Routing Technology Based On Virtualization Software-Defined Networking Concepti

Yurii Kravchenko[1], Kostiantyn Herasymenko[1], Olena Starkova[1] and Anna Bulgakova[1]

[1] *Taras Shevchenko National University of Kyiv, 60 Volodymyrska Street, Kyiv, 01033, Ukraine*

Abstract

One of the main characteristics of the digital revolution is the acceleration of change. The technologies that have fueled the digital revolution over the past decades are experiencing increasingly rapid innovation cycles.

For today, a substantial growth of amount of users, devices, applications and traffic has presented new challenges to service providers. The SDN paradigm emerged to address some of these emerging challenges. SDN simplifies network management and allows automated network configuration on demand with optimal use of network resources.

The work is devoted to the present networks research and the identification of opportunities for the implementation of virtualization of network functions in the context of SDN and NFV concepts. Because these are two new technological trends that are transforming network management. Together, they simplify the provisioning of network resources and provide greater network flexibility.

Keywords

Routing method, static routing, dynamic routing, NFV, SDN

1. Introduction

Network technologies are evolving at an exceptionally rapid pace, driven by the extensive utilization of information technologies and telecommunication systems across diverse sectors of human activity. Traditionally, network devices employed standard protocols and services to execute their intended functions effectively [1]. However, over time, several limitations and incompatibilities with the contemporary challenges of modern networks have become apparent. These challenges include the exponential increase in the number of users, the proliferation of services, the dramatic rise in the number of devices, and the expansion of communication channels, all contributing to heightened requirements and expectations for network infrastructures [2].

In response to these emerging challenges, new trends have surfaced in the field of network development, particularly focusing on the implementation of Software-Defined Networking (SDN) and Network Functions Virtualization (NFV). These technologies represent a fundamental shift from traditional network architectures, offering enhanced flexibility, scalability, and manageability of network resources. SDN and NFV aim to address the shortcomings of legacy network systems by decoupling network functions from the underlying hardware, thus enabling more dynamic and programmable network environments. This evolution necessitates the development of improved protocols and innovative technologies to meet the current and future demands of advanced information and communication networks and systems [3, 4].

The increasing adoption of network function virtualization is a primary factor driving the integration of SDN. SDN and NFV technologies are inherently complementary; while NFV focuses

14th International Scientific and Practical Conference from Programming UkrPROG'2024, May 14-15, 2024, Kyiv, Ukraine

* Corresponding author.

† These authors contributed equally.

yurii.kravchenko@knu.ua (Yu. Kravchenko); c.herasymenko@gmail.com (K. Herasymenko); elesta.tcs@gmail.com (O. Starkova); annbulgakova10@gmail.com (A. Bulgakova)

0000-0002-0281-4396 (Yu. Kravchenko); 0000-0002-9545-5272 (K. Herasymenko); 0000-0001-8985-2442 (O. Starkova)

on enhancing the actual network services that manage data flows, SDN provides a robust control plane that governs the overall behavior of the network [5]. This symbiotic relationship enhances the efficiency and effectiveness of network operations, allowing for greater adaptability and responsiveness to changing demands.

The primary objective of this research is to develop a comprehensive simulation environment for implementing and evaluating various network functions and protocol solutions. This environment is intended to serve as a critical tool for conducting experimental research, enabling researchers to test and refine new network technologies and protocols. In the long term, the insights gained from these experimental studies will facilitate the deployment of these innovative solutions within the SDN framework, ensuring they meet the stringent requirements of next-generation information and communication networks [2, 3].

2. First level heading Routing Technology Based on NFV and SDN Concepts

The core of this work is centered around the router, a traditional network device. The primary functions of a router include selecting optimal routes to remote networks, populating the routing table, and forwarding packets between networks [5].

In the routing process, routers consider several alternative routes to reach a single destination. These alternatives are the result of redundancy built into most network designs.

To develop the simulation system, the following were utilized:

- Python programming language;
- Python modules: ipaddress, threading, socket, json, tabulate.

The operation of simulating routers involves many objects interacting with each other [2]. The Simulation object is necessary for storing router objects. It can start all routers simultaneously using the start_routers method and print an interface table with statuses and other information about the interface objects with the print(simulation) method.

```
class Simulation:

    def __init__(self):
        self.routers = {}

    def add_router(self, router_name, router_interfaces):
        self.routers[router_name] = Router(router_name, router_interfaces)

    def start_routers(self):
        for router in self.routers:
            self.routers[router].start()

    def __str__(self):
        data = {"Router": [], "Number": [], "Port": [], "Broadcast": [], "IP": [], "Status": []}
        for router in self.routers:
            for interface in self.routers[router].interfaces:
                data["Router"].append(interface.hostname)
                data["Number"].append(interface.number)
                data["Port"].append(interface.local_port)
                data["Broadcast"].append(interface.broadcast_port)
                data["IP"].append(interface.get_ip())
                data["Status"].append(interface.status)

        return tabulate(data, headers='keys', tablefmt='grid')
```

Figure 1: Simulation Class.

The router is the primary object in the simulation. It stores interfaces that are necessary for the exchange of information between routers. Additionally, the router maintains a routing table and other essential data about the router.

```python
class Router:

    def __init__(self, hostname, interfaces):

        self.hostname = hostname

        self.interfaces = interfaces

        self.ip_list = []

        for interface in self.interfaces:
            self.ip_list.append(str(interface.get_ip()))
            interface.hostname = self.hostname
            interface.routing_function = self.__parse_interface_data

        self.__create_broadcast()

        self.data_packet = {"src_ip": "ip", "dst_ip": "ip", "data": "message"}
        self.broadcast_packet = {"src_ip": "ip", "src_port": "port", "data": "message"}

        self.threads = {}

        # self.routing_table = {"network": {"gateway": "0.0.0.0", "interface": 0, "metric": 20}}
        self.routing_table = {}

        self.dynamic_protocol = RIP(router=self)
```

Figure 2: Router Class.

The **Router** object has several public methods:

- *set_protocol:* This method sets the dynamic routing protocol object that will run on the router.
- has_ip: This method checks whether a given IP address exists on the router.
- *message_to_interface:* This method sends a message from a specific interface. It requires the interface number from which the message will be sent and the message itself [6].
- *message_to_ip:* This method sends a message to a specific IP address. It requires the IP address and the message to be sent. The router will then attempt to find a path for sending it through the connected interfaces or the routing table.
- *message_to_broadcast:* This method sends a message to the broadcast port. All interfaces with the same broadcast address will receive the message.
- *set_default_route:* This method sets the default route. It requires the IP address, the interface number through which network access is made, and the metric. The metric is necessary to find the quickest way to send the message.
- *add_route:* This method sets a route to a specific network. It requires the IP address and subnet mask of the remote network, the IP address and interface number through which access to the remote network is made, and the metric [2-6].
- *start:* This method starts the router. After this, the router begins to listen to the broadcast port and operate the dynamic routing protocol.

The **Interface** object represents a router interface that can interact with other interfaces for data exchange.

```
class Interface:

    def __init__(self, number, local_port, broadcast_port, interface):
        self.number = number
        self.local_port = local_port
        self.broadcast_port = broadcast_port
        self.interface = interface
        self.status = "Unused"

        self.hostname = None
        self.routing_function = None
        self.broadcast_socket = None
        self.listener = None

        self.connection = {}
```

Figure 3: Interface Class.

The **Interface** object has the following methods:

- *get_ip* Returns the IP address of the interface.
- *get_conn:* Returns the connection object of the interface to another interface.
- *connect_to_router*: Initiates a series of actions to establish a connection with another interface.
- *wait_connection:* Waits for a response from another interface to confirm the connection.
- *accept_connection:* Confirms the connection to another interface.
- *listen_conn:* Awaits the establishment of a connection with another interface.

To establish a connection between two interfaces, the following steps are necessary:

1. Call the connect_to_router method on the first interface and pass the IP address of the remote interface to which the connection is being created [7, 8]. The first interface will then wait for the connection to be established.
2. Call the accept_connection method on the second interface, to which the first interface is connecting, and pass the address of the first interface that wants to connect and its port for connecting the interfaces (local_port) [4-8].

RoutingProtocol is an abstract class designed for creating dynamic routing protocols. It defines the main methods that must be present in all dynamic routing protocols.

RIP is an object created to demonstrate the organization and operation of dynamic routing protocols in the simulation. It has the following methods:

- *new_message*: Processes the received message addressed to the dynamic routing protocol.
- *get_routing_table_to_send*: Processes the routing table for further transmission to other routers.
- *send_route*: Sends the routing table, prepared by the *get_routing_table_to_send* method, from all router interfaces (to all network neighbors).
- *runner*: Sends messages every n seconds.
- *start*: Initiates the protocol operation. After execution, the runner method launches a cycle that sends the routing table to all network neighbors every n seconds.

3. Testing the Developed Routing Technology

To begin the simulation, it is essential to create all necessary objects. This involves generating lists of interface objects by providing parameters such as the interface number, interface port, bandwidth port, and an **IPv4Interface** object for recording the IP address.

Next, you need to call a method from the simulation object to create and add routers to the simulation. This requires passing the router's name and the interfaces created in the previous step

[4-8]. You can create as many routers as needed, but careful attention must be given to the creation of interfaces, as the simulation will only function correctly with properly configured parameters, ensuring no data overlaps between interfaces except for the broadcast port.

Once the routers are set up, they are started, and the **message_to_broadcast** method is invoked on all routers. This step is crucial for the automatic connection of interfaces sharing the same broadcast port.

The simulation employs a software interface to facilitate data exchange between routers, achieved through inter-process communication mechanisms like sockets [4-8]. Network ports are necessary for socket operations, so during the simulation, these ports must be specified for each interface. Sockets replace the second layer of the OSI model, allowing the creation of socket connections between two software objects and enabling data exchange between them [4-8].

Within the interface object, broadcast sockets are responsible for listening and transmitting data on a designated port. When establishing a connection between two interfaces, a socket connection is created between the ports of these interfaces, enabling data transmission and reception between the two interfaces.

The process of connecting interfaces through socket connections can be summarized as follows:

1. Several routers have some interfaces using the same broadcast port. This setup is illustrated in the following example.

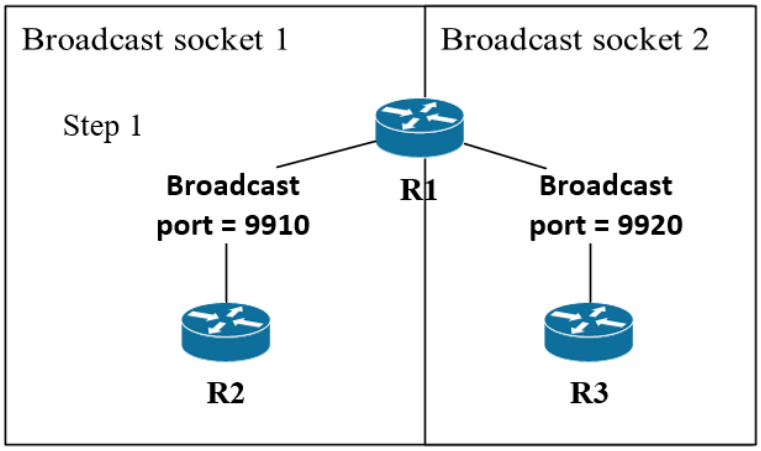

Figure 4: State Before Establishing Socket Connections.

2. One of the interfaces sends a message to the broadcast socket indicating its intention to establish a connection.
3. Another interface receives this message and, if it is available (i.e., not yet connected to another interface), responds affirmatively.
4. A connection (socket) is then established on each interface. From this point on, these interfaces are connected and can exchange data through the established socket connection.

All transmitted data is encapsulated in analog packets, which are structured as follows:

- A packet for a regular message via a socket connection looks like this:
  {"src_ip": "ip", "dst_ip": "ip", "data": "message"}
- A packet for a broadcast message is structured as follows:
  {"src_ip": "ip", "src_port": "port", "data": "message"}

In the RIP dynamic routing protocol, data is packaged as follows:

```
{"type": "RIP", "routing_table": None}
```

Once a connection is established between two interfaces, a message can be sent from one interface, received by the other, and processed if necessary [4-8].

Currently, there are no applications that allow for the creation of custom dynamic routing algorithms in Python. To integrate custom code as a dynamic routing protocol into the simulation, the following conditions must be met: the created dynamic routing class must inherit from the abstract class RoutingProtocol; the created class must implement two methods abstractly defined in the RoutingProtocol class. The RIP class can serve as an example for writing custom protocols based on its structure.

By adhering to these guidelines and thoroughly testing the developed routing technology, it ensures that the simulation is robust, reliable, and ready for real-world network environments.

4. Practical investigation of the developed routing technology

The practical significance of the developed routing technology lies in its ability to implement simulation results (both mathematical and imitative) using the developed technology. This allows for the practical realization of advanced routing methods without changing the actual protocol solutions, which often have numerous shortcomings and do not reflect the features of modern networks, thereby becoming obsolete.

For instance, solving a routing problem with the RIP protocol metric can be modeled as a Boolean programming task using Matlab's tools [9].

4.1. Routing problem formulation

Given:

- Number of nodes in the network *(m)*
- Number of communication channels in the network *(n)*
- Packet sender node
- Packet receiver node
- Traffic intensity *(r)* entering the network
- Bandwidths of the communication channels
- Metrics of the communication channels *(f)*

We need to determine:

- The shortest path from the sender node to the receiver node through the communication channel of the modeled network within the chosen metric.
- The dependence of the number of paths used in the routing process on the traffic density entering the network.
- The bandwidth of the communication direction from the sender node to the receiver node according to the chosen routing model.

4.2. Routing Problem Description as a Boolean Programming Problem Using RIP v.1 Metrics

The number of communication channels in the network (n) determines the size of vector x, where the coordinates $x_{i,j}$ represent the share of traffic in the communication channels between the i-th and j-th nodes. The size of the metric vector f also corresponds to the number of communication channels in the network (n), and its coordinates $f_{i,j}$ characterize the metric of the communication channels between the i-th and j-th nodes.

To implement routing, the coordinates of vector x are subject to the following constraints:

$$x_{i,j} \in \{0,1\} \quad (i,j = \overline{1,m}\,; \quad i \neq j), \tag{1}$$

This means that the variable $x_{i,j}$ can only take two values:

- 1, if traffic passes through the channel (i,j);
- 0, otherwise.

This Boolean value (1) ensures that there are no deviations in the flows along the network path, i.e., all traffic is transmitted via a single path.

In the first case, each communication channel is assigned a metric equal to 1 $\left(f_{i,j} = 1,; \quad i,j = \overline{1,m}; \quad i \neq j\right)$, corresponding to finding the path with the minimum number of retransmissions (*fval*) between a given pair of nodes, as in protocol RIP v.1.

When solving the routing problem, it is necessary to ensure the flow conservation conditions for each network node and the network as a whole:

$$\begin{cases} \sum_{j:(i,j)} x_{i,j} - \sum_{j:(j,i)} x_{j,i} = 1 \; - \text{ for the sender node} \\ \sum_{j:(i,j)} x_{i,j} - \sum_{j:(j,i)} x_{j,i} = 0 - \text{for the transit node} \\ \sum_{j:(i,j)} x_{i,j} - \sum_{j:(j,i)} x_{j,i} = \; -1 \quad - \text{for the receiver node} \end{cases} \tag{2}$$

In addition to the flow conservation condition (2), the condition to avoid link overload must also be satisfied:

$$r \cdot x_{i,j} \leq c_{i,j} \quad i,j = \overline{1,n}, i \neq j, \tag{3}$$

where $c_{i,j}$ – represents the capacity of the communication channel between nodes *I* and *j*.

Solving routing problems in modern network protocols typically involves finding the shortest path in a network, which defines the structure of the communication system. The shortest path problem in a network is formulated as a boolean programming problem, and the "Optimization Toolbox" in Matlab, specifically the "bintprog" subroutine, is used for its solution.

According to the equation (1), solving the boolean programming problem requires minimizing the objective function expressed in linear form $\min_x f^t x$ subject to a series of constraints represented as equality and inequality constraints: $A \cdot x \leq b; \quad Aeq \cdot x = beq$, where *f*, *x*, *b*, *beq* are vectors, and *A*, *Aeq* are matrices of corresponding dimensions.

Matlab's environment supports several syntax variants to call the "bintprog" subroutine:

1. $[x, fval] = bitprog(f, A, b)$ solves $fval = \min_x f^t x$ subject to $A \cdot x \leq b$.
2. $[x, fval] = bitprog(f, A, b, Aeq, beq)$ solves $fval = \min_x f^t x$ subject to $Aeq \cdot x = beq$.

If there are no inequality constraints, *A*=[] and *b*=[].

To describe the routing problem in Matlab's formalism, the flow conservation condition (2) should be expressed as a vector matrix: $Aeq \cdot x = beq$. Thus, the matrix *Aeq* has dimensions $m \times n$, and its coordinates take values $\{-1; 0; 1\}$ as follows $(j = \overline{1,m}, \quad i = \overline{1,n})$:

$a_{ji} = 1$, if the *i*-th communication channel goes from node *j*;

$a_{ji} = -1$, if the *i*-th communication channel enters node *j*;

$a_{ji} = 0$, if the *i*-th communication channel does not involve node *j*.

The size of the vector *beq* corresponds to the number of nodes in the network *(m)*, and its coordinates are generated as follows $(j = \overline{1,m})$:

$beq_j = 1$ if the i-th node is a packet sender;

$beq_j = -1$ if the i-th node is a packet receiver;

$beq_j = 0$ if the i-th node is a retransmitter.

Conditions (3) should also be expressed as a vector matrix with inequality $A \cdot x \leq b$.

4.3. As an example of a metric routing problem for the RIP v.1 protocol, formulated and solved in Matlab:

The number of communication channels in the network (*n*) determines the size of vector *x*, where the coordinates $x_{i,j}$ represent the share of traffic in the communication channels between the *i*-th and *j*-th nodes. The size of the metric vector f also corresponds to the number of communication channels in the network (*n*), and its

Let the network structure and channel capacities be shown in Fig. 5. The total number of nodes in the network is five (*m* = 5), and the number of communication channels is six (*n* = 6). Then, the packet sender node is 1, and the receiver node is 5.

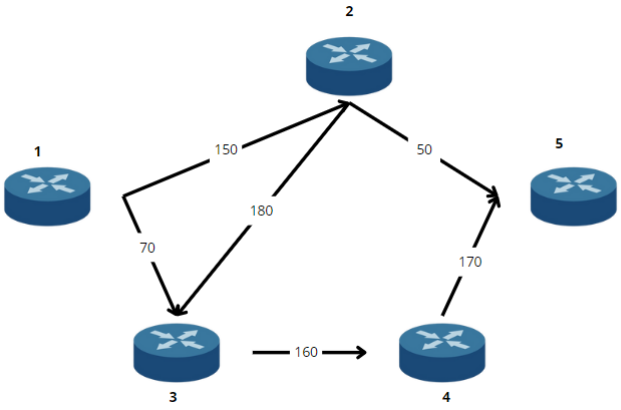

Figure 5: Example of a modeled network structure.

Let's formulate vector x and metric vector f :

$$x = \begin{bmatrix} x_{1,2} \\ x_{1,3} \\ x_{2,3} \\ x_{2,5} \\ x_{3,4} \\ x_{4,5} \end{bmatrix} \quad f = \begin{bmatrix} f_{1,2} \\ f_{1,3} \\ f_{2,3} \\ f_{2,5} \\ f_{3,4} \\ f_{4,5} \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}.$$

Formulate the flow conservation conditions for network nodes (2):

$$\begin{cases} x_{1,2} + x_{1,3} = 1; \\ -x_{1,2} + x_{2,3} + x_{2,5} = 0; \\ -x_{1,3} - x_{2,3} + x_{3,4} = 0; \\ -x_{3,4} + x_{4,5} = 0; \\ -x_{2,5} - x_{4,5} = -1. \end{cases}$$

Formulate the matrix Aeq and the vector beq:

$$Aeq = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 1 & 0 & 0 \\ 0 & -1 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & -1 & 0 & -1 \end{bmatrix}; \quad beq = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ -1 \end{bmatrix}.$$

Formulate the conditions for preventing overloading of communication channels (3):

$$\begin{cases} r \cdot x_{1,2} \le c_{1,2}; \\ r \cdot x_{1,3} \le c_{1,3}; \\ r \cdot x_{2,3} \le c_{2,3}; \\ r \cdot x_{2,5} \le c_{2,5}; \\ r \cdot x_{3,4} \le c_{3,4}; \\ r \cdot x_{4,5} \le c_{4,5}. \end{cases}$$

Formulate the matrix A and the vector b

$$A = \begin{bmatrix} r & 0 & 0 & 0 & 0 & 0 \\ 0 & r & 0 & 0 & 0 & 0 \\ 0 & 0 & r & 0 & 0 & 0 \\ 0 & 0 & 0 & r & 0 & 0 \\ 0 & 0 & 0 & 0 & r & 0 \\ 0 & 0 & 0 & 0 & 0 & r \end{bmatrix}; \quad b = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ -1 \end{bmatrix}.$$

In the example provided above, the program code appears as shown in Figure 6. Calculation results are displayed in the command window (Figure 6).

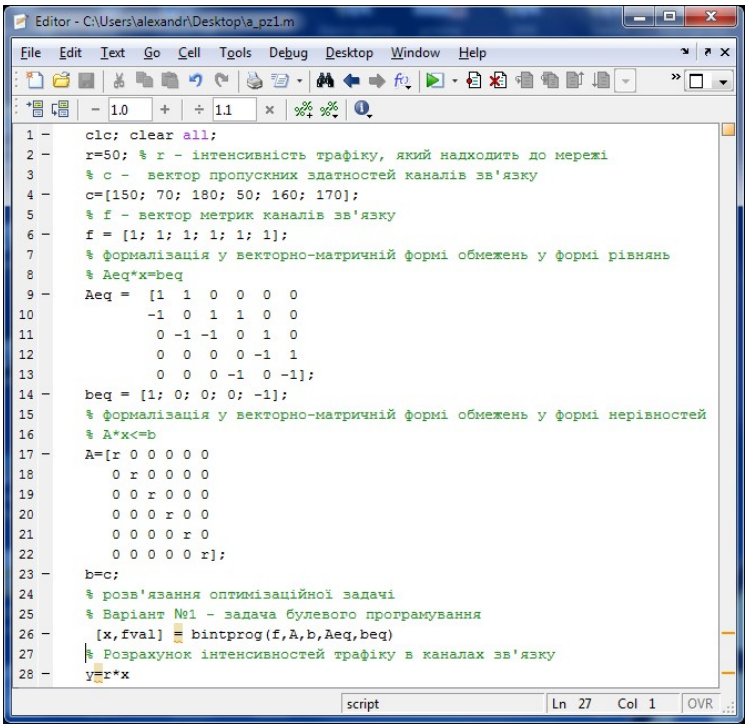

Figure 6: Program code in the Matlab file editor window.

According to these results, the "length" of the shortest path is 2 (fval = 2), with $x_{1,2} = 1$ and $x_{2,5} = 1$, indicating that it passes through nodes 1, 2, and 5 (Figure 7). Figure 7 illustrates the traffic intensity passing through the communication channel.

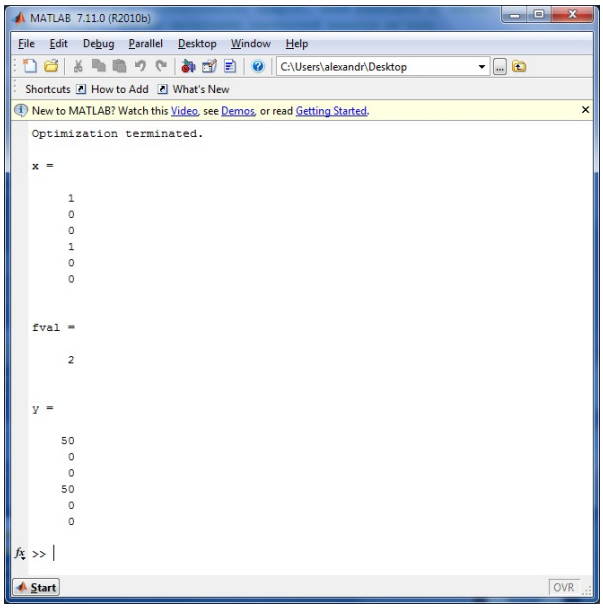

Figure 7: Calculation results in the command window.

As a result, it is necessary to construct a graph showing the number of paths (K) used in the routing process as a function of the traffic intensity (r) entering the network. The graph should also indicate which paths are used at which traffic intensities. Additionally, it is worthwhile to experimentally determine the bandwidth capacity in the communication direction from the sender node to the receiver node according to the implemented routing model.

The implementation of the described routing model using the Matlab mathematical package and developed technology has demonstrated analogous routing results.

5. Impact of Machine Learning on SDN/NFV-based Routing

The integration of Machine Learning (ML) with Software-Defined Networking (SDN) and Network Functions Virtualization (NFV) is revolutionizing network routing strategies. This synergy enhances network performance, adaptability, and efficiency. This section explores the transformative role of ML in SDN/NFV-based routing systems.

5.1. Machine Learning Algorithms in Network Routing

Several ML algorithms have shown promise in optimizing routing decisions:

1. Reinforcement Learning (RL) algorithms, particularly Q-learning and Deep Q-Networks (DQN), have been applied to dynamic routing problems. These algorithms learn optimal routing policies through trial and error, adapting to changing network conditions.
2. Deep Neural Networks (DNNs) and Convolutional Neural Networks (CNNs) are used for traffic prediction and classification, enabling proactive routing decisions.
3. Support Vector Machines (SVMs) have been employed for traffic classification and anomaly detection, enhancing the security aspects of routing.
4. Genetic Algorithms are used for optimizing routing paths in complex network topologies, especially in multi-objective routing scenarios.

5.2. ML-Enhanced SDN Controllers

ML algorithms integrated into SDN controllers can significantly improve routing efficiency:

1. Traffic Prediction: ML models analyze historical traffic data to predict future network loads, allowing for preemptive routing adjustments.
2. Adaptive Flow Management: ML algorithms dynamically adjust flow rules based on real-time network conditions, optimizing resource utilization.
3. Intelligent Load Balancing: ML-driven load balancing algorithms distribute traffic more efficiently across available paths, reducing congestion and improving overall network performance.

5.3. NFV Optimization through Machine Learning

In the context of NFV, ML contributes to more efficient routing by:

1. ML algorithms optimize the placement of Virtual Network Functions (VNFs) based on predicted traffic patterns and resource availability.
2. ML models determine the most efficient sequence of VNFs for processing network traffic, minimizing latency and maximizing throughput.
3. Predictive ML models assist in allocating computational resources to VNFs, ensuring optimal performance while minimizing resource waste.

5.4. Challenges and Future Directions

While ML shows great promise in enhancing SDN/NFV-based routing, several challenges remain:

1. Computational Overhead: The integration of ML algorithms can introduce additional computational load, potentially affecting real-time performance.

2. Data Quality and Availability: The effectiveness of ML models heavily depends on the quality and quantity of available network data.
3. Model Interpretability: Some ML models, particularly deep learning models, lack interpretability, which can be a concern in critical network operations.
4. Security Implications: The use of ML in routing decisions introduces new attack vectors that need to be addressed.

Future research should focus on developing lightweight ML algorithms suitable for real-time network operations, improving the interpretability of ML models in networking contexts, and addressing the security challenges associated with ML-driven routing systems.

The integration of Machine Learning with SDN/NFV-based routing represents a significant advancement in network management and optimization. By leveraging ML algorithms, networks can become more adaptive, efficient, and intelligent in their routing decisions. As this field continues to evolve, we can expect to see increasingly sophisticated routing solutions that can handle the complex demands of modern network environments. The synergy between ML, SDN, and NFV is paving the way for next-generation networks that are not only programmable and virtualized but also self-optimizing and predictive in nature.

6. Conclusions

In conclusion, it can be confidently asserted that the scientific and technical advancements in Software-Defined Networking (SDN) bring significant benefits to both data center enterprises and researchers exploring new network mechanisms and protocols. These advancements enable researchers to conduct more realistic experiments, while providing data center enterprises with tools to enhance their network infrastructures.

The emulation of network devices and their functionalities empowers stakeholders to develop and adapt their own standards, offering unparalleled flexibility in the design and management of corporate and global networks. This article has presented a methodological approach for developing and modifying network functions, accompanied by a robust testing environment for evaluating newly created dynamic routing protocols.

The study details a comprehensive modeling workflow, facilitating the validation of dynamic routing protocols implemented in Python across diverse network configurations involving any number of routers and interconnections. Furthermore, the implementation of a routing model based on the RIP protocol using Matlab's mathematical capabilities has corroborated the routing outcomes achieved through the innovative Python-based routing technology developed herein.

References

[1] Network routing: algorithms, protocols, and architectures / Medhi D., Ramasamy K. San Francisco: Kaufmann Publishers is an imprint of Elsevier, 2007. – 824 p.
[2] Software-defined networking (SDN): a survey. [web source] - Access mode: https://onlinelibrary.wiley.com/doi/epdf/10.1002/sec.1737
[3] Network Functions Virtualization (NFV). [web source] - Access mode: https://www.etsi.org/technologies/nfv
[4] Creating a simple router simulation using Python and sockets. [web source] - Mode of access:
[5] https://medium.com/swlh/creating-a-simple-router-simulation-using-python-and-sockets-d6017b441c09
[6] Exploring the Functions of Routing. [web source] - Access mode: https://www.learncisco.net/courses/icnd-1/lan-connections/functions-of-routing.html
[7] Use Container lab to emulate open-source routers. [web source] – Access mode: https://www.brianlinkletter.com/2021/05/use-containerlab-to-emulate-open-source-routers/

[8] Starkova, O., Herasymenko K., Nikolchev K., Bulgakova A. Virtualization And Programmability In Modern Networks In The Context Of SDN Concept. 2022 IEEE 4rd International Conference on Advanced Trends in Information Theory, ATIT 2022 – Proceedings, pp. 204 - 207.
[9] Starkova, O., Herasymenko K., Nikolchev K., Kravchenko O., Bulgakova A. Implementation Of Advanced Routing Methods Based On The SDN Concept And OS Linux. 2022 IEEE 4rd International Conference on Advanced Trends in Information Theory, ATIT 2022 – Proceedings, pp. 244 - 248.
[10] М.В. Семеняка, О.В. Лемешко, О.В. Старкова «Методичні вказівки до лабораторних робіт з дисципліни "Системи комутації та розподілу інформації. Частина 2"», Харків: ХНУРЕ, 2014. 92 с.
[11] M. V. Gundy, D. Balzarotti, G. Vigna, Catch me, if you can: Evading network signatures with web-based polymorphic worms, in: Proceedings of the first USENIX workshop on Offensive Technologies, WOOT '07, USENIX Association, Berkley, CA, 2007.
[12] D. Harel, LOGICS of Programs: AXIOMATICS and DESCRIPTIVE POWER, MIT Research Lab Technical Report TR-200, Massachusetts Institute of Technology, Cambridge, MA, 1978.
[13] K. L. Clarkson, Algorithms for Closest-Point Problems (Computational Geometry), Ph.D. thesis, Stanford University, Palo Alto, CA, 1985. UMI Order Number: AAT 8506171.
[14] D. A. Anisi, Optimal Motion Control of a Ground Vehicle, Master's thesis, Royal Institute of Technology (KTH), Stockholm, Sweden, 2003.
[15] H. Thornburg, Introduction to bayesian statistics, 2001. URL: http://ccrma.stanford.edu/jos/bayes/bayes.html.
[16] R. Ablamowicz, B. Fauser, Clifford: a maple 11 package for clifford algebra computations, version 11, 2007. URL: http://math.tntech.edu/rafal/cli11/index.html.
[17] Poker-Edge.Com, Stats and analysis, 2006. URL: http://www.poker-edge.com/stats.php.
[18] B. Obama, A more perfect union, Video, 2008. URL: http://video.google.com/videoplay?docid=6528042696351994555.
[19] D. Novak, Solder man, in: ACM SIGGRAPH 2003 Video Review on Animation theater Program: Part I - Vol. 145 (July 27–27, 2003), ACM Press, New York, NY, 2003, p. 4. URL: http://video.google.com/videoplay?docid=6528042696351994555. doi:99.9999/woot07-S422.
[20] N. Lee, Interview with bill kinder: January 13, 2005, Comput. Entertain. 3 (2005). doi:10.1145/1057270.1057278.
[21] J. Scientist, The fountain of youth, 2009. Patent No. 12345, Filed July 1st., 2008, Issued Aug. 9th., 2009.
[22] B. Rous, The enabling of digital libraries, Digital Libraries 12 (2008). To appear.
[23] M. Saeedi, M. S. Zamani, M. Sedighi, A library-based synthesis methodology for reversible logic, Microelectron. J. 41 (2010) 185–194.
[24] M. Saeedi, M. S. Zamani, M. Sedighi, Z. Sasanian, Synthesis of reversible circuit using cycle-based approach, J. Emerg. Technol. Comput. Syst. 6 (2010).
[25] M. Kirschmer, J. Voight, Algorithmic enumeration of ideal classes for quaternion orders, SIAM J. Comput. 39 (2010) 1714–1747. URL: http://dx.doi.org/10.1137/080734467. doi:10.1137/080734467.
[26] L. Hörmander, The analysis of linear partial differential operators. IV, volume 275 of Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences], Springer-Verlag, Berlin, Germany, 1985. Fourier integral operators.
[27] L. H¨ormander, The analysis of linear partial differential operators. III, volume 275 of Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences], Springer-Verlag, Berlin, Germany, 1985. Pseudodierential operators.
[28] IEEE, Ieee tcsc executive committee, in: Proceedings of the IEEE International Conference on Web Services, ICWS '04, IEEE Computer Society, Washington, DC, USA, 2004, pp. 21–22. doi:10.1109/ICWS.2004.64.

[29] TUG, Institutional members of the TEX users group, 2017. URL: http://www.tug.org/instmem.html.
[30] R Core Team, R: A language and environment for statistical computing, 2019. URL: https://www.R-project.org/.
[31] S. Anzaroot, A. McCallum, UMass citation field extraction dataset, 2013. URL: http://www.iesl.cs.umass.edu/data/data-umasscitationfield.