemented without cycles and without duplication is called *good*.

A finite arc closed set of paths *P* connecting all pairs of different hosts (i.e., *H*(*P*) is the largest normal set of host pairs) will be called *almost perfect* if there is no path separation after the path merging, and *perfect* if, in addition, there is no path merging after the path separation. A graph will be called *almost perfect* or *perfect* if it contains, respectively, an almost perfect or perfect set of paths.

A sufficient condition for the strict implementation of any normal set of host pairs without cycles and duplication can now be formulated as the following proposition.

Proposition 7. An almost perfect graph is almost good, and a perfect graph is good. Moreover, the almost perfect set of paths for each normal set of host pairs contains its strict implementation without cycles as a subset, and the perfect set of paths for each normal set of host pairs contains its strict implementation without cycles and without duplication as a subset.

Proof. Since the almost perfect set of paths is finite and there is no separation after merging, any subset of it is also finite and there is no separation after merging, therefore, by Proposition 5, it is arc closed and does not generate cycles. By definition, a perfect set is almost perfect, so any subset of it is also finite, arc closed and does not generate cycles. Since there are no merging after separation in a perfect set of paths, there is no merging after any separation in any of its subsets, and, according to Proposition 6, it does not generate duplication. Given a normal set *D* of host pairs and an almost perfect (perfect) set *P* of paths, we can choose a subset *P*(*D*) such that *H*(*P*(*D*)) = *D*. The set *P*(*D*) of paths strictly implements the set *D* of host pairs without cycles and without duplication if the set *P* is perfect.

□

V. Conclusions

The paper shows that any set of host pairs can be implemented on a connected graph using paths connecting the hosts of given pairs, without the occurrence of cycles through which packets will circulate endlessly and endlessly multiply, and without duplicate paths, i.e. different paths connecting the same host pairs. However, this may result in extra paths connecting additional host pairs that are not in the given set of pairs. If the absence of undesired paths is required, then for some graphs some sets of host pairs are implemented only with duplication or cycles. The requirements on the graph are formulated and proved, which are sufficient to implement any set of host pairs without cycles (possibly with duplication), and without cycles and without duplication. At the same time, the very possibility of implementing on a graph any set of host pairs without cycles and, especially, without cycles and without duplication seems to be a fairly strong requirement. Therefore, we can hypothesize that these requirements on the graph are also necessary. Confirmation or refutation of this hypothesis is one of the areas for further research.

Acknowledgment

This work is partly supported by RFBR project N 20-07-00338 A.

References

[1] Sezer S., Scott-Hayward S., Chouhan P. K., Fraser B., Lake D., Finnegan J., Viljoen N., Miller M. and Rao N. Are we ready for sdn? Implementation challenges for software-defined networks *IEEE Communications Magazine*, 2013, 51, 7: pp. 36-43.

[2] López J., Kushik N., Yevtushenko N. and Zeghlache D. Analyzing and Validating Virtual Network Requests. *Proc. ICSOFT2017*: pp. 441-6.

[3] Yevtushenko N., Burdonov I., Kossatchev A., Lopez J., Kushik N. and Zeghlache D. Test Derivation for the Software Defined Networking Platforms: Novel Fault Models and Test Completeness *Proc. IEEE East-West Design and Test Symposium*, EWDTS2018, N 8524712: pp. 1-5.

[4] Burdonov I. B., Yevtushenko N. V. and Kossatchev A. S. Testing switch rules in software defined networks., *Trudy ISP RAN/Proc. ISP RAS*, 2018, vol. 30, issue 6: pp. 69-88 (in Russian).

[5] Burdonov I., Kossachev A., Yevtushenko N., López J., Kushik N. and Zeghlache D. Verifying SDN Data Path Requests, 2019, *CoRR abs/1906.03101*.

[6] Boufkhad Y., De La Paz R., Linguaglossa L., Mathieu F, Perino D. and Viennot L. Forwarding tables verification through representative header sets, 2016, *arXiv preprint arXiv:1601.07002*.