Qiao: DIY your routing protocol in Internet-of-Things

Zhiyu Wu*
College of Mathematics and Computer Science
Zhejiang A&F University
Hangzhou, China
ludovico.zy.wu@gmail.com

Yisu Wang*
School of Communication Engineering
Xidian University
Xi'An, China
asueeer@163.com

***Abstract*—Routers in the Internet of Things (IoT) operate in a failure-prone, multi-device integrated, and dynamically changing network topology environment. Dealing with diverse network failures, device compatibility, and flexible configurations are the challenges faced by IoT routers [21]. Existing traditional Internet solutions and recently emerging Software-Defined Networking (SDN) solutions are not capable of addressing these issues effectively. They either lack support for flexible configurations or suffer from compatibility problems.**

This article proposes Qiao, a novel routing system written in a high-level language (HLL), with built-in concurrency and a robust standard library for industrial internet, that aims to tackle the interoperability issues in small to medium-sized industrial IoT networks. Qiao provides a universal, flexible, and efficient solution. First, Qiao is developed using the high-level programming language Go, which allows it to run on any Unix-compatible device, making it highly versatile. Second, Qiao leverages the advantages of multi-threading and concurrency to provide low-latency data forwarding capabilities. Third, Qiao adopts a layered and modular design, with all its source code being open-source, enabling anyone to customize their routing protocols according to their collaborative requirements in the industrial internet.

***Index Terms*—networks, routing, IoT, edge computing**

I. Introduction

The Industrial Internet of Things (IoT) facilitates seamless communication and interconnection among various industrial devices, such as sensors, control systems, smart mobile systems, and embedded industrial microsystems. This integration aims to enhance production efficiency and achieve industrial intelligence. Within this ecosystem, effective management, control, and optimization of networks are crucial, serving as the backbone for interconnected devices.

Industrial systems have specific network requirements that surpass the capabilities of traditional routing approaches. These requirements include low latency, high reliability, privacy and security, as well as flexible and configurable network topologies. With the development of SDN technology, many enterprises have embraced SDN [12] [24] strategies to centralize the management and flexible configuration of routing functions. By dynamically customizing routing rules and policies based on specific device requirements, they can address these challenges effectively. [11] Furthermore, the introduction of edge routers offers features such as low-power consumption, energy efficiency, and robustness. As a result, routing functions can be decentralized to the edge, enabling faster and more reliable local computation and data transmission for interconnected devices through edge computing.

However, despite the advantages of SDN, deploying fully centralized network management in IoT environments, which often experience frequent changes in network topology, presents certain limitations. These include potential single points of failure [17], increased complexity, and potential performance issues.

Taking these factors into account, we propose Qiao, a comprehensive solution for routing management and configuration that combines software routing with edge computing. Qiao leverages the capabilities of edge nodes by running general-purpose routing software on them. It introduces externally programmable interfaces through custom configurations in the Qiao protocol layer, enabling the implementation of customized routing rules in a modular manner across various edge nodes. This approach allows for the adoption of diverse global routing rules while maintaining a unified configuration for local routing rules. As a result, Qiao ensures resilience and flexibility in complex and ever-changing network topologies while addressing the limitations associated with fully centralized network management. We have made Qiao available at https://github.com/QiaoRouter/qiao.

Currently, Qiao supports the Routing Information Protocol Next Generation (RIPng) [2] and Dynamic Host Configuration Protocol (DHCP) [1]. Qiao possesses a robust and reusable hardware abstraction layer that facilitates the open-source community's contribution to developing additional routing protocols. Concurrently, Qiao provides a reliable and user-friendly API, enabling network administrators to easily specify the router's network functions. The main features are as follows.

First, Qiao is characterized by its lightweight, modular design philosophy. Implemented in Golang, a language known for its excellent package management and simplicity, QIAO divides the processing of network packets into multiple streamlined stages, which can be flexibly combined into a complete routing solution for specific protocols through pluggable interfaces. Users can modify, configure, and optimize rules, and

*These authors contributed equally to this work.

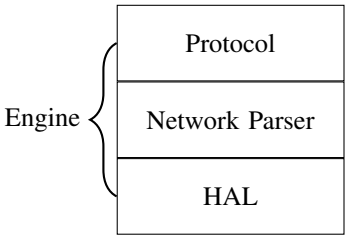

Fig. 1. Composition of the Qiao Engine architecture

design specific strategies tailored to different IoT products based on the foundation of general protocol rules.

Second, Qiao exhibits excellent portability, as it can easily run on any Unix-compatible system. With advancements in CPU architectures, including increased core counts, the emergence of new network devices like Data Processing Units (DPUs) [7], improved GPU parallel computing capabilities, and the availability of high-speed development kits such as the Data Plane Development Kit (DPDK) [15], the processing speed of general-purpose PCs for data packets has improved from milliseconds to microseconds.

Furthermore, Qiao provides a possible solution for building a modular IoT routing system. By offloading the functionalities of dedicated routers to general-purpose computers and enabling rapid routing upgrades and iterations through custom configurations, it allows IoT enterprises to design and customize routing solutions at minimal cost to match the diverse networking requirements of each connected device.

Qiao is designed to tackle the challenges encountered in complex IoT scenarios. As we move forward, our primary objective is to enhance Qiao's capabilities by supporting additional routing protocols and seamlessly integrating them with other solutions. We envision Qiao as a driving force behind advancements in software routing, unlocking new possibilities for collaborative computing.

II. DIY Scenarios

In the complex and diverse landscape of the Internet of Things (IoT), there exist numerous scenarios that necessitate the adoption of do-it-yourself (DIY) routing protocols. These scenarios often demand customized solutions for large-scale deployments, flexible configuration of parameters and routing behaviors, as well as considerations for security and privacy.

A. Flexible Configuration within a LAN

During the deployment process of a small to medium-sized industrial local area network (LAN), network administrators face the task of configuring pariticular switches and routers. This process demands specialized knowledge and consumes a significant amount of time. However, in the context of deploying IoT devices within a local network, such as in mining fields, coal plants, or factory production lines, the scale typically involves no more than ten thousand devices. To streamline this process, a viable approach involves predefining a shared random key for each device, enabling automatic IP address assignment and communication among devices sharing the same key through Layer 2 protocols. This mechanism facilitates the formation of an industrial LAN, which can be facilitated by Qiao's protocol layer. Moreover, devices equipped with multiple keys possess the capability to connect to multiple local networks, establishing distinct LANs tailored to specific operational needs. These configurations can be achieved through Qiao's automated provisioning capabilities.

B. Customizable Routing Strategies

In scenarios where network topology exhibits instability, IoT routers may necessitate employing dynamic routing strategies such as load balancing, flooding, path exploration, or multipath routing. For instance, routers can employ sampling techniques to identify hosts with a high packet retransmission rate, indicating potential delays in packet delivery along a specific path. In such cases, routers can intelligently attempt packet transmission through alternative paths. Statistical analysis of transmission success rates across different paths can guide the router in updating its routing table with the optimal path. While traditional general-purpose router products offer flexibility in customizing routing strategies, the match-action model of the P4 [8] language presents challenges in achieving such dynamic routing capabilities.

III. Architecture design

As illustrated in Figure 1, the architecture of Qiao's Engine is stratified into three primary layers, ascending from the bottom: the Hardware Abstraction Layer (HAL), the Network Parser, and the Protocol layer.

Indicated by its designation, the HAL functions to obscure low-level complexities, thus simplifying the design process for users. It encapsulates a series of design objectives such as decreasing the complexity of interaction with the underlying system, promoting code reusability, and concealing the backend mechanism for packet capture. By transforming the elements that interact with the lower layers into universally applicable interfaces, HAL significantly reduces the burden on developers to comprehend low-level APIs. This abstraction facilitates portability in Qiao, enabling a software router written on one platform to operate seamlessly on another, namely compile ones, run everywhere. Furthermore, HAL obfuscates the backend that captures data packets, allowing upper-level callers to remain indifferent to whether the backend is rooted in PCAP. They can consider HAL essentially as a 'black box' that receives data packets.

HAL exposes an interface, `NextPacket()`, enabling higher-level callers to construct a thread pool to process data packets. The `NextPacket()` function returns a Packet, which is an Ethernet protocol data packet represented as a byte array. The extraction of information such as the MAC address, IP address, and data contained within this byte array, following the Ethernet and IPv6 standards, falls within the purview of the Network Parser. For example, at the very beginning, we implemented the parser for ICMPv6 datagram

for system initiation - nodes exchange their states through Neighbour Discovery Protocol (NDP) [5].

We have implemented a highly abstracted interface provided by the lower-level Hardware Abstraction Layer (HAL) and exposed it to the upper protocol implementation. Specifically, in the 'Parse' family of functions, we offer a set of interfaces that enable standard parsing of datagrams. These interfaces unpack the datagram, read the byte stream, and convert it into an internal, readable data structure. Additionally, these interfaces implicitly convert the Network Byte Order to Host Byte Order, and conversely, the packing interfaces perform the corresponding conversion. These functionalities provide a fundamental infrastructure for the protocol layer.

At the apex of the engine architecture lies the Protocol module, which is designed to be modular and flexible. In adherence to the tenets of the Go programming language, the Protocol module is built around the concept of Interfaces. A module is classified as a Protocol module if it processes data packet information using the Network Parser, updates the routing table, or forwards data packets in compliance with the standards specified in RFC documents. Currently, Qiao's supported Protocol modules include RIPng and DHCP, with plans to expand to a broader range of RFC protocols in the future. Our work aims to facilitate the rewriting, reuse, and transplantation of existing protocols to suit our HAL infrastructure, while also leaving ample room for redesign. In this manner, our solution prioritizes programmer convenience, an easily extendable library, and relative hardware and operating system independence to prevent the router from being constrained by rigid solutions.

A. Hardware Abstraction Layer (HAL)

HAL is designed to efficiently manage and operate network interfaces and protocol stacks. HAL offers a range of functionalities, including handling network interfaces, packet capture and transmission, IPv6 address management, and NDP implementation.

HAL is capable of identifying and displaying all available network interfaces in the system. It retrieves detailed information about these interfaces, such as name, index, flags, Maximum Transmission Unit (MTU), and hardware address (MAC address).

Utilizing the GoPacket [14] library, HAL can capture and parse packets on specific network interfaces. Additionally, it can construct and transmit IPv6 packets. HAL is responsible for recognizing and maintaining IPv6 addresses in the system. It automatically generates and assigns IPv6 addresses for network interfaces, including link-local and globally unique addresses. HAL processes neighbor solicitation and neighbor advertisement messages to update the NDP cache table. HAL also provides timer functionalities for periodically updating and maintaining the NDP cache table, ensuring that the information is always up-to-date.

With these features, HAL offers a flexible, extensible hardware abstraction layer for networking applications, making it easier for developers to handle low-level network operations. Moreover, the modular design of HAL allows developers to easily extend or modify functionalities according to specific requirements.

By adopting HAL architecture, developers can focus on the functionality and performance optimization of their routing applications without having to worry about hardware compatibility issues. Furthermore, hardware vendors can provide dedicated drivers and HAL implementations tailored to their products, which can enhance their competitiveness and market share.

B. Network parser

The primary function of the Network Parser is to extract relevant information from specific network protocols from the continuous byte array received from the HAL. Contrasting with the operations in C language, Go language does not have pointer offset operations. To parse network byte arrays in big-endian order, the Network Parser employs a mechanism known as move-parse. It possesses an attribute known as 'pointer', which progresses by the length of the corresponding data type after each parse operation.

To accommodate a wide array of network protocols, such as Ethernet, IPv6, ICMP, RIPng, DHCP, and others, the Network Parser is equipped with parsing and extraction functionalities for multiple data types, including but not limited to uint8, uint16, uint32, EthernetAddress, and IPv6Address. Certain network protocols, like DHCP and ICMP, consist of several Variable-Length Options. To offer flexibility to the upper-level callers in dealing with these scenarios, the Network Parser further provides the capability to parse any arbitrary N number of bytes.

Take ICMPv6 for example, we use multiple Network Parser functions to parse the ICMPv6 header:

```
func (p *NetParser) ParseICMPv6Header() \
            (ICMPv6Header, error) {
    header := ICMPv6Header{}
    u8, err := p.ParseU8()
    if err != nil {
      return header, err
    }
    header.Type = ICMPv6Type(u8)
    header.Code, err = p.ParseU8()
    if err != nil {
      return header, err
    }
    header.Checksum, err = p.ParseU16()
    if err != nil {
      return header, err
    }
    header.RestOfHeader, err = p.ParseU32()
    if err != nil {
      return header, err
    }
    return header, nil
}
```

The Network Parser plays a pivotal role in streamlining the development of Variable-Length Options in IPv6 protocols, such as the format of DHCPv6 Options [6], by enabling efficient parsing of the packet data and extraction of the required information. Hence, from the fixed length option-len field, we could notice the length of option data, and call standard API `ParseNBytes(ByteNum)` to extract option data. In addition to this, we also provide a series of standardized interfaces (methods related to the struct `NetParser`) to handle fixed-length fields:

```
ParseU8() (uint8, error)
ParseU16() (uint16, error)
ParseU32() (uint32, error)
ParseBuffer() (Buffer, error)
ParseNBytes(N int) ([]byte, error)
ParseSkipNBytes(N int) error
ParseIpv6Addr() (Ipv6Addr, error)
ParseEthernetAddr() (EthernetAddr, error)
Eof() bool
```

C. Protocol

The protocol library package includes a library of internet protocols like RIPng, DHCPv6, eui64, etc. These upper-layer protocol libraries can fully utilize the network interfaces provided by HAL to implement data exchange.

Currently, Qiao has implemented two fundamental routing protocols: RIPng and DHCP.

Originally, Qiao was designed to provide lightweight and flexible solutions for small to medium-sized networks, aligning seamlessly with the design objectives of RIPng. As a universal software router, Qiao can also serve as a testbed, promoting innovative protocol research in network studies. Currently, the most commonly adopted network protocol among ISPs is BGP, around which there has been substantial research. In the future, Qiao plans to provide support for the BGP protocol.

IV. Implementation

Qiao is designed with a high degree of portability in mind and can operate on any Unix-like system across various architectures. A key attribute of Qiao's design strategy is its focus on compact, efficient code. Compared to well-established solutions like Bird and FRR, Qiao's codebase required for implementing the RIPng protocols is substantially more compact. Specifically, Qiao only requires around 447 lines of code, representing a reduction of approximately 74.9% and 92.2% compared to Bird [9] and FRR [13] respectively as shown in Table 1. This streamlined design strategy not only renders Qiao more lightweight but also simplifies the processes of code maintenance and future enhancements.

In addition to its compact size, Qiao's implementation exhibits commendable performance metrics, robustness, and stability. The reduced codebase size does not compromise the software's ability to deliver reliable and efficient service. Qiao manages to maintain this balance between minimalistic design and maximum functionality by employing optimized algorithms, smart memory management, and a clear separation of responsibilities among its modules.

TABLE I
Comparison of lines of code

Router Software	Lines of Code
Qiao	447
Bird	1781
FRR	5746

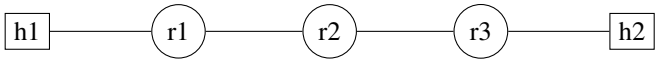

Fig. 2. The topology includes hosts h1 and h2, and routers r1, r2, and r3.

The network parser, for example, is solely responsible for data packet parsing, leaving the protocol module to handle protocol-specific actions. This leads to cleaner, more readable code, and a lower chance of error due to the separation of concerns.

Qiao's design also makes it conducive for ongoing and future development. The small and neat codebase can easily be extended or modified, enabling the swift addition of support for more protocols or the inclusion of newer features, thus increasing Qiao's versatility and adaptability in handling evolving networking needs.

The practicality of Qiao's design and its lean implementation makes it a promising contender in the software-defined networking space, capable of delivering robust, efficient service while also providing a platform conducive to ongoing innovation and development. This approach paves the way for Qiao's future enhancements and its potential to shape the networking solutions of tomorrow.

V. Environment Setup

In the context of experimentation, Qiao operates on an x86_64 platform with 64G of memory, under a virtual network topology built using Mininet. Thanks to Qiao's platform-agnostic characteristics, it can also function on other architectures running a Linux environment.

Because Qiao is based on IPv6 addressing, and considering that Mininet doesn't support IPv6, we have adopted ipmininet for environment setup, which serves as an extension of the Mininet software. Currently, Qiao supports Go version 1.18 and above.

As shown in Figure 2, we constructed the following network topology for testing to validate Qiao's interoperability and functional correctness. The code for building the topology is as follows:

```
class MyTopology(IPTopo):
  def build(self, *args, **kwargs):
    h1 = self.addHost('h1', use_v4=False)
    r1 = self.addHost('r1', use_v4=False)
    r2 = self.addHost('r2', use_v4=False)
    r3 = self.addHost('r3', use_v4=False)
    h2 = self.addHost('h2', use_v4=False)
    h1r1 = self.addLink(h1, r1)
```

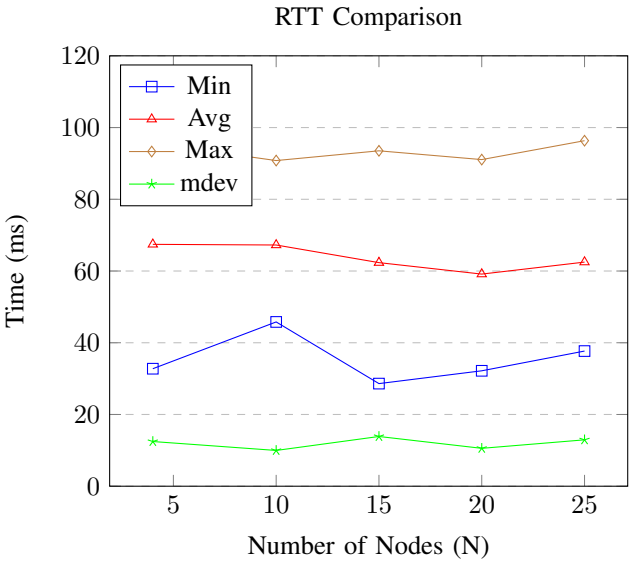

Fig. 3. The impact of the number of concurrent connections on RTT

```
r1r2 = self.addLink(r1, r2)
r2r3 = self.addLink(r2, r3)
r3h2 = self.addLink(r3, h2)
super().build(*args, **kwargs)
```

In the environment setup script, we command each router to run the Qiao program in the background. The relevant lines in the script look like this:

```
net['r1'].cmd('nohup ./qiao &')
net['r2'].cmd('nohup ./qiao &')
net['r3'].cmd('nohup ./qiao &')
```

This starts the Qiao program on each of the routers (r1, r2, r3) in the background, allowing for independent operation without interference from the main process.

In addition, we utilized xterm to issue commands to each node individually. Initially, we instructed routers r1 and r3 to run the Bird routing software, written in C. Simultaneously, the Qiao routing software was written and tested on router r2. We then attempted to ping host h2 from host h1 to validate the correctness of Qiao via this diffTest approach.

Once Qiao's correctness was assured, we initiated the Qiao program on routers r1 and r3 in the background as well to verify its interoperability. During the debugging process, we utilized the Wireshark software to capture packets emitted by each node. This helped in monitoring the network traffic and further fine-tuning Qiao's performance.

VI. Evaluation

Qiao is designed with a focus on simplicity and ease of implementation, rather than pursuing extreme performance. This pragmatic approach makes it more accessible and manageable, especially for smaller or less complex networks.

Since Go language lacks the feature of pointer offsetting, it cannot parse protocols as directly as C language, which can lead to some performance loss. We evaluated the performance comparison between BIRD and Qiao under two conditions. When the router is connected to only one host, BIRD takes 10ms to forward packets, while Qiao takes 15ms, resulting in a 30% performance degradation for Qiao. However, when the router is connected to multiple hosts, thanks to the lightweight nature of Go routines, Qiao naturally supports high-concurrency scenarios. As the number of packets concurrently processed by the router increases, the time required to forward a single packet is shown in Figure 3.

In the empirical assessment, an intricate set of experiments was meticulously designed and executed to ascertain the performance characteristics of the Qiao under varying degrees of concurrent network connections. The experimentation milieu encompassed a multitude of network scenarios, each manifested by a discrete number of hosts, each host operational on an isolated physical server, with intercommunication facilitated via the Qiao router.

The methodological choice of employing the ping utility as the central probing instrument in our experiments was premised on its ubiquitous application in quantifying the round-trip times (RTTs) within networked systems. Each host was systematically configured to dispatch ping requests to its peers, thereby emulating a scenario of network congestion. Detailed metrics about the minimum, average, and maximum RTTs for each initiated ping operation were meticulously recorded, complemented by the calculation of the mean deviation (mdev) of the RTTs.

Figure 3 encapsulates the findings of these empirical undertakings. The results shed light on the relatively unwavering performance of the Qiao router across a spectrum of concurrent connection levels, demonstrating minor fluctuations in RTT, which is a testament to the router's robust ability in proficiently handling concurrent connections. Furthermore, the consistency in mean deviation (mdev) values indicates the resilience of the router in maintaining minimal oscillations in network latency, despite an augmenting level of concurrent connections. This underlines the Qiao router's capacity in sustaining optimal performance even under escalating network loads.

In summation, the experimental outcomes underscore the efficacy and stability of the Qiao router, reinforcing its robustness and adaptability in high-concurrency network environments, thus validating our initial hypothesis.

VII. Related Works

There are already some mature routing software library solutions, and we are very grateful for the guidance these projects can provide us. For example, the BIRD project, written in C, has developed a fully functional dynamic IP routing daemon primarily targeted on (but not limited to) Linux, FreeBSD, and other UNIX-like systems and distributed under the GNU General Public License. Another approach, FRRouting (FRR), also in C, is a free and open-source Internet routing protocol suite for Linux and Unix platforms. It implements BGP [4], OSPF [3], RIP, IS-IS, PIM, LDP, BFD,

Babel, PBR, OpenFabric, and VRRP, with alpha support for EIGRP and NHRP.

Click [16] is a modular router software. Its code is designed at the granularity of "elements" and can specify the functionality of the router by writing configuration files. It supports multiple network protocols and the DPDK backend. Click's performance was critiqued in the context of OpenFlow [19]. Click has been integrated into various network simulators such as NS-3 [20], and there are also some NFV-related works based on Click, such as ClickOS [18].

xBGP [23] is a vendor-neutral API that exposes the key data structures and functions of any BGP implementation. xBGP explores how to provide more flexibility to routers. xBGP has extended FRRouting and BIRD by using virtual machine technology and exposing several networking function APIs.

Routers have always been seen as single-node systems, and there have been attempts in the field of network research to extend routers to multi-node systems [10]. However, due to the considerable difficulty in upgrading network equipment, distributed routing has not been widely used.

VIII. Conclusion

We have proposed a router software based on the high-level language, Go, and explored the benefits brought by a router written in a highly concurrent and modular high-level language.

With Qiao, we demonstrate how the properties of Go's robust standard library, built-in concurrency model, and the language's simplicity can be leveraged to develop lightweight and flexible router software that is easy to read, write, and maintain. We believe that this approach can significantly reduce the barriers to entry for network researchers, developers, and even students looking to delve into the inner workings of routing protocols.

Qiao delivers reasonable performance that makes it a valid solution for small to medium-sized networks. At present, Qiao only designs RIPng and DHCP protocols for small and medium-sized networks, exploring the application of the Go language on routers. In the future, Qiao plans to support more routing protocols, such as BGP. Qiao has designed different layers in its code to allow developers to extend it more easily. To improve the performance of forwarding packets, Qiao plans to support more efficient network card backends, such as DPDK, without affecting their expandability.

Qiao is a decentralized router that does not require a centralized control plane to control the router's ability to forward data. As Qiao is software-based, it can be easily upgraded and expanded. We also plan to study more flexible routing protocols based on Qiao in the future.

In conclusion, the development of Qiao has shown that a high-level language like Go can be effectively utilized for router software development, paving the way for easier, more flexible, and more accessible network software in the future. As we continue to expand and improve Qiao, we look forward to seeing it contribute to the evolution of network technology and protocol research.

Acknowledgment

This project is partially inspired by The BIRD Internet Routing Daemon Project and TsingHua Router-Lab [22].

References

[1] Dynamic Host Configuration Protocol. RFC 2131, 1997.

[2] RIPng for IPv6. RFC 2080, 1997.

[3] OSPF Version 2. RFC 2328, 1998.

[4] A Border Gateway Protocol 4 (BGP-4). RFC 4721, 2006.

[5] Neighbor Discovery for IP version 6 (IPv6). RFC 4861, 2007.

[6] Dynamic Host Configuration Protocol for IPv6 (DHCPv6). RFC 8415, 2018.

[7] P. A. Bob Wheeler. Dpu-based hardware acceleration: A software perspective. 2021.

[8] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker. P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95, jul 2014.

[9] z. CZ.NIC. Bird internet routing daemon, 2020. gitlab.nic.cz/labs/bird.

[10] M. Dobrescu, N. Egi, K. Argyraki, B.-G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy. Routebricks: Exploiting parallelism to scale software routers. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, SOSP '09, page 15–28, New York, NY, USA, 2009. Association for Computing Machinery.

[11] N. Feamster, J. Rexford, and E. Zegura. The road to sdn: An intellectual history of programmable networks. *SIGCOMM Comput. Commun. Rev.*, 44(2):87–98, apr 2014.

[12] A. D. Ferguson, S. Gribble, C.-Y. Hong, C. Killian, W. Mohsin, H. Muehe, J. Ong, L. Poutievski, A. Singh, L. Vicisano, R. Alimi, S. S. Chen, M. Conley, S. Mandal, K. Nagaraj, K. N. Bollineni, A. Sabaa, S. Zhang, M. Zhu, and A. Vahdat. Orion: Google's Software-Defined networking control plane. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 83–98. USENIX Association, Apr. 2021.

[13] T. L. Foundation. Frrouting project, 2020. https://frrouting.org/.

[14] Google. gopacket, 2022. https://github.com/google/gopacket.

[15] Intel. dpdk, 2023. https://www.dpdk.org/.

[16] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The click modular router. *ACM Trans. Comput. Syst.*, 18(3):263–297, aug 2000.

[17] Z. Lan. A comprehensive review of fault-tolerant routing mechanisms for the internet of things. *International Journal of Advanced Computer Science and Applications*, 14(7), 2023.

[18] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici. ClickOS and the art of network function virtualization. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 459–473, Seattle, WA, Apr. 2014. USENIX Association.

[19] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, mar 2008.

[20] ns3 contributors. ns3: Network simulator, 2023. https://www.nsnam.org/.

[21] A. L. K. K. J. A. I. U. Shah, Zawar and S. Singh. Routing protocols for mobile internet of things (iot): A survey on challenges and solutions. In *Electronics 2021, 10, 2320*, Oct. 2021.

[22] THU-CS-Lab. Router-lab. https://github.com/thu-cs-lab/Router-Lab, accessed June 29, 2023.

[23] T. Wirtgen, T. Rousseaux, Q. D. Coninck, N. Rybowski, R. Bush, L. Vanbever, A. Legay, and O. Bonaventure. xBGP: Faster innovation in routing protocols. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 575–592, Boston, MA, Apr. 2023. USENIX Association.

[24] S. Yan, X. Wang, X. Zheng, Y. Xia, D. Liu, and W. Deng. Acc: Automatic ecn tuning for high-speed datacenter networks. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, SIGCOMM '21, page 384–397, New York, NY, USA, 2021. Association for Computing Machinery.