electronics

MDPI

Article

M-Emu: A Platform for Multicast Emulation

Zhenyu Tian [1,2], Jiali You [1,2,3,*] and Hong Ni [1,2]

[1] National Network New Media Engineering Research Center, Institute of Acoustics, Chinese Academy of Sciences, Beijing 100190, China; tianzy@dsp.ac.cn (Z.T.); nih@dsp.ac.cn (H.N.)
[2] School of Electronic, Electrical and Communication Engineering, University of Chinese Academy of Sciences, Beijing 100049, China
[3] Peng Cheng Laboratory, Shenzhen 518055, China
* Correspondence: youjl@dsp.ac.cn

Abstract: Network layer multicast research is an important field of network research that requires simulators or emulators to support Software-Defined Networking (SDN) as well as to provide a specific structure at the network layer to facilitate packet forwarding, such as a multicast tree. The existing emulation platforms cannot effectively support the emulation of certain key multicast technologies, such as the Grafting Point (GP)-selection method and Rendezvous Point (RP)-selection method, for the following reasons: First, the programmable data plane of the existing emulation platform has many defects, such as the inability to process packet scheduling tasks, the prohibition of dynamic memory allocation and loops with unknown iteration counts, which make it difficult to deploy complex multicast protocols and algorithms. Secondly, at present, no emulation platform integrates network layer multicast emulation functions. As a result, users need to develop the multicast tree construction and maintenance mechanism in advance, which makes experiments laborious. To solve the above problems, based on NS4, we designed a multicast emulation platform, M-Emu. M-Emu presents a Service-Forwarding Architecture, which enables the data plane to deploy arbitrary complex protocols and algorithms. Based on the Service-Forwarding Architecture, M-Emu integrates a Multicast-Emulation Framework, which has a complete multicast tree construction and maintenance mechanism. We explain in detail how the various parts of M-Emu cooperate to complete the multicast emulation with an example and prove that M-Emu is efficient in CPU and memory consumption, etc., through a large number of experiments.

Keywords: multicast; emulator; efficient; framework; interfaces

Citation: Tian, Z.; You, J.; Ni, H. M-Emu: A Platform for Multicast Emulation. *Electronics* **2022**, *11*, 1152. https://doi.org/10.3390/electronics11071152

Academic Editor: Rashid Mehmood

Received: 3 March 2022
Accepted: 31 March 2022
Published: 6 April 2022

1. Introduction

Compared with application layer multicast or overlay multicast, network layer multicast has clear advantages in multicast tree cost, end-to-end delay, control overhead, etc., and thus it has always been an important research topic [1,2]. Recently, increasingly network layer multicast solutions have been designed based on SDN, which can effectively reduce the cost of creating and maintaining multicast trees and improve the efficiency of multicast group member management, etc. [3,4]. Most network layer multicasts create state information per multicast distribution tree in the network [3–5]; therefore, the research of network layer multicasting requires simulators or emulators to support SDN, as well as provide a specific structure at the network layer to facilitate packet forwarding, such as a multicast tree.

At present, the emulation platforms supporting SDN are mainly driven by Openflow and P4 [6–8]. Compared with OpenFlow, P4 can generalize the match-action framework to parsing–deparsing arbitrary header fields, flexible lookup tables with rich semantics, configurable control flow and platform-specific extensions. Based on these features, P4 has met with increasing enthusiasm [9–11]. NS4 is a state-of-the-art emulation platform driven by P4.

Although NS4 can effectively simulate a P4 network, it is difficult for NS4 to meet some specific requirements of multicast emulation due to some deficiencies of its architecture and functions. For example: (1) The design of P4 does not take packet scheduling into account, which makes it difficult for NS4 to deal with strict priorities, weighted fair queue, etc. However, in multicast systems, packet scheduling will greatly affect the allocation of network resources and the quality of multicast service, which is a major direction of multicast research.

(2) Different from imperative languages, such as C++, P4 is not a Turing-complete language. P4 does not support dynamic memory allocation, resulting in the failure of NS4's data plane to apply basic data structures, such as linked lists. Furthermore, P4 forbids loops with unknown iteration counts, which makes it difficult for the data plane to independently implement certain periodic operations, such as timeout verification and neighboring information detection. Although this feature of P4 can guarantee the delay of packet processing on the switch in real network, the research of multicast algorithms should not be limited by such engineering design. In addition, some network architectures require the data plane to have the ability to deploy applications [12–14] or respond to network requests without interacting with the control plane [15], which requires the data plane to be able to deal with arbitrary complex algorithms.

(3) NS4 does not have a complete multicast tree construction and maintenance mechanism, nor does it provide multicast algorithm interfaces, such as a Rendezvous Point (RP) selection method [16–20] interface, Grafting Point (GP) selection method [21–24] interface, etc., which increases the workload of emulations.

To solve the above problems, we designed and developed M-Emu based on NS4. The code is in https://gitee.com/T1anzhEnyu/multicast-emulation.git (accessed on 2 March 2022). In order to improve the architecture defects of NS4, M-Emu presents a Service-Forwarding Architecture. The data plane of the architecture consists of the forwarding module and service module. The forwarding module can forward packets based on the flow tables. The service module can independently formulate and execute various algorithms according to the network status information, such as the network load status and cache queue status. As the service module can create and monitor the cache queue, the data plane of NS4 can process packet scheduling tasks.

The service module is written in C++, which is a Turing-complete language and is capable of any Turing computable function. Thus, C++ enables the data plane of NS4 to deploy arbitrary complex multicast protocols and algorithms. In order to improve the functional deficiencies, based on the Service-Forwarding Architecture, M-Emu integrates a Multicast-Emulation Framework. The Multicast-Emulation Framework has a complete multicast tree construction and maintenance mechanism, which significantly reduces the emulation complexity of multicast technology. In addition, we deploy statistical tools for common multicast performance metrics.

We take the emulation of Protocol-Independent Multicast Sparse Mode (PIM-SM) as an example to explain in detail how the various parts of M-Emu cooperate to complete the multicast emulation, and we prove that M-Emu is efficient in CPU and memory consumption, etc., through a large number of experiments. The notations used in this article are shown in Table 1.

The contributions of this paper are as follows:

1. M-Emu presents a Service-Forwarding Architecture, which enables the data plane to process packet scheduling tasks and to deploy arbitrary complex multicast protocols and algorithms.
2. Based on the Service-Forwarding Architecture, M-Emu integrates a Multicast-Emulation Framework. The Multicast-Emulation Framework has a complete multicast tree construction and maintenance mechanism, which can reduce the workload emulations.
3. We deploy statistical tools for common multicast performance metrics.
4. The effectiveness and efficiency of M-Emu is analyzed theoretically and experimentally.

This paper is organized as follows. In Section 2, we present research related to our work. In Section 3, we introduce the implementation of M-Emu. In Section 4, we explore the effectiveness and efficiency of M-Emu theoretically and experimentally. In Section 5, we conclude the paper.

Table 1. Notations and definitions.

Notation	Definition
GP	Grafting Point
RP	Rendezvous Point
BP	Branching point
cRPs	candidate RPs
SBT	Source Based Tree
ST	Shared Tree
MFT	Multicast Forwarding Table
DR	Designated Router
NRS	Name Resolution System
NA	Network Address
SDN	Software-Defined Networking
Emu-node	The gateway of the emulation network and the real network.

2. Related Work

The need for simulators or emulators for SDN-based networks has already attracted research efforts. Each of them is equipped with different characteristics for their application scenarios. We investigated some well-known emulators and analyzed their features as follows.

ns-3 is a discrete-event driven network emulator. It is designed to meet academic and teaching needs, primarily in the research and education fields. The architecture of the simulated network in ns-3 is similar to that of the Open System Interconnection Reference Model (OSI model). ns-3 encapsulates data in the order of device layer, network layer, transport layer, API sockets and application layer. The data transfer process in ns-3 is similar to a physical network, making the simulation results of ns-3 accurate. ns-3's fd-net-device enables communication with a physical network by reading and writing traffic from file descriptors. ns-3's data plane is powered by OpenFlow, making ns-3 limited to a strict set of header fields. Furthermore, ns-3 does not provide a network layer multicast tree construction and maintenance mechanism [25,26].

Mininet can develop and validate various protocols based on the OpenFlow and OpenVswitch models. The applications of Mininet can be well migrated to hardware devices. It uses lightweight virtualization technology to make the system comparable to a real network. Mininet makes it easy to create a network that supports SDN. With Mininet, new features can be flexibly added to the network and tested and then easily deployed in real hardware environments. However, Mininet's CPU cycles are shared between hosts, switches and controllers, and the CPU scheduler cannot accurately control the scheduling sequence. As a result, Mininet's simulation results are not sufficiently accurate and are difficult to reproduce [27].

EstiNet is a commercial OpenFlow network emulator that was developed by a Taiwanese SDN solution vendor. EstiNet is capable of emulating thousands of OpenFlow-enabled switches. EstiNet has devised a method called Kernel re-entering, which enables each simulation node to run the linux operating system, causing applications of EstiNet to be well migrated to hardware devices with minor modifications. EstiNet's GUI is easy to use and can dynamically display the operation state of the simulation network, which is helpful for analyzing simulation results. However, EstiNet is powered by OpenFlow, making EstiNet is limited to a strict set of header fields. Furthermore, EstiNet does not provide a network layer multicast tree construction and maintenance mechanism [28,29].

OMNeT++ is a free multi-protocol discrete-event simulation software. It is not specifically designed for network systems and can simulate any system of components that pass information to each other. The software has many useful functions, such as traffic modeling between communication networks, protocol simulation, queuing networks and complex networks. OMNeT++ has good parallel simulation capabilities. However, the architecture of OMNet++ is not similar to that of the OSI network, thus, affecting the accuracy of the simulation results. However, OMNeT++ is powered by OpenFlow, making OMNeT++ limited to a strict set of header fields. Furthermore, OMNeT++ does not provide a network layer multicast tree construction and maintenance mechanism [30,31].

NS4 is a P4-driven network simulator [6]. NS4 integrates a P4 behavioral model in ns-3. It contains a control plane and a programmable data plane, which provides an accurate simulation of a P4-driven network system. NS4 is the first research effort to enable the simulation of a P4-enabled network, providing a useful tool for P4 research and development. Although NS4 has some deficiencies of its architecture and functions, NS4 effectively combines the advantages of P4 and ns-3 and is currently the best foundation for the development of a multicast simulation platform.

The characteristics of all simulation platforms are summarized in Table 2. ns-3 and NS4 are superior in terms of accuracy among all the simulators or emulators. NS4 integrates P4 on the basis of ns-3 to simulate a P4 network, which makes NS4 a state-of-the-art emulator. However, as we analyzed in Section 1, there are many problems in the data plane of NS4, resulting in difficulties to support the simulation of a network layer multicast. We argue that M-Emu comes to fill the gap in NS4 for network layer multicast research.

Table 2. Comparison of emulators.

Emulators	Protocol Independent	Accuracy	Multicast Function
ns-3	No	High	No
Mininet	No	Medium	No
EstiNet	No	High	No
OMNet++	No	Medium	No
NS4	Yes	High	No

3. Implementation

In this section, we introduce the implementation of M-Emu. M-Emu presents a Service-Forwarding Architecture, which enables the data plane to process packet scheduling tasks and to deploy arbitrary complex multicast protocols and algorithms. Based on the Service-Forwarding Architecture, M-Emu integrates a Multicast-Emulation Framework. The Multicast-Emulation Framework has a complete multicast tree construction and maintenance mechanism. In addition, we deployed data collection and analysis capabilities for common multicast performance metrics.

3.1. Service-Forwarding Architecture

The Service-Forwarding Architecture contains a control plane and a data plane. As for the control plane, M-Emu adds basic function module and algorithm module on the basis of the existing functions of NS4's control plane. The data plane of M-Emu is divided into a forwarding module and service module. The forwarding module contains all the functions of the NS4's data plane and adds the function of interacting with the service module. The service module is the core of M-Emu's Service-Forwarding Architecture, which enables the data plane of M-Emu to deal with complex multicast protocols and algorithms. The architecture diagram is shown in Figure 1.

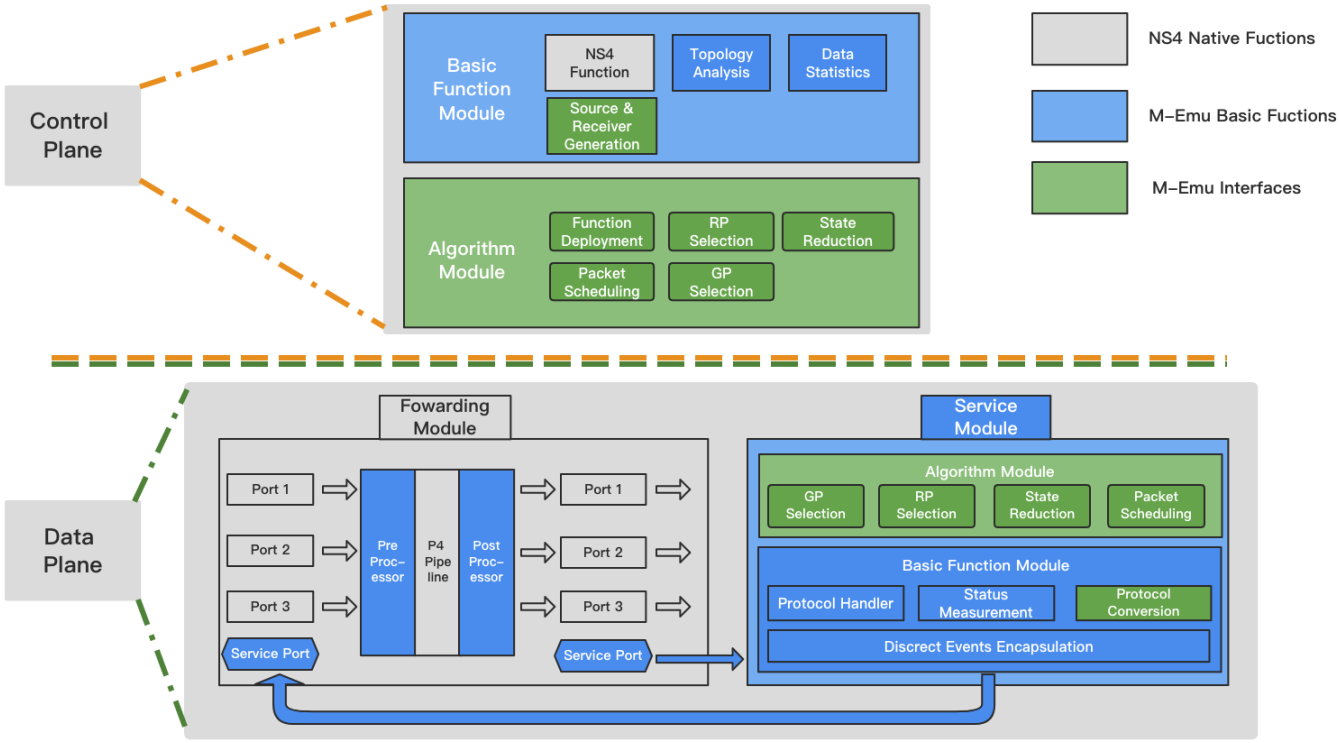

Figure 1. Service-forwarding architecture.

3.1.1. Control Plane

The control plane is divided into a basic function module and algorithm module. In the basic function module, the topology analysis module analyzes the centricity, connectivity and clustering attributes of a topology. The data statistics module calculates the load of all nodes in the network in a certain period and records it in a text file in a certain format. The source and receiver generation module provides an interface that can determine the request number that the Designated Router (DR) receives from sources and receivers. Users can flexibly modify the distribution of the request number according to the experimental requirements.

The algorithm module of the control plane mainly provides the interface of functional node deployment and some multicast algorithm interfaces. The functional node deployment interface can determine the number and location of some functional nodes, such as the RP and name resolution system nodes (NRS). The GP selection module, RP selection module and other algorithm interfaces can use global information to run in a centralized paradigm.

3.1.2. Data Plane

The data plane includes a forwarding module and service module. The forwarding module is responsible for forwarding packets. In the existing NS4's data plane, we added service ports. These ports are identified with a specific port number and bound to two queues, which are used to cache the packets entering or leaving the service module. The preprocessing module and post-processing module are used to process the data packets before or after the P4 pipeline. Their main function is to determine whether the data packets are entering or leaving the service module.

The service module of the data plane is the module that formulates and executes the algorithms. The service module mainly includes algorithm module and basic function module. The algorithm module includes the GP selection module, RP selection module, multicast state reduction module and packet scheduling module. Users can modify the methods in these modules to flexibly deploy experiments. The basic function module

includes a protocol handler module, network status measurements module, protocol conversion module and the discrete event encapsulation module.

The protocol handler module is responsible for the construction and maintenance of the multicast path. The status management module mainly manages and maintains the multicast forwarding table (MFT) according to the multicast protocol message. This module also manages the load information of the node and its neighboring domains.

The protocol conversion module is responsible for the protocol conversion between the emulation network and real network. The advantage of this module is that it can avoid modifying the protocol of emulation network or real network and, thus, significantly reduce the experiment workload. The discrete event encapsulation module encapsulates the operations of the service module into discrete events. One of the core concepts of ns-3 is to model the packet forwarding processing in the network system as discrete events occurring in a certain sequence. Encapsulating the actions of the service module as discrete events is in line with the design concept of ns-3.

3.1.3. Analysis of Service-Forwarding Architecture

M-Emu's Service-Forwarding Architecture effectively solves the deficiency of NS4 architecture. The service module can create and monitor the cache queue so that the data plane of NS4 can process multiple packets at the same time, thus, supporting the emulation of packet scheduling. The algorithm of the service module is written in C++, which overcomes the deficiency of p4 language and enables the data plane of NS4 to deploy complex multicast protocols and algorithms. The Service-Forwarding Architecture provides sufficient centralized algorithm interfaces on the control plane and distributed algorithm interfaces on the data plane, which reduces the workload of emulations.

3.2. Multicast-Emulation Framework

Many IP multicast solutions need to maintain multicast trees, such as PIM-SM, PIM-SSM and DVMRP. Therefore, the construction and maintenance mechanism of multicast trees is important for multicast emulation. According to the type of multicast tree established, multicast routing protocols can be categorized as a Source-Based Tree (SBT) or Shared Tree (ST)-based [32,33]. Both SBT and ST use the shortest paths between the source or core and receivers to form the multicast tree. In order to enable M-Emu to build SBT and ST, we designed the Multicast-Emulation Framework based on ILDM. This framework enables M-Emu to not only construct ST and SBT but also to flexibly adjust the position of multicast tree root points and grafting points, thus supporting more complex multicast tree structures. Therefore, the Multicast-Emulation Framework makes M-Emu highly versatile.

3.2.1. Introduction to Key Elements

The architecture involves a name resolution system (NRS), Designated Router (DR)s, Rendezvous Points (RPs), data sources, data receivers, multicast processing nodes and so on. NRS consists of multiple switches with the name resolution function. NRS uses globally unique names to identify multicast services and maintains the mapping between identifiers and network addresses (NAs) of one or more multicast processing nodes. DR is responsible for processing requests from data sources and receivers and performs RP and GP-selection methods to determine the locations of RP and GP for the multicast tree.

The RP in this framework is similar to the RP in PIM-SM. It is a set of pre-arranged nodes that can receive data from DRs and provide the data distribution service as the root node of the multicast tree. All multicast processing nodes maintain an MFT. Each entry in the MFT corresponds to a forwarding rule of a multicast group. The key of the entry is the ID of the multicast service, and the value is the NA list of the next-hops. Each node in the multicast distribution tree can register the mapping between the multicast service ID and its NA in NRS. Emu-nodes are the gateway of the emulation network and the real network, which is used to complete the data exchange between the real network and the emulation network.

3.2.2. Construction and Maintenance Mechanism of Multicast Tree

As shown in Figure 2, the controller can deploy RP, DR, NRS and Emu-nodes in the network according to the logic of its function deployment module. All RPs register their NAs on NRS with the same specific identity. The data source generates the data and the multicast service ID and sends it to DR (arrow 0). After receiving the data from the source, DR sends the RP request (arrow 1) to NRS, and NRS replies with the RP NAs list (arrow 2). DR selects an RP as the root of the multicast distribution tree according to the RP-selection method and sends data to the selected RP (arrow 3). Our Multicast-Emulation Framework considers that no multicast transmission path is constructed between the source-side DR and RP, and thus the data is transmitted through unicast in this stage.

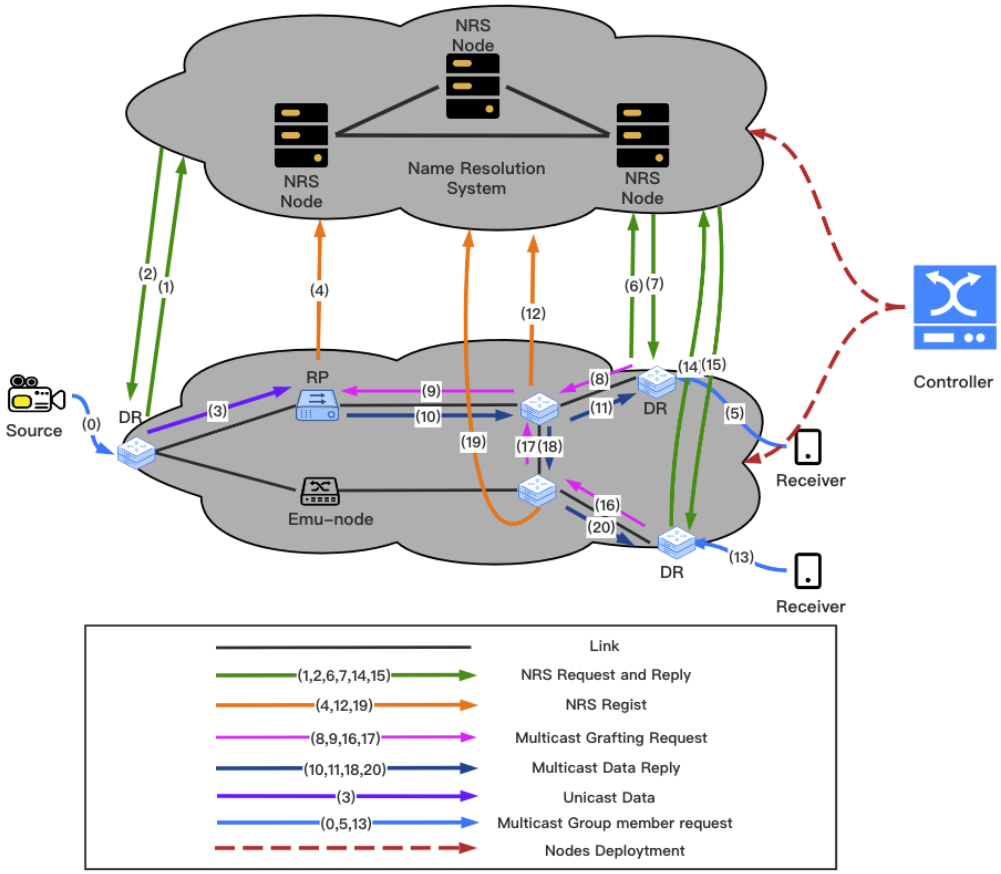

Figure 2. Multicast-Emulation Framework.

After receiving data from the source-side DR, the RP adds an entry to the MFT for the multicast service. The RP needs to register the mapping between the multicast service ID and its NA to NRS (arrow 4). After receiving data requests from the receivers (arrows 5–6), the receiver-side DR requests the NAs of the nodes in the multicast distribution tree from NRS based on the multicast service ID. After receiving the NA list replied to by NRS, DR selects an appropriate GP according to the GP-selection algorithm. DR sends a joining request (arrows 8, 9, 16 and 17) to the selected GP. After receiving the joining request, GP sends the Multicast Data Reply to the receiver-side DR (arrows 10, 11, 18 and 20).

Each node on the forwarding path determines whether to add an entry for the multicast service. If the node creates an entry in MFT for the multicast service, the node also needs to add its NA to the entry of the previous hop node. Furthermore, the node needs to register the mapping between its NA and the multicast service ID to NRS (arrow 12 and arrow 19). If the node does not create an entry in MFT for the multicast service, the node only needs to forward the multicast data to the receiver side DR.

Multicast Data Reply contains the DR NA and the next Branching Point (BP) NA. Next BP refers to the node that is reachable for GP through the same outgoing interface as the receiver-side DR. If a node found that the unicast paths of the receiver-side DR and the next BP diverge, the node becomes a new branch node for the multicast tree. This mechanism is essentially the same as NBM [34].

3.3. Statistics Tools

To help users analyze the performance of multicast algorithms, we developed some statistical tools that can conveniently calculate various common multicast evaluation metrics. When each node receives or forwards packets, it records the packet information in memory according to the format shown in Figure 3.

Figure 3. The packet information format.

Self ID is the ID of the current node. The link ID is the ID of the link through which the packet enters the current node. The Src Node ID indicates the ID of the node that sends the packet. The Dst node ID indicates the ID of the destination node of the data packet. The timestamp refers to the time when the packet is forwarded or received by the current node.

The data statistics module of the control plane periodically records the packet information and MFT of each node into the text file as the raw data. M-Emu provides analysis scripts for these raw data to generate some common multicast evaluation metrics, such as:

(1) The MFT load. The MFT load of each node is obtained by calculating the MFT size of each node.
(2) The Link load. With the link ID as the index, the amount of data passing through the link throughout the experiment can be obtained. In addition, the load changes of links in different time periods can be analyzed according to time stamps.
(3) The Multicast tree cost. With the multicast ID as the index, all the links that the multicast tree passes through can be obtained, and then the cost of the entire multicast tree can be obtained.
(4) The End-to-end delay. According to the multicast ID and packet type, the time when the DR sends data and the time when the DR receives data can be counted, and then the end-to-end transmission delay can be calculated.

In this form, most of the user's statistical needs can be satisfied. Users can add some specific information to the existing package information to meet their personalized needs.

3.4. Statement of Main Interfaces

M-Emu is still under development. The main interfaces that we currently open include:

- DR and cRP deployment function. The location of the DR determines the distribution of data sources and receivers in the network. The number and position of cRPs determine the position of multicast root nodes [1–3].
- RP-selection algorithm. This algorithm determines how the source-side DR selects the RP from the cRPs set [16–20,33].
- GP-selection algorithm. The algorithm determines how the receiver-side DR selects a node from the multicast tree to join [21–23].
- State reduction algorithm. The algorithm determines which nodes on the multicast path can create MFT entries [24,34–36].
- Statistical tools. These are used to calculate the network status and evaluate the advantages and disadvantages of each algorithm.

We elaborate on how these interfaces are used in Section 4, using the deployment of PIM-SM as an example.

4. Validation

In this part, we explore the effectiveness and efficiency of M-Emu. First, we take PIM-SM emulation as an example to introduce how M-Emu can effectively support the emulation of various multicast technologies. Secondly, we compare the differences between M-Emu and the original NS4 in terms of CPU usage and memory usage, etc., when deploying the PIM-SM emulation.

4.1. Case Study

We take the emulation of PIM-SM as an example to explain in detail how the various parts of M-Emu cooperate to complete the multicast emulation. We emulate PIM-SM in a fat tree topology, which is shown in Figure 4.

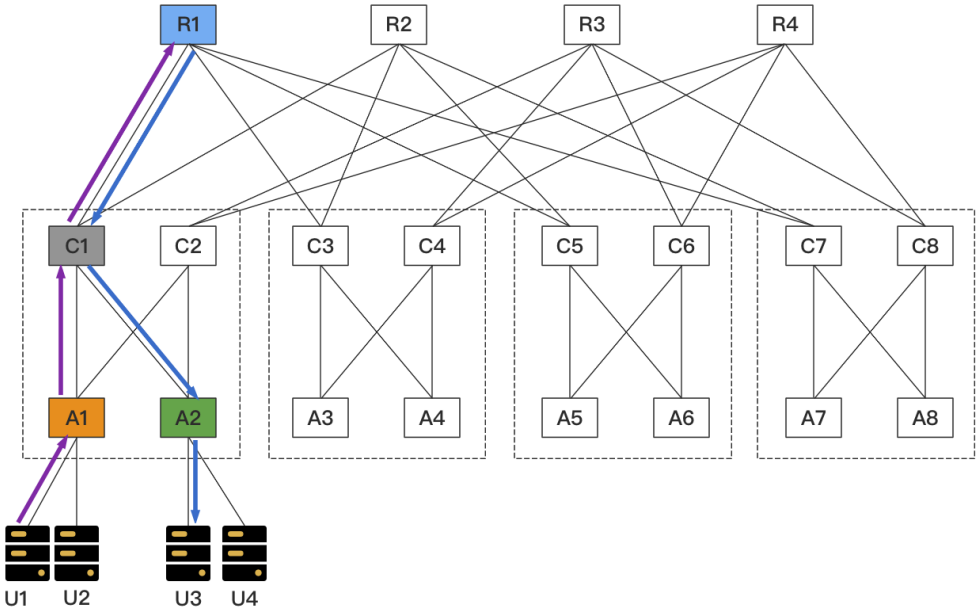

Figure 4. PIM-SM Emulation.

PIM-SM is the most popular IP multicast at present. The multicast tree of PIM-SM is a Shared Tree. Each source DR selects a node from the cRPs set as the RP. Each receiver-side DR then selects this RP as GP. First, we need to deploy the DRs. M-Emu provides the DR deployment interface, which is

```
void SetDR(
std::map<std::string,Ptr>nodeMap,
const NodeContainer nodes,
NodeContainer* senderDR,
NodeContainer* receiverDR)
```

Among the parameters of this interface, *std::map<std::string, Ptr>nodeMap* maintains the mapping between the node ID in the topology file and the node ID in the emulated network. A topology file is a file provided by users that describes the topology. The emulators or simulators can build a topology based on this file. The *nodeMap* is adopted because these two IDs are often inconsistent. *const NodeContainer nodes* contains all nodes in the topology. *NodeContainer* senderDR* and *NodeContainer* receiverDR*, respectively, store the pointers of the source-side DRs and receiver-side DRs. In this example, we add A1 to *senderDR* and A2 to *receiverDR*. The simplest way to do this is to find the IDs of A1 and A2 from the topology file, then use the *nodeMap* to find the node pointers of A1 and A2, and finally add them to the corresponding *nodeContainer*.

Second, we deploy the cRPs. M-Emu provides the RP deployment interface on the control plane, which is

```
void SetRP(
std::Map<std::string, Ptr>nodeMap,
const NodeContainer Nodes,
NodeContainer * cRPs)
```

NodeContainer *RP contains pointers to all cRPs. In this example, we add R1, R2, R3 and R4 to this nodeContainer.

When A1 receives data from U1 or U2, A1 runs the RP-selection method to select a node from the cRP set as the RP. M-Emu provides the RP-selection method interface on both the control plane and data plane, which is

```
uint32_t GetRootNode(
const NodeContainer cRPs,
uint8_t * buffer, void* info)
```

The return value of this interface is the ID of the RP we selected. *uint8_t* buffer* holds the multicast data. *void* info* contains external information on which the algorithm depends. The control plane can add some global information to *info*, while the data plane can add only local information to *info*. We approximate that PIM-SM selects the RP using the hash of the multicast ID, which is provided in *uint8_t* buffer*. In this example, we assume that A1 selects R1 as RP for the current multicast group.

After receiving the multicast data, R1 will register the mapping between its NA and multicast ID with NRS, as described in the previous section. At some point, A2 will receive a data request from U3. A2 runs the GP-selection method to select a node from the multicast tree as GP. M-Emu provides a GP-selection algorithm interface on both the control plane and data plane, which is

```
uint32_t GetGraftNode(
uint8_t * buffer, void* info)
```

The return value of this interface is the ID of GP. The difference between the control plane and the data plane of the interface is still reflected in the content of the *info*. In this example, the DR on the receiver side selects the root node as GP each time. To find the address of the root node, the DR queries the address list of nodes in the multicast tree from the NRS based on the multicast ID in *uint8_t * buffer*. The first node in the list is the root node, because in every multicast tree, the first node to register with the NRS must be the root node.

When R1 receives a joining request from A2, it sends multicast data back to A2. When the multicast data passes through C1, C1 determines whether to create the MFT entry based on the State Reduction method. M-Emu provides the interface on both the control plane and data plane, which is

```
bool StateReduction(
uint8_t* buffer, void *info)
```

If the return value of this interface is 0, the node should not register MFT entries. In this example, each node should register the MFT entry; therefore, when a node runs this interface, the return value is 1. The whole process of the emulation is shown in Figure 5.

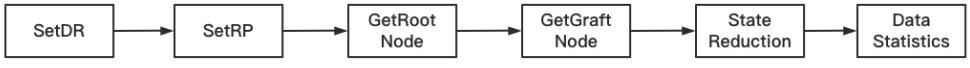

Figure 5. The process of emulation of PIM-SM.

After the above process, a multicast tree can be established in PIM-SM mode. According to Section 3.3, the *Statistical tools* provided by M-Emu records data packet information in a certain format. Next, we give an example of how M-Emu records packet information.

After A1 selects R1 as its RP, it sends data to R1 by unicast. When R1 receives multicast data from A1, it records the data as Figure 6. The meaning of the data in *Packet Type*

can be found in Section 3.2. When C1 forwards the multicast data sent from R1 to A2, the multicast data is recorded as shown in Figure 7. When A2 receives multicast data from R1, it records the data as shown in Figure 8. It can be seen that *Statistical tools* can accurately and comprehensively record the forwarding status of multicast packets on the network.

Self ID	Link ID	Packet Type	Multicast ID	Src Node ID	Dst Node ID	Packet Size	Time Stamp
R1	C1–R1	Unicast	1	A1	R1	100bits	00:00:00:01

Figure 6. Packet-Information 1.

Self ID	Link ID	Packet Type	Multicast ID	Src Node ID	Dst Node ID	Packet Size	Time Stamp
C1	C1–R1	MulticastData	1	R1	A2	100bits	00:00:00:02

Figure 7. Packet-Information 2.

Self ID	Link ID	Packet Type	Multicast ID	Src Node ID	Dst Node ID	Packet Size	Time Stamp
A2	C1–A2	MulticastData	1	R1	A2	100bits	00:00:00:03

Figure 8. Packet-Information 3.

According to the data packets recorded by all nodes, we can find the following key information:

- MFT load: R1, C1, A1 and A2 all need to maintain the multicast forwarding state for this multicast group, and thus their MFT load is +1.
- Link Load: Links R1-C1, C1-A1 and C1-A2 all need to forward the multicast stream, and their load is 100 Kb/s.
- Multicast tree cost: The multicast tree forwards 100 bits of data through three links. Therefore, the Multicast tree cost is 300 bits.
- End-to-end delay: According to the time stamp of each data packet, the time of sending data from A1 to A2 is 5 ms.

In many cases, users want to emulate SBT rather than ST. At this point, the user simply adds each DR to the cRPs set and forces the DR to select itself as the root node. In addition, users can also adjust the RP-selection method and GP-selection method to build a more complex multicast tree.

4.2. Emulation Performance

In this part, we test the performance of M-Emu and original NS4 in various aspects when deploying the experiments in Section 4.1. We require the data plane to execute multicast algorithms independently without interaction with the control plane. The reason for such a setting is that many researches require the data plane to have the ability to independently formulate and execute various algorithms [12,13,15].

For NS4, to enable the data plane to handle multicast protocols and algorithms independently, a natural thought would be to add a host, which we can call a service node, directly connected to each switch. The service nodes of NS4 operate as the service module of M-Emu, while the switch nodes of NS4 operate as the forwarding module of M-Emu. This approach is a degradation of M-Emu because it requires the deployment of additional nodes, resulting in increased system resource usage. Our experiments are conducted on a Dell R740xd PowerEdge server with two Xeon(R) Gold 5218R CPUs and 300 GB RAM.

The evaluation metrics include:

- Memory Consumption. This metric indicates the memory size required by the emulation platform to process a certain number of multicast flows.
- CPU utilization. This metric indicates the CPU utilization by the emulation platform to process a certain number of multicast flows.

- Traffic of Management Message. This metric indicates the traffic of various management messages when the same number of multicast flows (1000) are completed.
- Packet Process time of Emu-Node. This metric indicates the time required for the emulation platform to interact with the real network.
- Execution Time. This metric indicates the time required by the emulation platform to process a certain number of multicast flows.

4.2.1. Memory Consumption

We compare the memory consumption of each architecture in fat tree topologies when processing 1000 multicast flows. The results are shown in Figure 9. It can be seen that, with the increase of K, which indicates the size of fat trees, the memory consumption of both architectures increases gradually. This is because both M-Emu and NS4 need to populate routing entries in every switch. The larger the topology is, the more memory the flow tables occupy. Furthermore, the memory consumption of M-Emu is about 60% of that of NS4, Thus, compared to NS4, M-Emu can save more memory. This is because NS4 requires service nodes, which makes the topology of NS4 larger and the memory consumption of the flow tables higher.

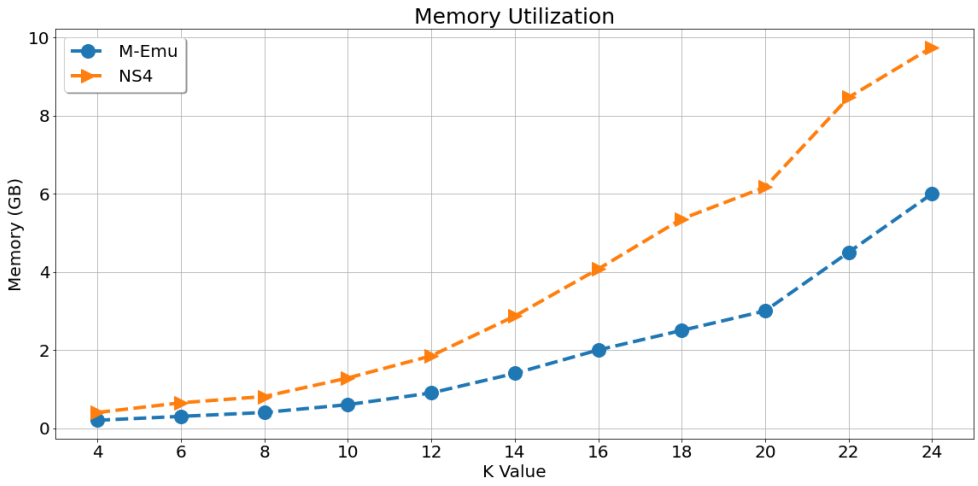

Figure 9. Memory consumption.

4.2.2. CPU Utilization

In this part, we compare the CPU consumption of each architecture in fat tree topologies when processing 1000 multicast flows. The results are shown in Figure 10. It can be seen that, with the increase of K, the CPU utilization of M-Emu rises slower than that of NS4. When K = 4, the CPU utilization of M-Emu was about 30% of that of NS4. This is because NS4 needs to forward packets from switches to service nodes, and it also needs to parse or encapsulate packets from the link layer to the application layer on the service nodes.

Both actions can be omitted in M-Emu. The service module and forwarding module of M-Emu are connected by queues, avoiding the addition of NetDevices and Channels in the network, which are two basic units of ns-3 or NS4. Furthermore, the service module in M-Emu can directly check specific fields of data packets according to offsets, which greatly improves the analysis efficiency. Therefore, the CPU usage of M-Emu is lower than that of NS4 when processing multicast protocols.

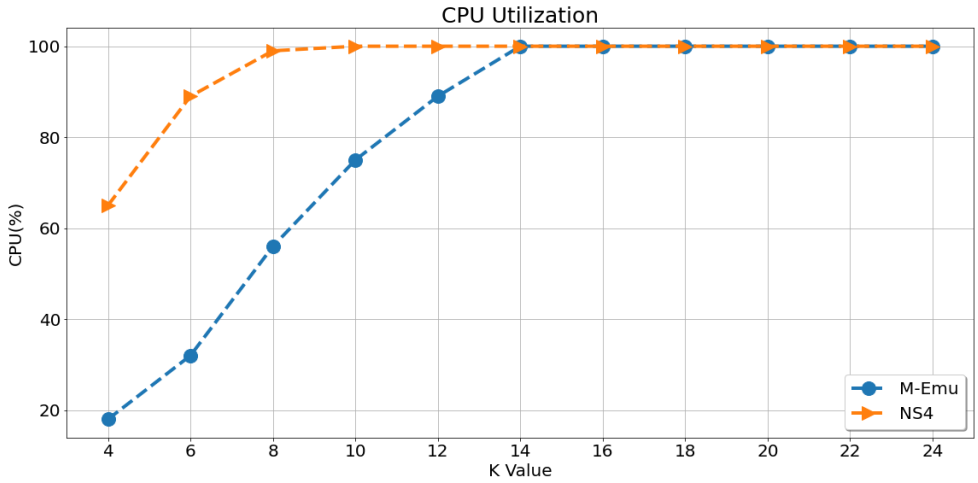

Figure 10. CPU utilization.

4.2.3. Traffic of Management message

In a fat tree topology with K = 6, we compare the management traffic required by each architecture when processing the same number of multicast flows. The management traffic refers to the traffic of multicast tree establishment and maintenance message. As can be seen from Figure 11, as the number of multicast flows increases, the management traffic of each architecture increases. The traffic of management messages of M-Emu is about 1/3 of that of NS4. This is because, for NS4, the traffic between service nodes and switch nodes accounts for about 2/3 of the total control management traffic, and this part of the traffic can be omitted in M-Emu.

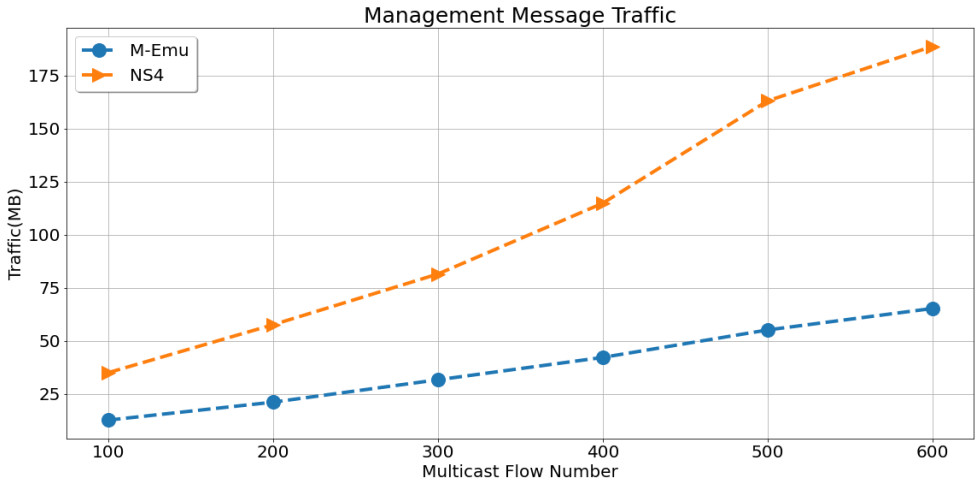

Figure 11. Traffic of the management message.

4.2.4. Packet Process Time of Emu-Node

Since the protocol conversion module of M-Emu needs to modify the header of each packet, the packet processing delay will increase slightly on Emu-node compared with NS4. We ran 1000 packet interactions with the real environment and compared the packet processing time of M-Emu with that of NS4. The results are shown in Figure 12. It can be seen that the modification of the packet header does increase the packet processing time slightly, by about 3% compared with NS4.

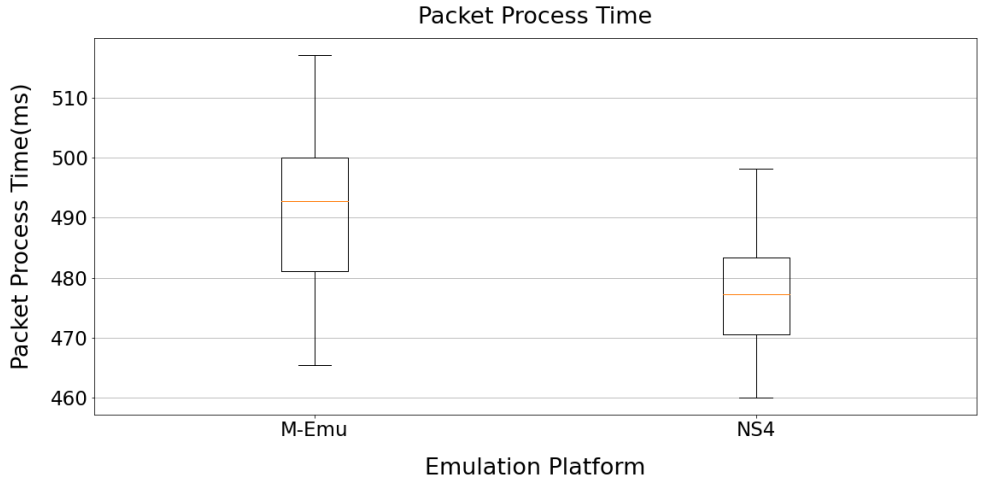

Figure 12. Packet process time.

4.2.5. Execution Time

In this part, we compare the execution time required by each architecture when processing the same number of multicast flows (1000 multicast flows). The execution time is related to many factors, such as the memory consumption, CPU utilization and so on; thus, it is a comprehensive factor to measure each architecture's performance. The results are shown in Figure 13. It can be seen that, with the increase of K, the execution time of these architectures increases gradually. The execution time of M-Emu is about 1/2 of that of NS4.

This is mainly because: (1) In NS4, the communication delay between service nodes and switches increases the complete time of a multicast flow, thus increase the total execution time. (2) The topology size of NS4 is almost twice as large as that of M-Emu. The larger the topology is, the more resources the experiment consumes and the slower the experiment runs. (3) The packet processing logic on the service node is more complex than that of M-Emu's service module. Thus, the Execution Time of NS4 is much larger than that of M-Emu.

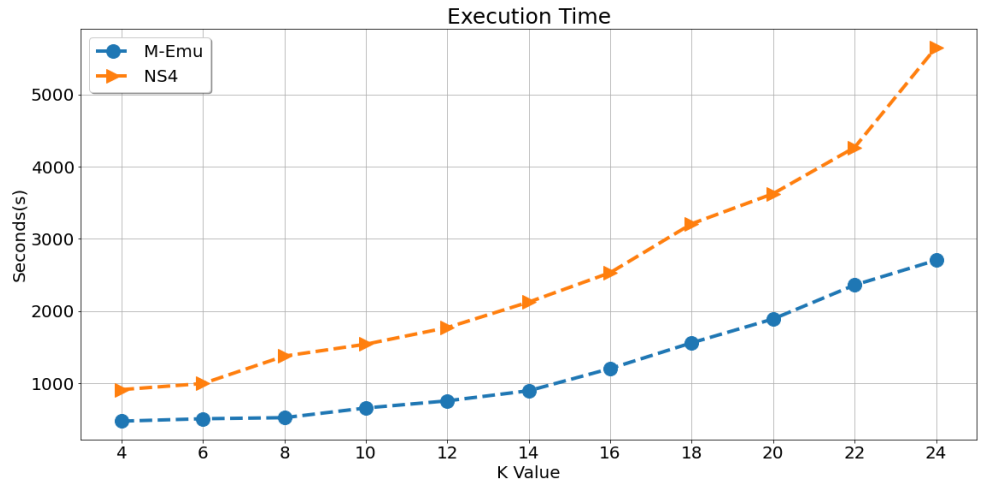

Figure 13. Execution time.

4.2.6. Code Complexity

In this section, we make a comparison between M-Emu and NS4 in terms of the code complexity (number of lines of code). Since M-Emu is an improvement on NS4, M-Emu has much more code on both the control plane and data plane than does NS4. The results are shown in Table 3.

Table 3. Code complexity.

Emulators	Control Plane	Forwarding Module	Service Module
NS4	360	265	0
M-Emu	1210	350	2045

We only counted the amount of code in the source file. It can be seen that the code quantity of M-Emu is larger than that of NS4 in all modules. In the forwarding module, because M-Emu only adds ports to the service module, there is not much more code than for NS4. As NS4 has no service module, the code quantity of its service module is 0. However, M-Emu has the multicast simulation function deployed in the service module, and thus the code quantity is much larger than NS4.

5. Conclusions

We designed and implemented a multicast emulation platform M-Emu based on NS4. M-Emu contains the Service-Forwarding Architecture, which enabled the data plane of M-Emu to process multicast scheduling tasks and deploy complex protocols and functions. A Multicast-Emulation Framework was deployed in M-Emu, which had a complete multicast tree construction and maintenance mechanism. The framework constructed all existing network layer multicast tree structures and flexibly adjusted the position of RPs and GPs to support more complex tree structures.

M-Emu mainly provides the DR and cRP deployment interface, GP-selection method interface, RP-selection method interface, state reduction interface, etc., which enable users to examine the multicast algorithm conveniently. We took the emulation of PIM-SM as an example to explain in detail how the various parts of M-Emu cooperate to complete the multicast emulation, and we proved that M-Emu is efficient in CPU utilization , memory consumption, etc., through a large number of experiments. In the future, we will further improve the function of the service module to provide support for the experiment of multicast security and reliability.

Author Contributions: Conceptualization, Z.T., J.Y. and H.N.; methodology, Z.T. and J.Y.; software, Z.T.; writing—original draft preparation, Z.T.; writing—review and editing, Z.T. and J.Y.; supervision, J.Y. and H.N.; project administration, J.Y. and H.N.; funding acquisition, J.Y. and H.N. All authors have read and agreed to the published version of the manuscript.

Funding: This work was funded by (1) the National Key R&D Program of China: Polymorphic Intelligent Network Environment (PINE) for Testing and Demonstrations (Project No.2020YFB1806402); (2) the Strategic Leadership Project of the Chinese Academy of Sciences: SEANET Technology Standardization Research System Development (Project No. XDC02070100).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data sharing is not applicable to this article.

Acknowledgments: We would like to express our gratitude to Jinlin Wang, Xiaoyong Zhu and Ri Han for their support of this work.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Paul, P.; Raghavan, S.V. Survey of multicast routing algorithms and protocols. In Proceedings of the Fifteenth International Conference on Computer Communication (ICCC 2002), Mumbai, India, 12–14 August 2002; pp. 902–926.
2. Roca, V.; Costa, L.; Vida, R.; Dracinschi, A.; Fdida, S. A Survey of Multicast Technologies; Work in Progress; Chicago, IL, USA, 2000. Available online: https://home.nr.no/~abie/Papers/sota_mcast.pdf (accessed on 1 March 2022).
3. AlSaeed, Z.; Ahmad, I.; Hussain, I. Multicasting in software defined networks: A comprehensive survey. *J. Netw. Comput. Appl.* **2018**, *104*, 61–77. [CrossRef]

4. Islam, S.; Muslim, N.; Atwood, J.W. A survey on multicasting in software-defined networking. *IEEE Commun. Surv. Tutor.* **2017**, *20*, 355–387. [CrossRef]
5. Oliveira, C.A.; Pardalos, P.M. A survey of combinatorial optimization problems in multicast routing. *Comput. Oper. Res.* **2005**, *32*, 1953–1981. [CrossRef]
6. Fan, C.; Bi, J.; Zhou, Y.; Zhang, C.; Yu, H. Ns4: A p4-driven network simulator. In Proceedings of the SIGCOMM Posters and Demos, Los Angeles, CA, USA, 22–24 August 2017; pp. 105–107.
7. Tortelli, M.; Rossi, D.; Boggia, G.; Grieco, L.A. Icn software tools: Survey and cross-comparison. *Simul. Model. Pract. Theory* **2016**, *63*, 23–46. [CrossRef]
8. Tortelli, M.; Rossi, D.; Boggia, G.; Grieco, L.A. Cross-comparison of icn software tools. In Proceedings of the ACM Conference on Information-Centric Networking (ICN'14), Paris, France, 24–26 September 2014.
9. Bosshart, P.; Daly, D.; Gibb, G.; Izzard, M.; McKeown, N.; Rexford, J.; Schlesinger, C.; Talayco, D.; Vahdat, A.; Varghese, G.; et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Comput. Commun. Rev.* **2014**, *44*, 87–95. [CrossRef]
10. Kanaya, T.; Nakao, A.; Yamamoto, S.; Yamauchi, H.; Nirasawa, S.; Oguchi, M.; Yamaguchi, S. Intelligent application switch supporting tcp. In Proceedings of the 2018 IEEE 7th International Conference on Cloud Networking (CloudNet), Tokyo, Japan, 22–24 October 2018; pp. 1–4.
11. Li, S.; Hu, D.; Fang, W.; Ma, S.; Chen, C.; Huang, H.; Zhu, Z. Protocol oblivious forwarding (pof): Software-defined networking with enhanced programmability. *IEEE Netw.* **2017**, *31*, 58–66. [CrossRef]
12. Nirasawa, S.; Nakao, A.; Yamamoto, S.; Hara, M.; Oguchi, M.; Yamaguchi, S. Application switch using dpn for improving tcp based data center applications. In Proceedings of the 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), Lisbon, Portugal, 8–12 May 2017; pp. 995–998.
13. Kreutz, D.; Ramos, F.M.; Verissimo, P.E.; Rothenberg, C.E.; Azodolmolky, S.; Uhlig, S. Software-defined networking: A comprehensive survey. *Proc. IEEE* **2014**, *103*, 14–76. [CrossRef]
14. Chen, S.; Gunluk, O.; Yener, B. The multicast packing problem. *IEEE/ACM Trans. Netw.* **2000**, *8*, 311–318. [CrossRef]
15. Wang, J.; Cheng, G.; You, J.; Sun, P. Seanet: Architecture and technologies of an on-site, elastic, autonomous network. *J. Netw. New Media* **2020**, *6*, 1–8.
16. KhadirKumar, N.; Bharathi, A. Real time energy efficient data aggregation and scheduling scheme for wsn using atl. *Comput. Commun.* **2020**, *151*, 202–207.
17. Lee, D.-L.; Youn, C.-H.; Jeong, S.-J. RP reselection scheme for real-time applications in delay-constrained multicast networks. In Proceedings of the 2002 IEEE International Conference on Communications. Conference Proceedings. ICC 2002 (Cat. No. 02CH37333), New York, NY, USA, 28 April–2 May 2002; Volume 2, pp. 1290–1294.
18. Wang, H.; Meng, X.; Zhang, M.; Li, Y. Tabu search algorithm for rp selection in PIM-SM multicast routing. *Comput. Commun.* **2010**, *33*, 35–42. [CrossRef]
19. Che, F.; Lloyd, E.L. The complexity of rp selection in multicast channelization. In Proceedings of the MILCOM 2009—2009 IEEE Military Communications Conference, Boston, MA, USA, 18–21 October 2009; pp. 1–7.
20. Biersack, E.; Nonnenmacher, J. Wave: A new multicast routing algorithm for static and dynamic multicast groups. In *International Workshop on Network and Operating Systems Support for Digital Audio and Video*; Springer: Berlin/Heidelberg, Germany, 1995; pp. 228–239.
21. Wang, C.-F.; Liang, C.-T.; Jan, R.-H. Heuristic algorithms for packing of multiple-group multicasting. *Comput. Oper. Res.* **2002**, *29*, 905–924. [CrossRef]
22. Guan, J.; Gao, S.; Li, X.; Zhou, H.; Zhang, H. The analysis and emulation of multicast join delay. In Proceedings of the 2009 IEEE International Conference on Network Infrastructure and Digital Content, Beijing, China, 6–8 November 2009; pp. 217–221.
23. Randaccio, L.S.; Atzori, L. Group multicast routing problem: A genetic algorithms based approach. *Comput. Netw.* **2007**, *51*, 3989–4004. [CrossRef]
24. Samadian-Barzoki, S.; Bag-Mohammadi, M.; Yazdani, N. BMP: An efficient and scalable multicast protocol. In Proceedings of the Canadian Conference on Electrical and Computer Engineering 2004 (IEEE Cat. No. 04CH37513), Niagara Falls, ON, Canada, 2–5 May 2004; Volume 4, pp. 1993–1996.
25. Henderson, T.R.; Lacage, M.; Riley, G.F.; Dowell, C.; Kopena, J. Network emulations with the ns-3 simulator. *SIGCOMM Demonstr.* **2008**, *14*, 527.
26. Lantz, B.; O'Connor, B. A mininet-based virtual testbed for distributed sdn development. *ACM Sigcomm Comput. Commun. Rev.* **2015**, *45*, 365–366. [CrossRef]
27. Oliveira, R.L.S.D.; Schweitzer, C.M.; Shinoda, A.A.; Prete, L.R. Using mininet for emulation and prototyping software-defined networks. In Proceedings of the 2014 IEEE Colombian Conference on Communications and Computing (COLCOM), Bogota, Colombia, 4–6 June 2014; pp. 1–6.
28. Wang, S.-Y.; Chou, C.-L.; Yang, C.-M. Estinet OpenFlow network simulator and emulator. *IEEE Commun. Mag.* **2013**, *51*, 110–117. [CrossRef]
29. Wang, S.-Y. Comparison of sdn OpenFlow network simulator and emulators: Estinet vs. mininet. In Proceedings of the 2014 IEEE Symposium on Computers and Communications (ISCC), Funchal, Portugal, 23–26 June 2014; pp. 1–6.
30. Varga, A. Omnet++. In *Modeling and Tools for Network Emulation*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 35–59.

31. Varga, A.; Hornig, R. An overview of the omnet++ simulation environment. In Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops, Marseille, France, 3–7 March 2008; pp. 1–10.
32. Li, B.; Wang, J. An identifier and locator decoupled multicast approach (ildm) based on icn. *Appl. Sci.* **2021**, *11*, 578. [CrossRef]
33. Oliveira, C.A.; Pardalos, P.M.; Resende, M.G. Optimization problems in multicast tree construction. In *Handbook of Optimization in Telecommunications*; Springer: Berlin/Heidelberg, Germany, 2006; pp. 701–731.
34. Bag-Mohammadi, M.; Yazdani, N. Next branch multicast (nbm) routing protocol. *Comput. Netw.* **2005**, *49*, 878–897. [CrossRef]
35. Kim, D.; Song, S.; Choi, B.-Y. Beam: Branch-based efficient and adaptive multicast for wireless sensor networks. *Int. J. Sens. Netw.* **2017**, *25*, 13–30. [CrossRef]
36. Stoica, I.; Ng, T.E.; Zhang, H. Reunite: A recursive unicast approach to multicast. In Proceedings of the IEEE INFOCOM 2000 Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No. 00CH37064), Tel Aviv, Israel, 26–30 March 2000; pp. 1644–1653.