13. Shevchuk, V. P. Metrologiya intellektual'nyh izmeritel'nyh sistem [Text]: monografiya / V. P. Shevchuk, V. I. Kaplya, A. P. Zheltonogov, D. N. Lyasin. – Volgograd, 2005. – 210 p.
14. Cherepanska, I. Yu. Planuvannia, modeliuvannia ta veryfikatsiya protsesiv u hnuchkykh vyrobnychykh systemakh. Praktykum [Text] / I. Yu. Cherepanska, V. A. Kyryllovych, A. Yu. Sazonov, B. B. Samotokin. – Zhytomyr: ZhDTU, 2015. – 285 p.
15. Kolker, Ya. D. Matematicheskiy analiz tochnosti mekhanicheskoy obrabotki detaley [Text] / Ya. D. Kolker. – Kyiv: Tekhnika, 1976. – 200 p.
16. Tomashevs'kiy, V. M. Modelyuvannya sistem [Text] / V. M. Tomashevs'kiy. – Kyiv: Vidavnicha grupa BHV, 2005. – 352 p.

Запропоновано стратифікований підхід до імітаційного моделювання програмно-конфігурованих мереж. Запропоновано імітаційні моделі мережі, активних і пасивних компонентів – контролера, комутатора, хоста та комунікаційних каналів. Придатність підходу до цільового використання підтверджено шляхом співставлення одержаних результатів імітаційного моделювання із результатами емуляції мережі у середовищі Mininet

Ключові слова: програмно-конфігурована мережа, імітаційне моделювання, дискретно-подійна специфікація системи, великі дані

Предложен стратифицированный подход к имитационному моделированию программно-конфигурируемых сетей. Предложены имитационные модели сети, активных и пассивных компонентов – контроллера, коммутатора, хоста и коммуникационных каналов. Пригодность подхода к целевому использованию подтверждена путем сопоставления полученных результатов имитационного моделирования с результатами эмулирования сети в среде Mininet

Ключевые слова: программно-конфигурируемая сеть, имитационное моделирование, дискретно-событийная спецификация системы, большие данные

UDC 004.75 : 004.94
DOI: 10.15587/1729-4061.2017.110142

DEVELOPMENT OF STRATIFIED APPROACH TO SOFTWARE DEFINED NETWORKS SIMULATION

V. Shkarupylo
PhD, Associate Professor*
E-mail: shkarupylo.vadym@gmail.com
S. Skrupsky
PhD, Associate Professor*
E-mail: sskrupsky@gmail.com
A. Oliinyk
PhD, Associate Professor**
E-mail: olejnikaa@gmail.com
T. Kolpakova
PhD, Senior Lecturer**
E-mail: t.o.kolpakova@gmail.com
*Department of Computer Systems and Networks***
Department of Software Tools*
***Zaporizhzhya National Technical University
Zhukovskoho str., 64, Zaporizhzhya, Ukraine, 69063

1. Introduction

Nowadays the current state of global networking is on a verge of its drastic refinement. The questions of networks management convenience, control, monitoring, reconfiguring and scaling are topical here. The answer can be in modern approaches and concepts usage. Possible solution can be found in Software Defined Networking (SDN) principles following. There are some of them: differentiation between control and data planes, lightweight switches utilization, "controller" concept adoption – to coordinate switches in a centralized manner [1].

The SDN abbreviation will also be associated with Software Defined Network itself.

One of the main goals of SDN technology adoption is to foster the existing opportunities to effectively utilize available network resources in order to operatively meet the ad-hoc requirements of certain business-process (processes). To do that properly, the significant work yet has to be done. A plethora of different approaches and techniques have already been proposed to date though, e. g., to divert important traffic on a backup path to prevent packets loss and reduce jitter [2]. Appropriate solutions can be generalized as follows: it can be painful to get on with, the majority of solutions are aimed at emulation. This is not always acceptable in terms of corresponding time costs.

The development and deployment of systems on a basis of SDN principles is a non-trivial task, because of technology novelty and complexity. The validation of resulting solutions can be conducted by way of simulation or by way of testing. The simulation herein is a significantly less resource-intensive process, especially in the context of iterative development [3]. That's why the creation of an approach to such systems simulation is a topical task.

2. Literature review and problem statement

SDN technology is tightly bound with other ones, e. g., Cloud Computing [4], Neural Networks [5–7], Big Data [8], Artificial Intelligence [9], Internet of Things [10]. For instance, the synergy of SDN and Neural Networks technologies can be expedient in the context of parallel computing systems' resources allocation [11, 12]. Moreover, SDN is considered as a solution for Wireless Sensor Networks challenges [13]. The opportunities for SDN technology integration with existing 4G/5G mobile networks have also been discussed [14].

Different approaches to SDN simulation and emulation/prototyping are aimed at different aspects. Let us previously try to cover comparably well-known solutions.

The Mininet emulator is considered to be one of the most accessible platforms for SDN experimentations [15]. The background for that – easiness to get started with and use. The significant drawback – impossibility to conduct simulation (model time can't outpace the real time). Network creation and testing are time-consuming. Time costs grow exponentially with respect to hosts number growth (tree-like topology) [16]. Nevertheless, Mininet frequently appears in scientific publications as an exemplar to compete with and to validate own solutions [15]. To check the proposed approach, the same way of acting will be utilized. For instance, the CloudSimSDN framework is aimed directly at data centers virtualization scenarios [17]. The main accent here is put on cost-effectiveness and scalability aspects, with cloud computing scenarios under consideration. To validate the framework, the Mininet has been used as an exemplar.

It should also be noted that Mininet is a Linux-based solution, and can be installed as a virtual machine or as an installation for Linux platform. Not to mention that Windows platforms are significantly more widespread [18].

A good alternative to Mininet in terms of performance and scalability looks to be an EstiNet [19]. More in-depth comparison, encompassing also the ns-3 emulator, has been conducted in [20]. The mentioned ns-3 emulator can be considered a complementation to Mininet, or vice versa. A similar approach has been utilized to develop an OpenNet emulator, which combines the advantages of both mentioned solutions – the compatibility of Mininet and wireless networking support of ns-3 [21].

In an attempt to blur the boundaries between simulation and deployment, the framework on a basis of ns-3 has been proposed to allow running the controller software within the simulation environment [22]. With an accent on rapid standards refinement, an attempt has also been made to adopt the support of relatively novel OpenFlow 1.3 standard features, e. g., the support of multiple auxiliary TCP/UDP connections (to decrease jittering, packet loss), in a simulation environment within the OFSWITCH13 framework [23].

To generalize, it should be noted that the main trend of the reviewed approaches can be characterized as an attempt to combine both the simulation and emulation aspects within the proposed solutions' feature sets. This peculiarity inevitably implies certain drawbacks, e. g. the complication of these solutions practical usage, combined with overheads, stipulated by the necessity to accomplish a bunch of required preliminary steps, e. g., install and configure a virtual machine, etc. The necessity to deploy a virtual machine also typically stipulates the substantial requirements to hardware, which inevitably induces certain obstacles on these solutions practical usage. The proposed approach is devoted exceptionally to simulation. The main aspects: orientation on Windows-environment, easiness of reconfiguration, simulation-related overhead decrease.

To build the formalization for the described context and to formulate the problem, the DEVS (Discrete Event System Specification) formalism has been chosen [24]. It has been successfully proven to be a good solution for the purpose of validation by way of discrete event simulation, especially in the context of composite web services usage scenarios [25, 26].

To formulate a problem, let us consider the SDN network as a system, and the components of the latter – as system components. Let us distinguish the following groups of components: the active and passive ones. Let us represent the components of both groups with the following structure:

$$\langle \{controller\}, Switches, Hosts, Links, cs, sh \rangle, \quad (1)$$

where $\{controller\}$ – the one-element set representing the Controller component of the system – a single entity providing the centralized coordination of switches, represented with the elements of the $Switches$ set; $Hosts$ – a set of hosts to be connected to switches; $Links = CSLinks \cup SHLinks$ – a set of elements representing passive network components – links between the controller and switches ($CSLinks$ set) and links between switches and hosts ($SHLinks$ set); $cs: \{controller\} \times Switches \to CSLinks$ – a function to set connections between the elements of $\{controller\}$ and $Switches$ sets; $sh: Switches \times Hosts \to SHLinks$ – a function to set connections between the elements of $Switches$ and $Hosts$ sets.

Each set of the proposed structure (1) is intended to represent the type of the corresponding component.

The group of $\{controller\}$, $Switches$ and $Hosts$ sets is intended to represent the types of active network components, $Links$ set – the single type of passive ones.

The specificity of structure (1) is depicted in Fig. 1.

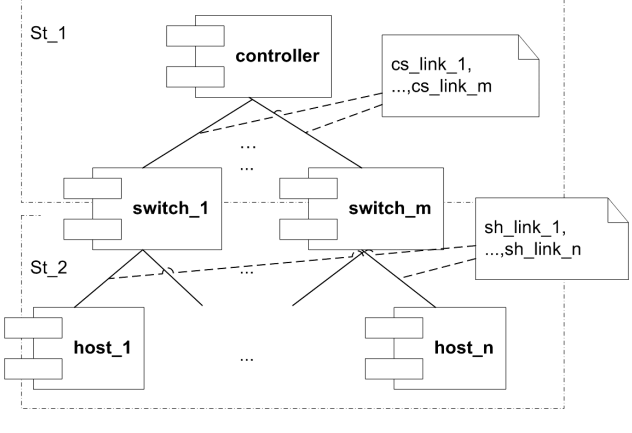

Fig. 1. Layers of network topology

In Fig. 1, the proposed stratified approach to SDN simulation is depicted, where both St_1 and St_2 layers represent the southbound elements of SDN architecture, with a slight accents shift – with respect to the types of active network devices involved. This step has been made to foster more transparent and structured transition from the proposed system representation to simulation models, grounded solely

on DEVS formalism concepts, e. g., "atomic" and "coupled" models.

As a reason for stratification, the differentiation between control and data planes of SDN architecture has been considered. The control plane takes place on the upper St_1 layer – between the elements of $\{controller\}$ and *Switches* sets, the data plane occurs on the lower St_2 one – between the elements of *Switches* and *Hosts* sets. Here it should also be noted that the elements of the *Switches* set can be associated with both – upper and lower – layers, because their functioning is all about control flows, as well as data flows, managing. The switches hence can be considered in our case as mediators (boundary components). This peculiarity implies certain limitations on their ports sets in particular.

Let us take a look at DEVS formalism concepts in detail, with respect to the named components types (1).

The atomic DEVS model is devoted to representing a certain system component of one of the named types – controller, switch, host, link.

The template of such model is given below to be used as the basis:

$$atomic = \langle IN, OUT, ST, \delta_{ext}, \delta_{\text{int}}, \lambda, ta \rangle, \quad (2)$$

where *atomic* is a structure representing certain component in terms of DEVS formalism; IN (OUT) – a set of events taking place on input (output) ports of the same names; $ST = \{st_1, st_2\}$ – a set of states labels: let $st_1 \in ST$ means that the component is passive (not functioning), $st_2 \in ST$ – is active (functioning); $\delta_{ext} : (in, st, e) \mapsto st'$ – external transition function, where $in \in IN$ – an event prompting the change of state label, $st \in ST$ – current state label, $0 \le e \le ta(st)$ – time, elapsed since the last transition, $st' \in ST$ – subsequent state label; $\delta_{\text{int}} : st \mapsto st'$ – internal transition function; $\lambda : st \mapsto out$ – output function, where $out \in OUT$ – an event on the corresponding output port; $ta : ST \to R^{+}_{0,\infty}$ – time advance function.

To proceed further, let us put some accent on the "event" notion. The events are of two types, depending on where they actually take place: on input ports – the first type; on output ports – the second one. Both these types can generally be considered in conjunction – the occurrence of the event of the second type can be characterized as a precondition for the occurrence of the first type event.

The auxiliary task here is to specify atomic DEVS models for all of the types of components distinguished, represented with controller, switches, hosts and links. It should be noted that, because of solely passive nature of the latter type representatives, the appropriate atomic models will play the roles of time delays, determined by the properties of a certain segment of virtual topology. It should also be mentioned that an accent has been put on this type distinguishing in order to simulate also the non-functional properties of a system (e. g., time delays, costs) with respect to the altering nature of virtual topology.

Atomic models of all the mentioned types then have to be grouped within the coupled DEVS model concept:

$$\langle CIN, COUT, Atomics, set, break \rangle, \quad (3)$$

where *CIN* (Coupled Input ports) – total set of input ports of all atomic models of all types (1), interconnected within the coupled one; *COUT* (Coupled Output ports) – total set of output ports; *Atomics* – total set of atomic models involved (e. g., $atomic \in Atomics$ (2)); $set : COUT \times Atomics \to CIN \times Atomics$ – a function to set connections between output and input ports;

$$break : 2^{Atomics} \to Atomics$$

is a function to decouple the connections between atomic models.

In comparison with the aforementioned atomic models (2), in the scope of the resulting coupled DEVS model (3), an accent is put on ports (not the events) – to set up the topology.

3. The aim and objectives of the study

The aim is to develop the approach to Software Defined Networks simulation. This will allow simplifying the validation of the obtained solutions with respect to the corresponding time costs lowering.

To achieve the formulated aim, the following objectives have to be accomplished:

– to propose the atomic and coupled simulation DEVS-models of the components of the named types and system;

– to check the resulting system coupled model adequacy;

– to evaluate the validity of the proposed approach.

4. Description of stratified approach to simulation and basic models

4. 1. Atomic models of components

One of the distinctive features of the approach is reconfigurability – with respect to the altering nature of virtual topology and non-functional properties.

As a starting point, let us take into consideration the types of network components, represented with structure (1). The first step is aimed at the aspects of these components representation within the scope of the atomic DEVS-model concept. Let us consider the (2) structure as a template. Depending on the type of component (1), structure (2) has to be modified appropriately.

Let us consider first the one-element $\{controller\}$ set. This step has been done to represent the centralization aspect of SDN. A single controller as the main point for control flows centralized orchestration. A certain analogy can be seen taking a look at the orchestration model, defined within WS-BPEL (BPEL4WS) specification, describing the rules of atomic web services centralized coordination to make the resulting composite web services function as required [27].

To specify the *controller* atomic model, the IN^c and OUT^c sets should be formed first, where the upper index *c* means the "controller".

$$IN^c = \{activate\} \cup \{in_i^c \mid i = 1,2,..,m \in N\},$$

where *activate* – an event on the input port of the same name to manually activate the model of the *controller* component, in_i^c – the *i*-th event on the input port of the same name to maintain the feedback from *m* atomic models of $switch_i \in Switches$ components. It's achieved via the atomic models of *m* corresponding $link_i \in CSLinks \subset Links$ compo-

nents (1). $OUT^c = \{out_i^c\}$ – a set of m events on output ports of the same names to maintain the control flow.

Now let us take a look at states labels set ST and functions. The ST set has been chosen to be completely the same as in (2). Moreover, this statement is valid for all atomic models described below. The differences between these models are going to be in the IN and OUT sets and δ_{ext}, λ functions: $\delta_{ext}^c : (activate, st_1, e) \mapsto st_2$ – to manually shift the model from the state with the $st_1 \in ST$ label to the state with the $st_2 \in ST$ label; $\delta_{\text{int}} : st_2 \mapsto st_1$ – to make the model become passive automatically – when all control flow-related computations are finished; $\lambda^c : st_2 \mapsto out_i^c$. Thus, the resulting atomic model of the *controller* component will be as follows:

$$controller = \langle IN^c, OUT^c, ST, \delta_{ext}^c, \delta_{\text{int}}, \lambda^c, ta \rangle. \quad (4)$$

Now let us consider the *Links* set of structure (1) as a type of the same name. This type has been introduced with a specific intention in mind – to simulate the non-functional properties of the system and, at the same time, to foster the easiness of topology reconfiguration – with respect to the ad-hoc nature of virtual topologies.

Each atomic model of *Links* type is intended to represent the corresponding path of the route, taking place in some current virtual topology between the components of different types. This means that models of components of the same type can't be connected via some model of $link_i \in Links$ – because of the specificity of previously defined tree-like stratified topology (Fig. 1).

Each atomic model of the $link_i \in Links$ component is intended to be implemented as two-directional. Considering, for instance, the upper St_1 layer of the stratified topology (Fig. 1), it can be seen that $\forall switch_i \in Switches \; \exists link_i \in Links$, which makes it possible to transmit the data (control flow) from *controller* to certain $switch_i \in Switches$, and also maintain the feedback.

To demonstrate the way of this assumption implementation, let us consider the upper layer St_1 of the stratified topology, where the control flow takes place. In the lower layer, the picture will be pretty the same, despite the fact that instead of single *controller* component representation there will be a set of $switch_i \in Switches$ components representations, and instead of the latter – $host_j \in Hosts$, and the data flow representation will take place.

The way of this approach implementation with respect to the St_1 layer is given in Fig. 2.

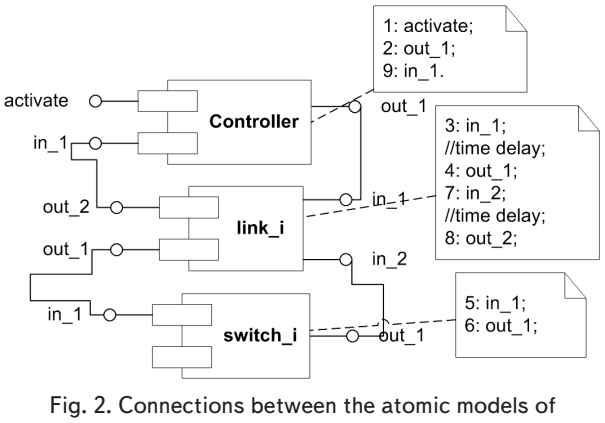

Fig. 2. Connections between the atomic models of components

In Fig. 2, the fragment of the upper layer (St_1) related to the topology (Fig. 1), implemented with DEVS formalism, is represented, with a reduced number of ports – to put an accent on the specificity of communicational delays implementation. The sequence of events occurrences on ports of the same names is represented as a numbered list within the note boxes. It should be mentioned here that representation of communicational channels as separate atomic DEVS models is one of the distinctive features of the approach, fostering the easiness of topology reconfiguration. This step will also simplify the simulation of nonfunctional properties, e. g., communicational delays.

Now let us estimate the total number of models to be synthesized to conduct the validation by way of DEVS-driven simulation. To this end, let $|Switches| = m$ and $|Hosts| = n$ (1). Then, taking into consideration Fig. 1, $|Links| = m + n$: $Links = CSLinks \cup SHLinks$: $CSLinks \cap SHLinks = \varnothing$, where $CSLinks = \{link_1, link_2, ..., link_m\}$ – a set of links between the *controller* component and the elements of the *Switches* set, $SHLinks = \{link_{m+1}, link_{m+2}, ..., link_{m+n}\}$ – a set of links between the elements of the *Switches* set and *Hosts* set. Hence, including also the atomic model of *controller* (4), the number of atomic models, needed to be synthesized, can be estimated with the following function:

$$f_1(m,n) = 2 \cdot (m+n) + 1. \quad (5)$$

It is also necessary to set up a topology itself. That's why a coupled model yet needs to be synthesized. Thus, the total number of DEVS models (atomic and coupled) to be synthesized can be estimated with the following function:

$$f_2(m,n) = f_1(m,n) + 1 = 2 \cdot (m+n) + 2 = 2 \cdot (m+n+1). \quad (6)$$

In accordance with Fig. 2, structure (1) and template structure (2), the atomic model of the

$$link_i \in Links \quad (i = 1,2,...,m+n)$$

component will be as follows:

$$link_i = \langle IN_i^l, OUT_i^l, ST, \delta_{ext}^l, \delta_{\text{int}}, \lambda^l, ta \rangle, \quad (7)$$

where, depending on a layer (St_1 or St_2, Fig. 1),

$$IN_i^l = \{in_1^{li}, in_2^{li}\}$$

is a set of events for a pair of input ports of the same name: $in_1^{li} \in IN_i^l$ – an event representing the appearance of message from *controller* ($switch_i \in Switches$, $i = 1,2,...,m$) on an input port, $in_2^{li} \in IN_i^l$ – from $switch_i \in Switches$ ($host_j \in Hosts$, $j = 1,2,...,n$) – if St_1 (St_2) is being considered; $OUT_i^l = \{out_1^{li}, out_2^{li}\}$ – a set of events for a pair of output ports of the same name: $out_1^{li} \in OUT_i^l$ – an event representing the appearance of message on an output, intended to be transmitted to $switch_i \in Switches$ ($host_j \in Hosts$), $out_2^{li} \in OUT_i^l$ – to *controller* ($switch_i \in Switches$) – if St_1 (St_2) is being considered; $\delta_{ext}^l : (in, st_1, e) \mapsto st_2$, where $in \in IN_i^l$; $\lambda^l : st_2 \mapsto out$, where $out \in OUT_i^l$.

Now let us take a look at boundary elements, represented with $switch_i \in Switches$ $(i = 1,2,...,m)$ elements – the representatives of both layers – St_1 and St_2:

$$switch_i = \left\langle IN_i^s, OUT_i^s, ST, \delta_{ext}^s, \delta_{\text{int}}, \lambda^s, ta \right\rangle, \quad (8)$$

where $IN_i^s = \left\{ in_1^{si}, in_2^{si}, ..., in_{k+1}^{si} \right\}$ – a set of events on input ports of the same name, where $in_1^{si} \in IN_i^s$ – an event, stipulated by the input message from *controller*, $in_2^{si} \in IN_i^s, ..., in_{k+1}^{si} \in IN_i^s$ – the events, associated with data obtaining from the corresponding hosts, $k \le n$ – the number of hosts, connected to certain $switch_i \in Switches$; $OUT_i^s = \left\{ out_1^{si}, out_2^{si}, ..., out_{k+1}^{si} \right\}$ – a set of events on output ports of the same name, where $out_1^{si} \in OUT_i^s$ – an event representing the acknowledgement of certain control flow part completion,

$$out_2^{si} \in OUT_i^s, ..., out_{k+1}^{si} \in OUT_i^s$$

is the events representing the appearance of messages (data) to be sent to the corresponding hosts connected; $\delta_{ext}^s : \left(in_1^{si}, st_1, e \right) \mapsto st_2$; $\lambda^s : st_2 \mapsto out$, where $out \in OUT^s$.

The atomic model of the $host_j \in Hosts$ component is as follows:

$$host_j = \left\langle IN_j^h, OUT_j^h, ST, \delta_{ext}^h, \delta_{\text{int}}, \lambda^h, ta \right\rangle, \quad (9)$$

where $IN_j^h = \left\{ in_1^{hj} \right\}$ – the one-element set of events taking place on a single input port of the same name, prompted by the message from some $switch_i \in Switches$, $OUT_j^h = \left\{ out_1^{hj} \right\}$ – on a single output port of the same name, prompted by message occurrence, ready to be transmitted to

$$switch_i \in Switches; \ \delta_{ext}^h : \left(in_1^{hj}, st_1, e \right) \mapsto st_2; \ \lambda^h : st_2 \mapsto out_1^{hj}.$$

4. 2. Coupled model of system

To specify the resulting coupled DEVS model, all previously given atomic models (4), (7)–(9) need to be aggregated with respect to structure (3). To set up the topology, the accent here is put on ports – not the events. For this purpose, the *CIN*, *COUT* and *Atomics* sets should be specified in particular:

$$CIN = IN^c \cup \left[\bigcup_{i=1}^{m+n} IN_i^l \right] \cup \left[\bigcup_{i=1}^{m} IN_i^s \right] \cup \left[\bigcup_{j=1}^{n} IN_j^h \right], \quad (10)$$

$$COUT = OUT^c \cup \left[\bigcup_{i=1}^{m+n} OUT_i^l \right] \cup \left[\bigcup_{i=1}^{m} OUT_i^s \right] \cup \left[\bigcup_{j=1}^{n} OUT_j^h \right], \quad (11)$$

$$Atomics = \left\{ controller \right\} \cup Switches \cup Hosts \cup Links. \quad (12)$$

The expressions (10)–(12) will be used to form structure (3).

5. Experimentation and results analysis

To conduct the experimentation, the following configuration has been successfully used:

– software environment: OS Microsoft Windows 7 Pro, SP; Java Runtime Environment 1.7; VirtualBox-4.3.10; mininet-2.2.1-150420-ubuntu-14.04-server-i386 virtual machine; eclipse-SDK-3.6.1-win32; DEVS Suite 2.0.0 as simulation environment; Xming X Server – to make it possible to visualize the topology of SDN network from Mininet in Windows environment; PuTTY utility – to connect to Linux-based virtual machine (where Mininet emulator is actually installed) from Windows-environment;

– hardware environment: central processing unit (CPU) – AMD Athlon 440 X3, 3GHz; random access memory (RAM) – 2 GB, 1333 MHz, DDR3.

To emulate SDN-network, the following resources have been allocated for the needs of virtual environment: 1024 MB of RAM, 2 physical cores of CPU, 32 MB of video memory. The configuration of network interfaces for the Linux-based virtual environment: the first network adapter has been chosen to be of Intel PRO/1000 MT Desktop (82540EM) type, the second one – of the same type, configured as VirtualBox Host-Only Ethernet Adapter.

To simulate SDN network, the "ping all" scenario has been utilized: each host of network pings other hosts. The tree topology with parameter depth=2 has been chosen. The simulation has been conducted from the non-functional properties values aggregation viewpoint (ping delays). For these purposes, ping delays have been represented with structure (7). The resulting coupled DEVS-model has been implemented on a basis of (3), (10)–(12) expressions.

The results are given in Table 1, where the results of DEVS-driven simulation on a basis of the proposed stratified approach are compared to the ones, obtained by way of emulation in Mininet environment.

Table 1

The results

No.	|*Switches*|, m	|*Hosts*|, n	$\xi_1(m,n)$, sec	$\xi_2(m,n)$, sec	$\xi_3(m,n)$, sec
1	1	2	5.667	5.518	0.228
2	3	4	5.985	5.946	0.411
3	4	9	6.427	6.515	0.735
4	5	16	9.489	9.611	1.213

In Table 1, $\xi_1(m,n)$ – time to complete the "pingall" command in Mininet environment; $\xi_2(m,n)$ – estimated values, obtained via DEVS Suite with respect to the proposed approach; $\xi_3(m,n)$ – actual time, spent on simulation. It can be noted that $\xi_3(m,n)$ values are significantly lower than $\xi_1(m,n)$ values – from 7.8 to 24.9 times.

On a basis of (5) and (6) expressions, the total number of all (just atomic) models synthesized is as follows: 8(7), 16(15), 28(27), 44(43).

6. Discussion of approach checking results

The adequacy of the resulting coupled DEVS-model has been proven with t-criterion for degrees of freedom $df = 8 - 2 = 6$ and significance level $\alpha = 0.05$:

$$t_{emp} = 0.992 < t_{cr} = 2.447,$$

where t_{emp} – calculated value, t_{cr} – critical value.

The lowering of time costs, as a result of the proposed approach to simulation usage, is prompted by the representation of non-functional properties – time costs – as estimated values.

The advantages of the proposed approach are conditioned by the fact that it is aimed solely at simulation (not taking into consideration the emulation aspects). Here are some of these: network visualization-related labor costs lowering – no need to deploy Xming X Server and use PuTTY utility, working on widespread Windows-platform; easiness of system model reconfiguration.

The drawback of the approach – impossible to test controller software.

The proposed approach is going to be extended further in terms of verification aspects adoption – to check the consistency of the resulting coupled model prior to the simulation. For this purpose, the model checking method can be successfully used [28]. One of the proved solutions to pay attention to is TLA+ formalism, based on the appropriate Temporal Logic of Actions [29]. It has been successfully used to check the Amazon web services design solutions [30]. Some steps in the direction of DEVS models TLA verification have also been done earlier [31].

7. Conclusions

1. The stratified approach to Software Defined Networks simulation, based on DEVS formalism, atomic and coupled models concepts usage, has been proposed. Controller-switches and switches-hosts communication layers have been defined. That allowed obtaining structured solutions, covering both functional and non-functional properties of a system.

2. Atomic simulation DEVS-models of active (controller, switch, host) and passive (link) components of the network (system), as well as the resulting DEVS-model of SDN network itself, have been proposed. This allowed conducting the validation of the system by way of simulation with an accent on visualization and easiness of the coupled model reconfiguration to meet ad-hoc requirements prompting the altering nature of virtual topologies.

3. The validity of the proposed approach has been proven experimentally. The adequacy of the resulting coupled DEVS-model has been proven with t-criterion. The results of DEVS-simulation have been compared to the results, obtained by way of emulation in Mininet environment. It has been shown during the case study on a basis of Mininet network testing scenario that the corresponding time costs are significantly lower (when talking about the proposed approach usage) in comparison with the ones, obtained by way of emulation in Mininet environment: from 7.8 to 24.9 times – on the utilized set of experimental data. Approach usage in Windows environment implies the following advantages: no need to utilize Xming X Server and PuTTY utility.

Further research is aimed at mathematically strict TLA+ formalism adoption to check the consistency of the resulting coupled DEVS model prior to the simulation.

Acknowledgements

The work has been conducted as part of:

– Erasmus+ Internet of Things: Emerging Curriculum for Industry and Human Applications ALIOT Project (reference number 573818-EPP-1-2016-1-UK-EPPKA2-CB-HE-JP), participated by Computer Systems and Networks (CSN) Dept., Software Tools Dept. of Zaporizhzhya National Technical University (Ukraine);

– research work "Methods and means of computational intelligence and parallel computing for processing large amounts of data in diagnostic systems" (number of state registration 0116U007419), the work "Methods and means of decision-making for data processing in intelligent image recognition systems" of Software Tools Dept. of Zaporizhzhya National Technical University (Ukraine).

References

1. Feamster, N. The road to SDN [Text] / N. Feamster, J. Rexford, E. Zegura // ACM SIGCOMM Computer Communication Review. – 2014. – Vol. 44, Issue 2. – P. 87–98. doi: 10.1145/2602204.2602219
2. Barrett, R. Dynamic Traffic Diversion in SDN: testbed vs Mininet [Text] / R. Barrett, A. Facey, W. Nxumalo, J. Rogers, P. Vatcher, M. St-Hilaire // 2017 International Conference on Computing, Networking and Communications (ICNC). – 2017. doi: 10.1109/iccnc.2017.7876121
3. Larman, C. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development [Text] / C. Larman. – 3rd ed. – New Jersey: Prentice Hall, 2004. – 736 p.
4. Govindarajan, K. Realizing the Quality of Service (QoS) in Software-Defined Networking (SDN) based Cloud infrastructure [Text] / K. Govindarajan, K. C. Meng, Hong Ong, Wong Ming Tat, Sridhar Sivanand, Low Swee Leong // 2014 2nd International Conference on Information and Communication Technology (ICoICT). – 2014. doi: 10.1109/icoict.2014.6914113
5. Kolpakova, T. Tender Participants Selection Based on Artificial Neural Network Model for Alternatives Classification [Text] / T. Kolpakova, V. Lovkin // Advances in Intelligent Systems and Computing. – 2016. – P. 3–10. doi: 10.1007/978-3-319-48923-0_1
6. Oliinyk, A. O. Using parallel random search to train fuzzy neural networks [Text] / A. O. Oliinyk, S. Yu. Skrupsky, S. A. Subbotin // Automatic Control and Computer Sciences. – 2014. – Vol. 48, Issue 6. – P. 313–323. doi: 10.3103/s0146411614060078
7. Oliinyk, A. O. Experimental investigation with analyzing the training method complexity of neuro-fuzzy networks based on parallel random search [Text] / A. O. Oliinyk, S. Yu. Skrupsky, S. A. Subbotin // Automatic Control and Computer Sciences. – 2015. – Vol. 49, Issue 1. – P. 11–20. doi: 10.3103/s0146411615010071
8. Oliinyk, A. A. The model for estimation of computer system used resources while extracting production rules based on parallel computations [Text] / A. A. Oliinyk, S. Yu. Skrupsky, V. V. Shkarupylo, S. A. Subbotin // Radio Electronics, Computer Science, Control. – 2017. – Issue 1. – P. 142–152. doi: 10.15588/1607-3274-2017-1-16
9. Subbotin, S. Individual prediction of the hypertensive patient condition based on computational intelligence [Text] / S. Subbotin, A. Oliinyk, S. Skrupsky // 2015 International Conference on Information and Digital Technologies. – 2015. doi: 10.1109/dt.2015.7222996
10. Silva, J. S. People-Centric Internet of Things [Text] / J. S. Silva, P. Zhang, T. Pering, F. Boavida, T. Hara, N. C. Liebau // IEEE Communications Magazine. – 2017. – Vol. 55, Issue 2. – P. 18–19. doi: 10.1109/mcom.2017.7841465
11. Oliinyk, A. Parallel Computer System Resource Planning for Synthesis of Neuro-Fuzzy Networks [Text] / A. Oliinyk, S. Skrupsky, S. A. Subbotin // Advances in Intelligent Systems and Computing. – 2016. – P. 88–96. doi: 10.1007/978-3-319-48923-0_12

12. Oliinyk, A. A. The decision tree construction based on a stochastic search for the neuro-fuzzy network synthesis [Text] / A. A. Oliinyk, S. A. Subbotin // Optical Memory and Neural Networks. – 2015. – Vol. 24, Issue 1. – P. 18–27. doi: 10.3103/s1060992x15010038
13. Kobo, H. I. A Survey on Software-Defined Wireless Sensor Networks: Challenges and Design Requirements [Text] / H. I. Kobo, A. M. Abu-Mahfouz, G. P. Hancke // IEEE Access. – 2017. – Vol. 5. – P. 1872–1899. doi: 10.1109/access.2017.2666200
14. Sun, S. Integrating network function virtualization with SDR and SDN for 4G/5G networks [Text] / S. Sun, M. Kadoch, L. Gong, B. Rong // IEEE Network. – 2015. – Vol. 29, Issue 3. – P. 54–59. doi: 10.1109/mnet.2015.7113226
15. Gupta, M. Fast, accurate simulation for SDN prototyping [Text] / M. Gupta, J. Sommers, P. Barford // Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking – HotSDN '13. – 2013. doi: 10.1145/2491185.2491202
16. Keti, F. Emulation of Software Defined Networks Using Mininet in Different Simulation Environments [Text] / F. Keti, S. Askar // 2015 6th International Conference on Intelligent Systems, Modelling and Simulation. – 2015. doi: 10.1109/isms.2015.46
17. Son, J. CloudSimSDN: Modeling and Simulation of Software-Defined Cloud Data Centers [Text] / J. Son, A. V. Dastjerdi, R. N. Calheiros, X. Ji, Y. Yoon, R. Buyya // 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. – 2015. doi: 10.1109/ccgrid.2015.87
18. Ganguli, S. Computer Operating Systems: From every palm to the entire cosmos in the 21st Century Lifestyle [Text] / S. Ganguli // Computer Society of India Communications. – 2017. – Vol. 40, Issue 11. – P. 5–8. – Available at: http://www.csi-india.org/Communications/CSIC_Feb_2017.pdf
19. Wang, S.-Y. Comparison of SDN OpenFlow network simulator and emulators: EstiNet vs. Mininet [Text] / S.-Y. Wang // 2014 IEEE Symposium on Computers and Communications (ISCC). – 2014. doi: 10.1109/iscc.2014.6912609
20. Wang, S.-Y. EstiNet openflow network simulator and emulator [Text] / S.-Y. Wang, C.-L. Chou, C.-M. Yang // IEEE Communications Magazine. – 2013. – Vol. 51, Issue 9. – P. 110–117. doi: 10.1109/mcom.2013.6588659
21. Chan, M.-C. OpenNet: A simulator for software-defined wireless local area network [Text] / M.-C. Chan, C. Chen, J.-X. Huang, T. Kuo, L.-H. Yen, C.-C. Tseng // 2014 IEEE Wireless Communications and Networking Conference (WCNC). – 2014. doi: 10.1109/wcnc.2014.6953088
22. Ivey, J. Comparing a Scalable SDN Simulation Framework Built on ns-3 and DCE with Existing SDN Simulators and Emulators [Text] / J. Ivey, H. Yang, C. Zhang, G. Riley // Proceedings of the 2016 annual ACM Conference on SIGSIM Principles of Advanced Discrete Simulation – SIGSIM-PADS '16. doi: 10.1145/2901378.2901391
23. Yang, H. Support Multiple Auxiliary TCP/UDP Connections in SDN Simulations Based on ns-3 [Text] / H. Yang, C. Zhang, G. Riley // Proceedings of the Workshop on ns-3 – 2017 WNS3. – 2017. doi: 10.1145/3067665.3067670
24. Discrete-Event Modeling and Simulation: Theory and Applications [Text] / G. A. Wainer, P. J. Mosterman (Eds.). – CRC Press, 2010. – 534 p. doi: 10.1201/b10412
25. Shkarupylo, V. A technique of DEVS-driven validation [Text] / V. Shkarupylo // 2016 13th International Conference on Modern Problems of Radio Engineering, Telecommunications and Computer Science (TCSET). – 2016. doi: 10.1109/tcset.2016.7452097
26. Shkarupylo, V. A Simulation-driven Approach for Composite Web Services Validation [Text] / V. Shkarupylo // Proc. 27th Int. Central European Conference on Information and Intelligent Systems, CECIIS 2016. – 2016. – P. 227–231.
27. Papazoglou, M. P. Service-Oriented Computing: State of the Art and Research Challenges [Text] / M. P. Papazoglou, P. Traverso, S. Dustdar, F. Leymann // Computer. – 2007. – Vol. 40, Issue 11. – P. 38–45. doi: 10.1109/mc.2007.400
28. Clarke, E. M. Model Checking [Text] / E. M. Clarke, O. Grumberg, D. Peled. – Cambridge, MA: MIT Press, 1999. – 314 p.
29. Lamport, L. Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers [Text] / L. Lamport. – Boston: Addison-Wesley, 2002. – 364 p.
30. Newcombe, C. How Amazon web services uses formal methods [Text] / C. Newcombe, T. Rath, F. Zhang, B. Munteanu, M. Brooker, M. Deardeuff // Communications of the ACM. – 2015. – Vol. 58, Issue 4. – P. 66–73. doi: 10.1145/2699417
31. Cristia, M. A TLA+ Encoding of DEVS Models [Text] / M. Cristia // Proc. Int. Modeling and Simulation Multiconference. – Buenos Aires, 2007. – P. 17–22.