# Formatting Submissions for a USENIX Conference: An (Incomplete) Example

Your N. Here
*Your Institution*

Second Name
*Second Institution*

## Abstract

The rapid evolution of network applications and environments necessitates advanced, customized routing protocols. However, the lack of comprehensive development and testing platforms often impedes progress in this field. We introduce ROMAM, an intra-autonomous system (AS) routing framework designed to accelerate the research and development of routing protocols. ROMAM's key innovation lies in its modular, highly adaptable framework that integrates both static and dynamic network information, enabling swift prototyping and assessment of advanced routing solutions. We demonstrate ROMAM's capabilities through five progressive use cases, spanning from traditional protocols to traffic and QoS-aware approaches. These cases showcase ROMAM's versatility in supporting the evolution of routing strategies. In addition, we present a comprehensive monitoring toolchain for pre-deployment evaluation of routing solutions. Our empirical evaluations reveal that ROMAM can reduce development efforts by up to 90% while providing robust support for advanced routing algorithms. By bridging the gap between theoretical concepts and practical implementation, ROMAM can be a versatile tool to speed up network routing research, paving the way for next-generation adaptive routing protocols.

## 1 Introduction

Traditional Internet routing, designed primarily for simplicity and scalability, aimed to deliver packets in a best-effort manner [12]. Today's network demands, driven by emerging applications such as AR/VR/XR and real-time control, require different dimensions such as ultra-high reliability, ultra-low latency, and minimal jitter [9]. These requirements exceed the capabilities of conventional routing protocols. Moreover, modern networks are increasingly heterogeneous and dynamic, encompassing land, air, sea, and space domains, utilizing diverse networking devices from ground to satellites [22]. This complexity introduces new challenges, making traditional "one-size-fits-all" routing protocols inadequate.

In response, large network operators have begun developing customized routing protocols tailored to specific environments, such as Software Defined Networks (SDNs) [15, 16, 19, 29] for data centers and Content Delivery Networks (CDNs) [8, 30] for efficient content distribution. While these specialized protocols address niche needs effectively, they lack the flexibility to be applied across varied network types and scenarios.

Furthermore, routing protocol development is hampered by the inherent complexity, high cost, and limitations of existing development frameworks. While SDN frameworks like OpenFlow [23] and P4 [6] offer powerful capabilities for programming the data plane and centralizing network control, they were not specifically designed to support the development of distributed routing protocols. These frameworks focus on network-wide programmability rather than providing a structured approach for designing and implementing new distributed routing algorithms. Similarly, platforms such as FRRouting [10], Bird [20], and OpenWRT [26], designed with legacy systems in mind, do not provide the tools or architectural support necessary to develop intelligent, adaptive routing protocols that can respond autonomously to changing network conditions and user demands.

To address these challenges and support the rapid evolution of network technologies, we present **ROMAM**[1] (ROuting MAnage Modules), a novel intra-autonomous system (AS) routing framework. By modularizing the routing protocol development process, ROMAM not only simplifies the creation of advanced routing algorithms but also enhances the adaptability of network operations to dynamic conditions, significantly improving packet-level Quality of Service (QoS). The architecture supports a flexible assembly of its components, allowing for customized solutions that address the specific requirements of modern network environments.

We demonstrate the effectiveness of ROMAM through the implementation of five routing protocols in ns-3, showcas-

[1] ROMAM is the accusative of direction in Latin meaning "to Rome", symbolizing our framework's goal of exploring multiple routing possibilities, much like the diverse roads that once led to Rome.

ing substantial improvements in service quality and network efficiency. Our contributions include:

- We present ROMAM, a novel modular framework for developing routing protocols. This framework decomposes routing functions into reusable components, allowing rapid prototyping and development of different routing strategies. By leveraging this modular design, ROMAM reduces coding effort by up to 90% compared to traditional approaches, significantly streamlining the protocol development process.

- We develop an extensive library of routing algorithms, queue disciplines, and traffic detection methods within the ROMAM framework. This comprehensive library serves as a foundation for researchers and developers to quickly experiment with and innovate new routing protocols, thereby accelerating the network routing research and development process.

- ROMAM integrates static network topology information with real-time traffic data, enabling the development of adaptive routing protocols. This integration allows protocols to effectively respond to changing network conditions, which is critical for improving packet-level Quality of Service (QoS) in dynamic network environments.

- We demonstrate the capabilities of ROMAM by implementing five different routing protocols, including two traditional and three innovative designs. These implementations showcase ROMAM's flexibility in supporting a wide range of routing strategies, from basic approaches to highly sophisticated, adaptive protocols, illustrating its versatility in addressing various network routing challenges.

The rest of the paper is organized as follows: Sec. 2 outlines the design principles of ROMAM. Sec. 3 details the architecture and implementation methodology. Sec. 4 presents five use cases demonstrating ROMAM's capabilities. Sec. 5 provides a comprehensive evaluation of ROMAM's performance. Sec. 6 discusses related work, and Sec. 7 concludes the paper with insights into future directions.

ROMAM is open source, and all codes are anonymously available at https://anonymous.4open.science/r/romam-7BC0/.

## 2 ROMAM Design Principles

The ROMAM framework is built upon two principles: the strategic utilization of both static and dynamic network information, and a modular system architecture. These principles are designed to address the complex challenges of modern network routing while providing a flexible and efficient development framework.

### 2.1 Leveraging Static and Dynamic Information

The primary objective of intelligent routing is to determine optimal paths for data packet delivery, aligning with specific QoS requirements at minimal cost. Achieving this goal within dynamic network environments involves navigating two critical trade-offs:

- **Scope of Information Collection vs. Associated Cost:** Expanding the scope of information collection improves routing decision accuracy but increases computational and bandwidth overhead.

- **Timeliness of Information vs. Inherent Collection Delays:** Timely information enables responsive routing, but data collection introduces delays, potentially leading to inaccurate or outdated information for decision-making.

To address these challenges, ROMAM employs a dual-information approach:

#### 2.1.1 Static Information

This includes network topology, link capacities, and other infrequently changing attributes. Static information forms the basis for forwarding direction exploration, providing a stable framework for initial route computation and regular operations.

#### 2.1.2 Dynamic Information

This encompasses rapidly changing network conditions such as traffic volumes, link statuses, and queue lengths. Dynamic information is vital for making real-time adjustments to routing decisions.

By leveraging static and dynamic information, ROMAM give routing developers the flexibility to make a trade-off between information scope and cost, as well as timeliness and accuracy. This tradeoff provides adaptability of routing protocols, allowing them to meet diverse network demands efficiently. For instance, ROMAM might use static topology information for baseline path computation, while dynamically adjusting routes based on real-time congestion data. This methodology facilitates both proactive resource management and dynamic adaptation to current network conditions, thereby improving overall network efficiency and responsiveness.

### 2.2 Modularity Principle

Inspired by the open-closed principle in software engineering, ROMAM redefines routing protocol architecture by decomposing it into smaller, reconfigurable components. This modular approach significantly enhances development efficiency, enabling rapid and flexible protocol development.

Key aspects of ROMAM's modularity include:

- **Component Isolation:** Each module (e.g., route discovery, traffic detection) functions independently yet interoperates seamlessly with others.
- **Standardized Interfaces:** Well-defined interfaces between modules facilitate easy integration and replacement of components.
- **Reusability:** Common functionalities are encapsulated in reusable modules, reducing redundant development efforts.

## 3 Design and Implementation Methodology

ROMAM architecture decouples the routing process into two main stages: route discovery and route selection. In the discovery stage, routing protocols utilize static network data to compile comprehensive lists of potential routes. The route selection stage is dynamic, with decisions made based on real-time traffic conditions and specific QoS requirements to optimize network performance.

To facilitate these stages, we define four core modules for ROMAM: the Information Collection Module (ICM), the Route Discovery Module (RDM), the Traffic Detection Module (TDM), and the Intelligent Forwarding Module (IFM). Fig. 1 illustrates the overall architecture and the interactions between these modules.

### 3.1 Information Collection Module (ICM)

The Information Collection Module (ICM) is a cornerstone component of ROMAM, crucial for optimizing control plane operations and facilitating efficient, scalable data exchange across the network infrastructure. The ICM's architecture is designed to decouple data management from computational processes, significantly enhancing system flexibility and operational efficiency.

Key functionalities of the ICM include:

- Advanced packet format management and parsing, supporting multiple protocol standards.
- High-performance, distributed database operations for control plane data.
- Dynamic configuration capabilities through a sophisticated Finite State Machine (FSM) implementation.
- Real-time network state collection and dissemination.

The ICM's modular design represents a paradigm shift from traditional monolithic architectures, offering unprecedented customization to address diverse network requirements. This approach enables fine-grained control over data collection granularity, frequency, and scope, allowing for optimized resource utilization and enhanced network visibility.

Table 1 delineates the core primitives provided by the ICM. These primitives form a comprehensive toolkit for network information exchange and storage, enabling efficient implementation of complex routing protocols and adaptive network management strategies.

### 3.2 Route Discovery Module (RDM)

The RDM tasked with identifying potential routes to destinations utilizing static network map data. This module houses a comprehensive library of routing algorithms and maintains a Routing Information Base (RIB) that stores route information. This information can be customized to align with the diverse objectives of different routing protocols, enhancing the adaptability and effectiveness of the routing process.

A key design feature of the RDM is its ability to leverage the extensive path diversity inherent in mesh networks. This capability is crucial for adapting routing strategies based on changing network conditions and for maximizing network throughput and reliability. In mesh networks, the number of potential paths increases exponentially with the size of the network. For example, in a $10 \times 10$ grid network, the number of 18-hop paths between two diagonal nodes can reach 48,620. Managing this complexity efficiently without overwhelming the RIB is a primary challenge that the RDM addresses.

ROMAM utilizes a distributed routing approach, allowing each network node to maintain multiple route options, each directed through one of its neighbors. This collaborative strategy enables the exploration of all possible paths, expanding the RIB linearly with the number of neighbors, which is manageable even as the network scales.

Fig. 2 illustrates how the ROMAM RIB is structured based on the network topology shown in Fig. 2a. Unlike traditional Shortest Path Tree (SPT) methods that identify a single shortest path from one node to all others, the Shortest Path Forest (SPF) method generates multiple paths through each neighbor of a node, significantly enriching the RIB (Fig. 2d). This approach not only supports the standard SPT algorithm but also accommodates advanced multi-path routing algorithms, enhancing the routing system's ability to handle network dynamics and congestion.

Moreover, the RDM includes next-hop interface information in the RIB, which is critical for making informed routing decisions, especially in scenarios where certain interfaces may experience congestion. This feature significantly aids in proactive routing adjustments, ensuring more reliable and efficient packet delivery.

The RDM supports a variety of popular routing algorithms, including Dijkstra's algorithm for unicast forwarding and ECMP routing, widely used in OSPF and IS-IS protocols. It also incorporates advanced algorithms like the Shortest Path Forest (SPF) and Recursive Shortest Path Forest (RecurSPF),

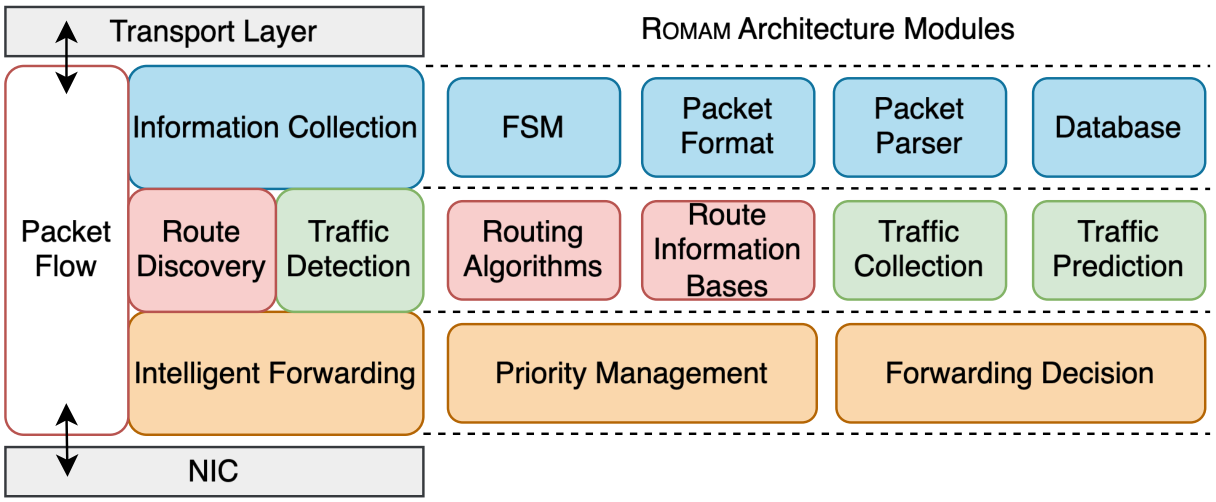


Figure 1: ROMAM-based routing operates between the transport layer and the network interface card (NIC). The architecture decouples information collection from algorithmic processes and combines static and dynamic data inputs. This flexible and scalable framework supports customizable routing protocols and the integration of advanced routing strategies, enabling dynamic responses to network conditions.

facilitating the exploration of alternate routes and enhancing the network's resilience to congestion and failures. This modular approach not only streamlines the development of new routing protocols but also fosters innovation by allowing researchers to easily integrate or modify existing algorithms to meet specific network requirements.

## 3.3 Traffic Detection Module (TDM)

The TDM enhances router capabilities by enabling the processing and prediction of dynamic traffic conditions, which are inherently more variable than static map data.

This function actively gathers dynamic traffic data such as link status, queue lengths, and traffic volumes. In ROMAM, these tasks are managed flexibly through a Finite State Machine (FSM), utilizing a suite of primitives from the Information Collection Module (ICM). Developers can specify the data type, its lifespan, and exchange scope based on routing needs, ensuring that collected data is pertinent and timely. Once gathered, this data is formatted and stored via database operations to support immediate and future routing decisions.

Additionally, the traffic prediction aspect of the TDM extends the module's utility by incorporating statistical and machine learning methods to forecast traffic conditions. Basic statistical tools provide initial insights by analyzing historical data to predict future traffic patterns. Moreover, a Markov chain model forecasts near-term traffic conditions from these trends. For more complex predictions, the module integrates advanced forecasting techniques such as time series analysis and Long Short-Term Memory (LSTM) networks, enhancing the router's ability to anticipate and adapt to changing network dynamics.

This dual approach not only supports real-time adaptive routing decisions but also bolsters the network's overall responsiveness and efficiency by leveraging both current and predictive traffic data insights.

## 3.4 Intelligent Forwarding Module (IFM)

The IFM serves as the decision-making core of the routing process, dynamically synthesizing information from the RDM and the TDM to enhance routing decisions. Unlike traditional routing protocols, which predominantly rely on static routing table lookups, the IFM enables adaptive route selection based on real-time traffic data and network conditions, aligning with specific network performance objectives.

The IFM leverages the modular architecture of ROMAM to facilitate the development of sophisticated routing solutions. For example, Equal-Cost Multi-Path (ECMP) routing can randomly select among multiple routes of equivalent cost. However, the definition of 'cost' can be strategically adapted to consider various factors such as latency, hop count, or bandwidth, depending on the routing objectives.

A crucial aspect of intelligent forwarding is the avoidance of routing loops, especially when multiple paths are considered at each hop. Traditional techniques like Reverse Path Forwarding (RPF), commonly used to prevent multicast loops, do not suffice due to the asymmetric nature of multi-path routing. ROMAM employs a distance-based loop avoidance strategy, ensuring that packets progressively move closer to their destination at each hop, effectively preventing loops. The definition of 'distance' can vary, including metrics such as hop count, delay, or even geographical distance, thus allowing for flexibility in loop avoidance strategies.

Table 1: Primitives in the ICM for network information exchange and storage.

| **Primitive** | **Input Arguments** | **Description** |
|---|---|---|
| PacketParse | Packet | Parses incoming control packets to extract vital information, ensuring effective data handling and routing decisions. |
| UnicastInfo | IPv4 Address | Manages the unicast of packets to specified IPv4 addresses, facilitating directed communication within the network. |
| BroadcastInfo | TTL | Executes packet broadcasting across the network, with propagation controlled by the Time-To-Live (TTL) parameter to prevent looping. |
| FormatData | Packet | Ensures data packets conform to the database schema for efficient storage and retrieval, enhancing data integrity and accessibility. |
| AccessDatabase | DatabaseID | Provides interfaces for robust database interactions, enabling efficient data queries and updates essential for dynamic routing decisions. |

In addition, ROMAM supports the incorporation of advanced forwarding algorithms, such as probabilistic route selection, Multi-Armed Bandit (MAB) or other reinforcement learning based approaches [1, 3, 7, 14]. Such methods enable a dynamic balance between exploring various paths and exploiting the most rewarding ones, based on observed network performance. This adaptive mechanism facilitates continuous refinement of routing decisions in response to changing network conditions, allowing ROMAM-based protocols to evolve and optimize their behavior over time.

Moreover, advancements in hardware capabilities and artificial intelligence have significantly expanded the computational capacity of network devices, enabling more complex decision-making processes. The IFM of ROMAM standardizes output interfaces for forwarding decisions, ensuring compatibility with these advanced technologies. This compatibility facilitates the integration of innovative, learning-based routing strategies, further enhancing the adaptability and efficiency of network operations.

## 4 ROMAM Use Cases

This section outlines five use cases demonstrating the versatility of the ROMAM framework in developing, adapting, and enhancing routing protocols. These cases progressively introduce increasingly sophisticated routing strategies that exemplify the framework's ability to support and enhance Quality of Service (QoS) and adapt to dynamic network conditions.

### 4.1 Overview of Implemented Protocols

#### 4.1.1 OSPF: Open Shortest Path First Protocol

OSPF [25], is a foundational protocol that calculates efficient routing paths using link-state information. Within ROMAM, OSPF's core functionalities of link-state data collection and path computation are seamlessly integrated, showcasing efficient protocol replication.

However, OSPF is essentially a static protocol that does not dynamically adjust to fluctuating network conditions. Its support for Equal-Cost Multi-Path (ECMP) routing, while useful, often falls short in dynamically congested networks due to the rarity of truly equal-cost paths.

#### 4.1.2 K-Shortest Path Routing Protocols

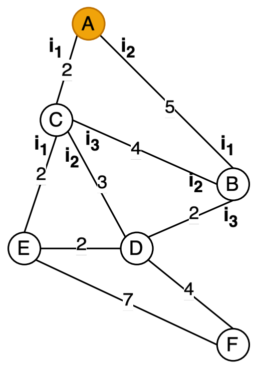
Building on OSPF's capabilities, K-Shortest Path protocols [21], explore multiple shortest paths to enhance routing choices, particularly under congested network conditions. These protocols leverage ROMAM's capabilities for path computation and dynamic path selection based on current traffic states. This integration allows for an adaptive routing response to real-time network conditions, showcasing an improvement in handling network dynamics compared to OSPF.

#### 4.1.3 Octopus: A MAB-based Intelligent Forwarding

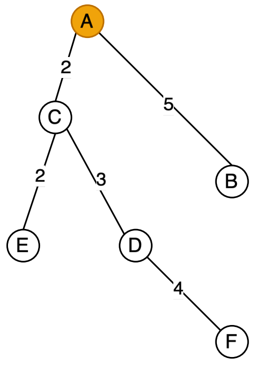
Octopus advances the concept of adaptive routing by incorporating Multi-Armed Bandit (MAB) algorithms, specifically the lightweight Exp3 algorithm [2], for dynamic path selection. This protocol, developed under the ROMAM framework, utilizes route information combined with real-time traffic data. Octopus's design, detailed in the Appendix 7, illustrates its ability to continuously refine its routing decisions based on ongoing network feedback, enhancing the adaptability of routing protocols to traffic dynamics.

#### 4.1.4 DGR: Delay Guaranteed Routing Protocol

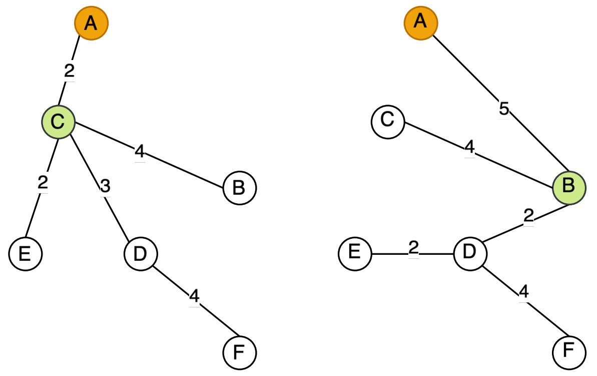
DGR [32] protocol introduces delay sensitivity into routing decisions, prioritizing routes that meet specific packet delivery deadlines. It utilizes both static and dynamic network data to dynamically adjust routes in response to detected traffic



(a) Topology



(b) Shortest Path Tree (SPT)



(c) Shortest Path Forest (SPF)

| Destination | Interface | Cost | Next Hop Interface |
|---|---|---|---|
| B | $i_2$ | 5 | – |
| C | $i_1$ | 2 | – |
| D | $i_1$ | 5 | $i_2$ |
| E | $i_1$ | 4 | $i_1$ |
| F | $i_1$ | 9 | $i_2$ |
| B | $i_1$ | 6 | $i_3$ |
| C | $i_2$ | 9 | $i_2$ |
| D | $i_2$ | 7 | $i_3$ |
| E | $i_2$ | 9 | $i_3$ |
| F | $i_2$ | 11 | $i_3$ |

(d) Routing Information Base (RIB)

Figure 2: Illustration of the ROMAM architecture's routing information base process. Fig. 2a shows an undirected graph representing a mesh network topology with node A seeking routes to other nodes. Fig. 2b displays the SPT generated by Dijkstra's algorithm from node A. Fig. 2c highlights the Shortest Path Forest (SPF) algorithm results, creating multiple SPTs, each rooted at a different neighbor of node A to explore diverse routing paths. Fig. 2d details the RIB used by both SPT and SPF algorithms, with dotted enclosures indicating new fields for SPF support, enhancing the RIB's adaptability to dynamic network conditions.

conditions. The integration of a priority management function ensures that packets with critical deadlines are expedited, enhancing the QoS support over previous protocols.

#### 4.1.5 DDR: Deadline-Driven Routing Protocol

DDR [33] protocol addresses inaccuracies in traffic information due to delays in network data transmission, which can crucially impact routing decisions. By implementing a predictive traffic state function, DDR uses historical and current data to make more accurate routing decisions. This enhancement helps compensate for the latency issues inherent in dynamic network environments, significantly improving the reliability and performance of routing decisions under real-world conditions.

### 4.2 Coding Efficiency

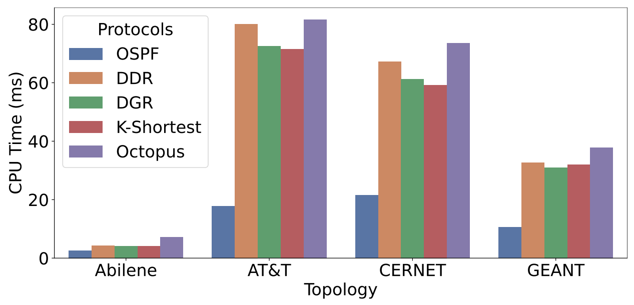
The ROMAM framework allows developers to focus solely on the unique aspects of each protocol, eliminating the need to rewrite redundant code. This approach leads to remarkable efficiency in protocol development:

- K-Shortest Path implementation primarily focuses on the routing algorithm and RIB format.
- Octopus concentrates on the MAB-based intelligent route selection function design.
- DGR's implementation centers on traffic information exchanges among neighbors and handling packet deadlines during route selection.
- DDR introduces a Markov chain-based prediction model for traffic and related path selection functions.

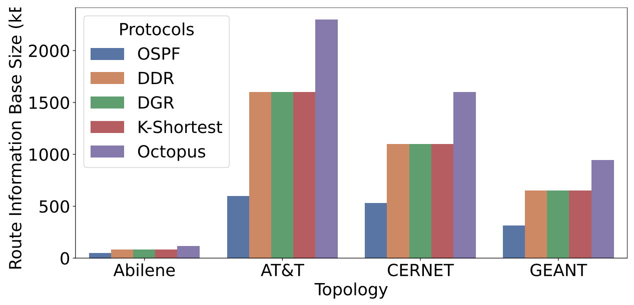
Each protocol implementation required less than 500 lines of code changes to the existing ROMAM framework, whereas implementing these protocols from scratch or modifying non-ROMAM implementations typically requires over 10,000 lines of code per protocol [10, 13]. This comparison demonstrates that ROMAM achieves more than a 90% reduction in coding efforts.

## 5 Evaluation

We implemented the aforementioned five routing protocols using the ROMAM framework and deployed them on four well-known network topologies, Abilene, AT&T, CERNET, and GEANT, sourced from the Internet Topology Zoo dataset [18]. The implementations and tests were run on ns-3 [27] to facilitate evaluation on various networks. The evaluations were performed on a consumer-grade computer with an Ubuntu 22.04 operating system, an Intel i7-12700 CPU, and 32 GB of RAM. Using ROMAM's monitoring tools we developed, we can observe the protocol cost metrics such as CPU usage and



(a) CPU time for protocol initialization.



(b) Memory cost for RIB.

Figure 3: Resource Consumption Monitoring. The CPU and memory usage during the initialization of routing protocols across network topologies of Abilene, ATT, CERNET, and GEANT. Fig. 3a displays the CPU time consumed from the installation of the routing protocol to the completion of the routing tables, indicating when routers are operational. Fig. 3b illustrates the memory utilization for maintaining the RIB and queue states.

memory usage, and the network performance metrics such as packet delay.

## 5.1 CPU and Memory Usage

CPU and memory consumption are crucial metrics for assessing the deployment costs of routing protocols. Fig. 3 illustrates the CPU time and memory consumption needed to initialize different routing protocols across four network topologies. Among the five algorithms evaluated, OSPF exhibits the lowest CPU and memory usage, attributed to its straightforward initialization process. Conversely, DDR, DGR, K-Shortest, and Octopus demonstrate higher CPU and memory consumption, with Octopus being slightly more resource-intensive than the others. This is primarily due to additional initialization requirements, such as calculating cumulative path losses, which necessitate greater computational resources.

Compared to OSPF, advanced routing protocols consume approximately 2× to 3× more CPU and memory resources. This increased resource usage is necessary for discovering multiple routes and maintaining a larger RIB. Despite the heightened complexity, these protocols do not introduce significant computational overhead when compared to large-scale learning-based information processing. Moreover, they do not necessitate additional hardware such as GPUs, rendering them highly compatible with existing network equipment.

## 5.2 Convergence of Octopus

For Octopus, its convergence performance is a key issue. In the conducted experiments, each topology entailed the selection of random source-destination pairs, and the protocol's performance was assessed by continuously measuring the end-to-end delay of each delivered packet. During the experiment, each router provided queuing delay reports to one-hop

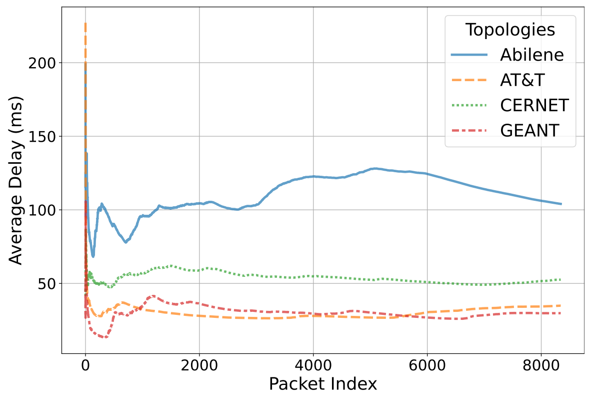
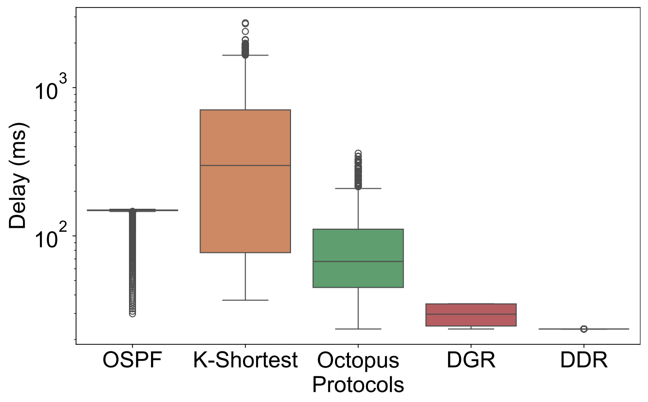
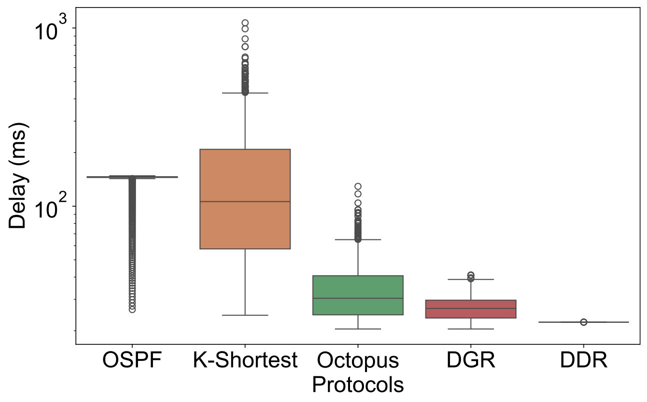


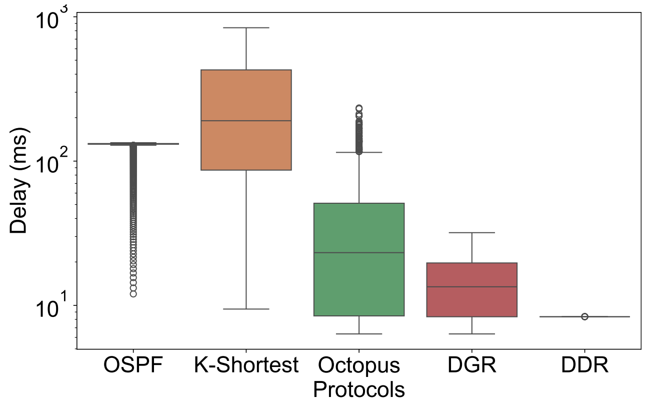
Figure 4: Convergence of the average packet delay of the Octopus protocol. This figure illustrates the dynamic adaptation of the Octopus protocol by showing the evolution of the average packet delay. Over time, the protocol's routing decisions increasingly favor paths with reduced delays, demonstrating its ability to learn and iteratively optimize routing efficiency.



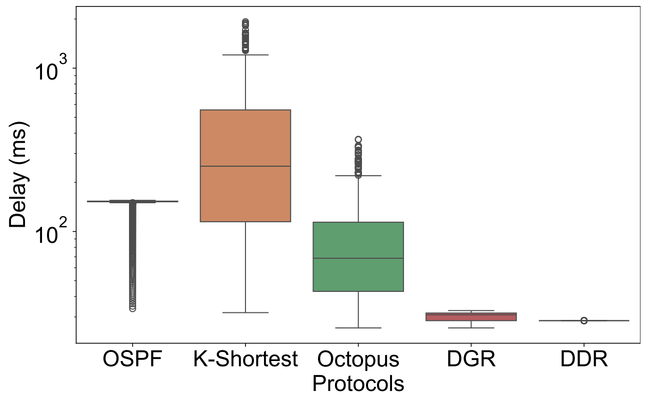
(a) Packet delay distribution at Abilene



(b) Packet delay distribution at AT&T



(c) Packet delay distribution at CERNET



(d) Packet delay distribution at GEANT

Figure 5: Packet delay distribution under single-point congestion across different network topologies. This boxplot illustrates the impact of network congestion on packet delay across five routing protocols. Each protocol's response to congestion is depicted through median, quartiles, and outliers, highlighting their performance in terms of resilience and load balancing. OSPF experiences significant delay spikes under congestion, relying on a single path. In contrast, K-Shortest Path, DDR, DGR, and Octopus distribute traffic across multiple paths, enhancing network resilience. Particularly, Octopus, after training, shows superior performance by selecting paths with lower latencies through its MAB algorithm. DDR and DGR demonstrate the best performance by preemptively filtering out non-viable paths and exploring alternative optimal routes before congestion escalates, thus maintaining reliable and timely data transmission.

neighbors. This information was crucial for updating the cumulative loss associated with each routing choice for Octopus.

Fig. 4 shows how network performance evolves over time when Octopus is deployed. Throughout the experiment, Octopus demonstrates its adaptability and learning efficiency, as evidenced by the gradually decreasing trend in average packet delay. In other words, routers increasingly favor more efficient paths based on accumulated experiences.

**Observations:** The gap introduced by the inherent uncertainties in real networks poses significant challenges in bridging theoretical models with their practical engineering deployment. This discrepancy is particularly relevant when training the Octopus protocol, where the feedback necessary for the learning algorithm ($d_{i'}$) is inherently delayed.

To effectively implement the Octopus protocol in such conditions, it is essential to employ a strategy known as learning with delayed feedback model, as detailed in [14]. In this model, updates to the learning algorithm do not necessarily occur within the same iteration as the decision-making process. For example, if a routing decision for a packet must be made before feedback from the previous iteration is available, the current model's state is used to make the decision.

However, as [14] highlights, the implications of delayed feedback vary based on the nature of the environment. In stochastic settings, separating decision-making and model updating incurs only an additive penalty. Conversely, in adversarial environments, delayed feedback can significantly impair the convergence of MAB, affecting it in a multiplicative manner. This remains an important further research issue.

### 5.3 Packet Delay Comparison

In this set of experiments, we compare the five routing protocols by measuring the end-to-end delay of a source-destination pair for each network topology under single-point congestion. The single-point congestion was introduced by first determining the optimal routing path in an ideal network free of competing traffic, then randomly selecting a node along this path and imposing heavy background traffic to saturate the bandwidth of the node's outgoing link. This experiment aims to mimic a scenario where a bottleneck occurs along a routing path, testing whether the routing algorithms can intelligently reroute around the congested link.

We collected the end-to-end delay of packets received at the destination. The measured delay distribution allowed us to analyze the effectiveness of each routing protocol under practical conditions. It is important to note that the performance statistics for the Octopus protocol shown in Fig. 5 are based on results after the protocol underwent an extensive training period of $10,000$ packets.

Fig. 5 compares the delay distribution of the five routing protocols across four different network topologies with single-point congestion, presented as box plots. A box plot is a standardized way of displaying data distribution based on a five-number summary: minimum, first quartile (Q1), median, third quartile (Q3), and maximum, where the box is drawn from Q1 to Q3 with a horizontal line inside to denote the mean delay, and whiskers from the minimum to the maximum delay. Any data that are outside of the whiskers are outliers.

From Fig. 5, across the four topologies, DDR, and DGR exhibit the best delay performance compared to OSPF, Octopus and K-Shortest. Their performance difference is mainly due to the different forwarding strategies employed by each protocol. OSPF suffers large delays, as the single-path routing protocol cannot adapt to network congestion.

Although K-Shortest can select multiple paths for packet delivery, its decisions are made randomly and lack responsiveness to real-time network conditions. Consequently, while some packets may achieve low delay, the overall delay performance of K-Shortest tends to be suboptimal.

Octopus outperforms K-Shortest. This is because, after training, the Exp3 algorithm tends to choose paths with lower latency, and it can keep track the neighborhood queue length reports to adapt to network dynamics accordingly.

DDR and DGR further outperform Octopus. This is because, based on their design, aiming to ensure the end-to-end delay is below the packet delay requirement (which is set to 50 ms in our experiment), the forwarding function effectively filters out paths that do not meet the delay requirements, resulting in all packets with delay below the threshold. DDR is better than DGR, as DGR uses the received neighborhood queue reports directly, while DDR further uses a traffic prediction function to estimate the future queue lengths of each node's one-hop neighbors. Such prediction can effectively help the node choose the most suitable paths.

Due to space limitations, we only report the results of a small set of experiments in this paper, and more experiments have been conducted that show a similar trend. The codes are available at `https://anonymous.4open.science/r/romam-7BC0/`, so the experiments can be reproduced with any setting. Overall, when there is no congestion, single-path routing such as OSPF can achieve decent performance. When network congestion or link failure occurs, multi-path routing protocols are more advantageous, provided that the protocols (like DGR, DDR and Octopus) are aware of neighborhood situations and can adjust forwarding decisions accordingly. If the traffic hotspot changes over time, advanced traffic detection and prediction methods (e.g., the one used by DDR) are beneficial.

## 6 Related Work

### 6.1 Routing Protocol Suites

Conventional routing protocol development has largely relied on comprehensive software suites. FRRouting [10], an evolution of Quagga, provides a collection of IP routing protocols for Unix platforms, while XORP [11] offers a programmable routing platform supporting multiple protocols. These suites, while offering stability and wide protocol support, lack the flexibility and modularity required for rapid prototyping of novel routing strategies.

Recent efforts like ONOS [5] have created more programmable environments, but they focus primarily on SDN scenarios rather than distributed routing protocols. xBGP [31] enhances BGP with custom code execution, aligning with ROMAM's goals of modularity and extensibility. However, xBGP is limited to BGP-specific enhancements, whereas ROMAM provides a comprehensive framework applicable to a variety of intra-AS routing protocols.

### 6.2 SDN and Programmable Networking

The advent of Software-Defined Networking has introduced new paradigms in network programmability. OpenFlow [23] pioneered the separation of control and data planes, while P4 [6] enabled protocol-independent packet processors. These approaches primarily focus on centralized control and data plane programmability, which, while powerful, have limitations in distributed environments. Developments such as Intel Tofino chip [4], have pushed the boundaries of programmable networking hardware. While these efforts have advanced hardware and programming languages, they lack a routing framework for rapid protocol development. ROMAM fills this gap by combining distributed routing with modular design principles, enabling the creation of adaptive, intelligent routing protocols that can operate effectively in both centralized and distributed environments.

## 6.3 Machine Learning in Routing

Recent years have seen growing interest in applying machine learning to networking problems. Mestres et al. [24] introduced the concept of Knowledge-Defined Networking, proposing a new paradigm that incorporates machine learning in network operations. Klaine et al. [17] surveyed machine learning techniques in self-organizing networks, highlighting the potential of AI in this domain.

While these works demonstrate the promise of AI in networking, they often remain theoretical or limited in scope. For instance, RouteNet [28] shows promising results in predicting network performance but doesn't provide a framework for implementing ML-driven routing protocols. ROMAM bridges this gap by providing native support for integrating machine learning algorithms into practical routing decisions, enabling the development and deployment of intelligent, adaptive routing protocols.

## 7 Conclusion

ROMAM is a platform designed to facilitate the development of intelligent distributed routing protocols. By integrating modular components that foster the adoption of advanced routing algorithms and traffic management strategies, ROMAM offers substantial benefits to network researchers and developers, enabling them to efficiently develop and deploy sophisticated customized routing protocols. In addition, it provides a modularized library that simplifies the customization and extension of routing capabilities, making it easier to adapt to varying network requirements and conditions.

We have showcased the capability of ROMAM by implementing five routing protocols, underscoring its and effectiveness as a research tool. Looking forward, ROMAM is poised to play a crucial role in the intersection of artificial intelligence and network management, potentially driving the evolution of next-generation intelligent routing technologies.

ROMAM would be more useful if it can be test using real-world network traffic data. Currently, ROMAM relies on ns-3 for generating network traffic, using random generators for TCP/UDP traffic. This approach falls short of capture the complexity and variability of actual network traffic, creating a gap that may limit our ability to obtain realistic network state. Given the promising results, our future plans include implementing ROMAM in re-programmable routers.

In a nutshell, Octopus uses the SPF algorithm in the RDM to find possible routes. With SPF, each node will generate an RIB, where each destination is associated with multiple entries. The number of entries equals the number of neighbors with a smaller "distance" to the destination (to avoid loop).

Each entry in the RIB includes the interface of the current node to use, the estimated cost if using this interface, and the next-hop's interface along the path to the destination. For Octopus, the estimated cost in the RIB is determined as the minimum path delay (assuming zero queueing delay along the path) from the current node to the destination via the next-hop node associated with each entry. Subsequently, the traffic information within the one-hop neighborhood is utilized to calculate the reward of each routing decision, effectively limiting communication overheads.

Formally, consider a graph $\mathcal{G} = (V, E)$. For each node $v \in V$, denote by $N_v$ the set of its neighbor nodes. Denote by $D_v = V \setminus v$ the set of all possible destinations for packets to be forwarded by $v$.

After running SPF, for each destination $d \in D_v$, $v$ finds a set of available paths, associated with its interface set $v_d$. Each path (interface) $i \in v_d$ is associated with the cost $c_i$ (i.e., the minimum path delay if choosing interface $i$ to $d$ without queueing delay) and the next-hop interface $i'$. Here, $c_i$ is considered static unless there is any topology changes.

The fast-changing part in the network is the queueing delays. The traffic information exchange between neighbor nodes can occur frequently, as the number of bits needed for traffic information between neighbors is quite negligible. Therefore, Octopus can use the queuing delay of both current and next-hop interfaces for path selection.

With a slight abuse of notation, we denote by $d_i$ the queueing delay for a packet going through interface $i$. Then, we define the loss of selecting $i \in v_d$ for forwarding one packet as

$$l_i := 1 - e^{-(c_i + d_i + d_{i'})},$$

where $d_{i'}$ is the queuing delay for interface $i'$ on the next-hop nodes. Note that $d_i$ can be observed by the current node, and $d_{i'}$ requires the feedback from the next-hop neighbor. $l_i$ increases when delay increases.

**Algorithm 1** Octopus Routing Selection on Node $v \in V$

```
Require: $v_d, \forall d \in D_v$ after running SPF
 1: for $d \in D_v$ do                ▷ Initialization for all destinations
 2:     $N_d = 0$
 3:     for $i \in v_d$ do
 4:         $L_i \leftarrow 1 - e^{-c_i}$
 5:     end for
 6: end for
 7: while Packet do
 8:     $d \leftarrow$ Packet.destination
 9:     $N_d \leftarrow N_d + 1$
10:     $\eta_d = \sqrt{\frac{|v_d| \ln |v_d|}{N_d}}$
11:     $p_i \leftarrow \frac{e^{-\eta_d L_i}}{\sum_{j \in v_d} e^{-\eta_d L_j}}, \forall i \in v_d$
12:     Forward Packet to interface $i$ according to $p_i$
13:     Observe the queuing delay $d_{i'}$ on the next-hop interface $i'$
14:     Update $L_i \leftarrow L_i + \frac{l_i}{p_i}$
15: end while
```

With the above notations, Octopus is given in Algorithm 1.

For each destination $d \in D_v$, a counter $N_d$ is used to count how many packets have been forwarded to $d$, and for each route entry $i$, we maintain a variable $L_i$, which keeps track of the cumulative loss for forwarding packets to $d$ via $i$. $N_d$ and $L_i$ are initialized in the for-loop of lines 1 to 5 in Algorithm 1.

Path selection in Octopus fundamentally operates with two processes: observing queue reports from neighboring routers and adjusting its forwarding decisions accordingly. Neighbors sent queue length reports through all active links.

The reception of a one-hop queuing delay report can be used to obtain $d_{i'}$ to update $l_i$ and $L_i$. $L_i$ and the learning rate $\eta_d$ (determined by $N_d$), in line 10, is used to calculate the probability of selecting $i$, $p_i$ in line 11 of Algorithm 1. Finally, the forwarding decision is sampled from the distribution $p_i$.

## References

[1] Maria Apostolaki, Ankit Singla, and Laurent Vanbever. Performance-driven internet path selection. In *Proceedings of the ACM SIGCOMM Symposium on SDN Research (SOSR)*, pages 41–53, 2021.

[2] Peter Auer, Nicolo Cesa-Bianchi, Yoav Freund, and Robert E Schapire. Non-stochastic bandit problems. In *Annual Symposium on Foundations of Computer Science*, pages 163–172. IEEE, 2002.

[3] Baruch Awerbuch and Yishay Mansour. Adapting to a reliable network path. In *Proc. ACM Symposium on Principles of Distributed Computing (PODC)*, pages 360–367, 2003.

[4] Barefoot Networks. Barefoot tofino. https://www.barefootnetworks.com/products/brief-tofino/, 2018. Accessed: 2023-05-30.

[5] Pankaj Berde, Matteo Gerola, Jonathan Hart, Yuta Higuchi, Masayoshi Kobayashi, Toshio Koide, Bob Lantz, Brian O'Connor, Pavlin Radoslavov, William Snow, et al. ONOS: towards an open, distributed SDN OS. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 1–6, 2014.

[6] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2014.

[7] Sébastien Bubeck, Nicolo Cesa-Bianchi, et al. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends® in Machine Learning*, 5(1):1–122, 2012.

[8] Jiayi Chen, Nihal Sharma, Tarannum Khan, Shu Liu, Brian Chang, Aditya Akella, Sanjay Shakkottai, and Ramesh K Sitaraman. Darwin: Flexible learning-based CDN caching. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pages 981–999, 2023.

[9] Shuping Dang, Osama Amin, Basem Shihada, and Mohamed-Slim Alouini. What should 6G be? *Nature Electronics*, 3(1):20–29, 2020.

[10] FRRouting project. https://frrouting.org/, 2017. Accessed: 2024-05-05.

[11] Mark Handley, Orion Hodson, and Eddie Kohler. XORP: An open platform for network research. *ACM SIGCOMM computer communication review*, 33(1):53–57, 2003.

[12] Christian Huitema. *Routing in the Internet*. Prentice-Hall, Inc., 1995.

[13] Paul Jakma and David Lamparter. Introduction to the Quagga Routing Suite. *IEEE Network*, 28(2):42–48, 2014.

[14] Pooria Joulani, Andras Gyorgy, and Csaba Szepesvári. Online learning under delayed feedback. In *International Conference on Machine Learning (ICML)*, pages 1453–1461. PMLR, 2013.

[15] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pages 1–12, 2017.

[16] Anuj Kalia, Michael Kaminsky, and David Andersen. Datacenter RPCs can be general and fast. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 1–16, 2019.

[17] Paulo Valente Klaine, Muhammad Ali Imran, Oluwakayode Onireti, and Richard Demo Souza. A survey of machine learning techniques applied to self-organizing cellular networks. *IEEE Communications Surveys & Tutorials*, 19(4):2392–2431, 2017.

[18] Simon Knight, Hung X Nguyen, Nickolas Falkner, Rhys Bowden, and Matthew Roughan. The Internet topology zoo. *IEEE Journal on Selected Areas in Communications*, 29(9):1765–1775, 2011.

[19] Thomas Koch, Shuyue Yu, Sharad Agarwal, Ethan Katz-Bassett, and Ryan Beckett. Painter: Ingress traffic engineering and routing for enterprise cloud networks. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pages 360–377, 2023.

[20] CZ.NIC Labs. Bird Internet Routing Daemon. https://bird.network.cz, 2024. Accessed: 2024-07-10.

[21] Huiping Liu, Cheqing Jin, Bin Yang, and Aoying Zhou. Finding top-k shortest paths with diversity. *IEEE Transactions on Knowledge and Data Engineering*, 30(3):488–502, 2017.

[22] Jiajia Liu, Yongpeng Shi, Zubair Md Fadlullah, and Nei Kato. Space-air-ground integrated network: A survey. *IEEE Communications Surveys & Tutorials*, 20(4):2714–2741, 2018.

[23] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM computer communication review*, 38(2):69–74, 2008.

[24] Albert Mestres, Alberto Rodriguez-Natal, Josep Carner, Pere Barlet-Ros, Eduard Alarcón, Marc Solé, Victor Muntés-Mulero, David Meyer, Sharon Barkai, Mike J Hibbett, et al. Knowledge-defined networking. *ACM SIGCOMM Computer Communication Review*, 47(3):2–10, 2017.

[25] J. Moy. OSPF Version 2. RFC 2328, IETF, 1998. Accessed: 2024-05-05.

[26] OpenWRT. https://openwrt.org/, 2024. Accessed: 2024-05-05.

[27] George F. Riley and Thomas R. Henderson. *The ns-3 Network Simulator*, pages 15–34. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[28] Krzysztof Rusek, José Suárez-Varela, Paul Almasan, Pere Barlet-Ros, and Albert Cabellos-Aparicio. RouteNet: leveraging graph neural networks for network modeling and optimization in SDN. *IEEE Journal on Selected Areas in Communications*, 38(10):2260–2270, 2020.

[29] Hua Shao, Xiaoliang Wang, Yuanwei Lu, Yanbo Yu, Shengli Zheng, and Youjian Zhao. Accessing Cloud with Disaggregated Software-Defined Router. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 1–14, 2021.

[30] Zhenyu Song, Daniel S Berger, Kai Li, Anees Shaikh, Wyatt Lloyd, Soudeh Ghorbani, Changhoon Kim, Aditya Akella, Arvind Krishnamurthy, Emmett Witchel, et al. Learning relaxed belady for content distribution network caching. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 529–544, 2020.

[31] Thomas Wirtgen, Tom Rousseaux, Quentin De Coninck, Nicolas Rybowski, Randy Bush, Laurent Vanbever, Axel Legay, and Olivier Bonaventure. xBGP: Faster Innovation in Routing Protocols. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 575–592, 2023.

[32] Pu Yang, Lin Cai, and Tianfang Chang. DGR: delay-guaranteed routing protocol. In *2023 IEEE 20th International Conference on Mobile Ad Hoc and Smart Systems (MASS)*, pages 19–27. IEEE, 2023.

[33] Pu Yang, Tianfang Chang, and Lin Cai. DDR: a deadline-driven routing protocol for delay guaranteed service. In *IEEE INFOCOM 2024-IEEE Conference on Computer Communications*, pages 1–10. IEEE, 2024.