DIP: Unifying Network Layer Innovations using Shared L3 Core Functions

Ziqiang Wang
Southeast University
Nanjing, China
ziqiangwang@seu.edu.cn

Zhuotao Liu
Tsinghua University and
Zhongguancun Laboratory
Beijing, China
zhuotaoliu@tsinghua.edu.cn

Xiaoliang Wang
Capital Normal University
Beijing, China
wangxiaoliang@cnu.edu.cn

Songtao Fu
Tsinghua University
Beijing, China
fust18@mails.tsinghua.edu.cn

Ke Xu
Tsinghua University and
Zhongguancun Laboratory
Beijing, China
xuke@tsinghua.edu.cn

ABSTRACT

The IP protocol has made a great contribution to the development of the Internet and has become the narrow waist of the Internet. However, the fixed packet processing of IP hinders the functional expansion and evolution of the Internet. In order to solve the rigidity of the Internet, our community has proposed various new L3 protocols to better support various network functions at the network layer. In this paper, we propose DIP (Dynamic Internet Protocol), a novel primitive to unify these protocols. DIP builds a common network function core shared by these L3 protocols based on a new L3 function core primitive, named Field Operation (FN). With FNs, each standalone L3 protocol can be decomposed into a combination of multiple FNs, and meanwhile it is feasible to compose various FNs to realize new (derived) L3 protocols. We demonstrate the feasibility of DIP by realizing five radically different network layer protocols[1]: the canonical IP forwarding, NDN [41], XIA [12], OPT [16], and NDN+OPT (a derived L3 protocol combining the merits of both NDN and OPT). We implement a prototype of DIP and evaluate its forwarding performance.

CCS CONCEPTS

• **Networks** → **Network protocol design**;

KEYWORDS

Network Layer Innovations, Field Operations

ACM Reference Format:
Ziqiang Wang, Zhuotao Liu, Xiaoliang Wang, Songtao Fu, and Ke Xu. 2022. DIP: Unifying Network Layer Innovations using Shared L3 Core Functions. In *The 21st ACM Workshop on Hot Topics in Networks (HotNets '22), November 14–15, 2022, Austin, TX, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3563766.3564092

1 INTRODUCTION

The Internet architecture with IP narrow-waist design has achieved great success. It shields the diversity of heterogeneous underlying technologies and supports the thriving top-level ecosystem, ensuring the ubiquitous connection and vitality of the Internet. However, innovations about the Internet infrastructure itself are difficult. The reason could be attributed to the inability of the fixed Internet architecture to accommodate innovations in core mechanisms, such as the inability to dynamically deploy a non-IP addressing model better suited to the mobile scenarios. As a result, many Internet improvement mechanisms are ad-hoc in nature, based on various patching through middleboxes or/and traffic indirection.

In response to these problems, the academic community proposed a line of research in architecture customization and coexistence to eliminate the deadlock of innovation by increasing the diversity of Internet functions. Pioneers have contributed a lot of valuable experience and substantial results achievements [5, 12, 22, 34]. For example, active network [34] proposes "capsules" - program fragments that are executed at each network router/switch they traverse and allow users to inject customized programs into the nodes

[1] We clarify that NDN and XIA are next-generation Internet architecture proposals. In this paper, whenever we referred to NDN and XIA, we mean their core packet forwarding protocols.

HotNets '22, November 14–15, 2022, Austin, TX, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9899-2/22/11…$15.00
https://doi.org/10.1145/3563766.3564092

of the network. Active network sparked various subsequent work on network programmability, such as OpenFlow and SDN [23], P4 [3] and runtime programmable devices [10]. In addition to active network, the community also proposed different Internet architectures to advocate radically different Internet designs. For instance, XIA [12, 24] replaces the single address with a directed acyclic graph and uses the "fallback" technology to support multi-protocol coexistence. NDN [41] uses data names instead of IP addresses for better content delivery with interest packets and data packets. SCION [42] is a security-oriented architecture that enables explicit path control and selection to achieve secure inter-domain routing.

Besides these new Internet architecture proposals, a series of security-oriented protocols (such as NetFence [19], OPT [16], MiddlePolice [20], EPIC [17]) also proposed to change the L3 protocols. For instance, NetFence inserts a slim customized header between L3 and L4 to emulate congestion control (*i.e.,* additive increase and multiplicative decrease, AIMD) inside the network to mitigate DDoS attacks (MiddlePolice shares a similar design principle, with a more deployable architecture and more flexible traffic control protocols). OPT and EPIC, designed based on SCION, requires on-path routers to verify and update the cryptographically generated code carried customized packet headers to achieve source validation and path authentication.

In this paper, we argue that unifying the key merits of these prior protocols is (at least) equally important as innovating yet another new L3 protocol. Trotsky [22] attempts to move the service model of the Internet from the predefined L3 to configurable (software) L3.5. Yet, the unification discussed in this paper differs significantly from Trotsky. On the one hand, we focus on researching a novel primitive to natively implement a wide range of radically different L3 protocols, including the canonical IP protocol, the addressing protocols defined in NDN [41] and XIA [12], and security protocols such as OPT [16]. On the other hand, because of the fine-grained primitive (referred to DIP hereafter), the protocol unification in our paper can essentially merge or consolidate different L3 protocols. Therefore, DIP enables *derived L3 protocols* with merits that are more advanced than any individual L3 protocols. For instance, DIP derives NDN+OPT from NDN [41] and OPT [16], which adds security ingredients to the original NDN design.

Contributions. We propose DIP, a novel primitive to enable a wide range of network functions on the network layer. We position DIP as an orthogonal supplement to Trotsky, and meanwhile a concrete realization of a limited, yet more deployable, form of active networking based on the emerging programmable switches. We demonstrate the capability of DIP through the concrete instantiation of 5 radically different L3 protocols, including IP, NDN, OPT [16], XIA [12] and a derived L3 protocol combining both NDN and OPT.

2 DIP DESIGN

This section introduces the design of DIP. We first introduce field operation (FN), the basic protocol primitive in DIP to enable dynamic packet processing by composing various FNs as desired. Then, we show a draft of the DIP header and describe the host and per-hop DIP-packet processing designs. Finally, we address several design design concerns.

2.1 Field Operations

To unify (often) mutually incompatible L3 protocols, DIP needs to build a common network function core shared by these protocols. Such an L3 network function core should simultaneously achieve decomposability and composability: each standalone L3 protocol can be decomposed into a combination of multiple core functions, and meanwhile, it is feasible to compose various fine-grained core functions to implement new (derived) L3 protocols.

The network community previously discussed a conceptually similar L3 function core in the context of enabling flexible L3 forwarding. Notably, I3 [28] proposes to use a Peer-to-Peer lookup service to implement fine-grained packet processing at Rendezvous points; Capsules [34] treat each network message as a program and every capsule contains a program fragment (with at least one instruction) that may include embedded data; Network pointers [35] are similar to C-pointers and each data packet becomes a thread of control that works its way through a chain of dereferencing operations. Despite their flexibility, neither of these proposals was designed to decompose and compose different L3 protocols.

In DIP, we abstract a new L3 function core primitive, named Field Operation (FN), to define fine-grained packet processing functions on the network layer. Each FN consists of two elements: a target field and an operation to be applied on the corresponding target field. In particular, a target field is a sequence of consecutive bits present in the packet header with specific meaning (*e.g.,* the destination address field in IP forwarding, and the MAC-protected congestion control tag in NetFence [19]. The operation is a functional module that takes the field as input and performs pre-defined calculations or matches, and then modifies the packet field (*e.g.,* update the verification tags in OPT [16] and EPIC [17]) or determine the packet fate (*e.g.,* discard the packet if the carried path authentication tags are incorrect).

Surprisingly, this simple abstraction of FNs achieves excellent decomposability and composability desired in DIP. As we shall see in § 3, we design four radically different L3 protocols uniformly using FNs (*i.e.,* IP, NDN, OPT [16] and XIA [12]) and one derived protocol that combines NDN and OPT. Another benefit of the FN abstraction is that it is

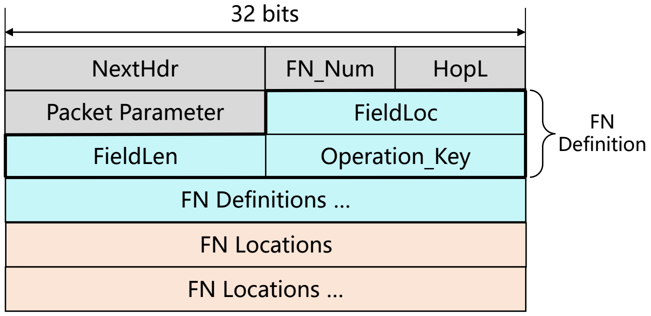

Figure 1: The structure of a DIP packet header.

hardware-friendly. The emerging programmable switch technology, centering around Protocol-Independent Switch Architecture (PISA), enables programmable switching pipeline in which one can define customized packet header manipulations (*e.g.,* dynamic packet header parsing). Such hardware innovations enable highly performant FN implementations, as shown in § 4.2.

2.2 A Draft of DIP Header

Figure 1 plots a draft of DIP packet header, including the basic header (colored in gray), FN definitions (colored in blue), and FN locations (colored in orange).

The basic header of DIP consists of four fields: next header, FN number, hop limit, and packet parameter. We use the FN number to indicate the number of FNs defined in this packet. There are currently two attributes defined in the packet parameter field. The lowest bit indicates whether the operation modules can be executed in parallel and this flag is used to improve packet processing speed when the modular parallelism technique [31, 32] is used. The higher ten bits represent the length of FN locations and the remaining five bits are reserved for other use.

Each FN is specified by three fields in the packet header: field location, field length, and operation key. The actual packet location where an FN reads from and writes to is defined as FN location, which is specified by the FN's field location and field length. The operation key indicates the operation that needs to be performed on the FN location. The highest bit of the operation key field is a *tag* bit to indicate whether the operation should be performed by the router or the host. Since the triplet structure of an FN is fixed, we can use the FN number and the FN locations length to derive the DIP header length.

2.3 Packet Processing Logic

The actual packet processing logic in DIP is determined by the FNs carried in the packet header. In this section, we use the following example to illustrate the packet processing logic in DIP: a host requests content with content name, and meanwhile it verifies the content's source and the network path used to deliver the content are secure.

Algorithm 1: The packet processing logic in router

```
Input: pkt
parse basic DIP header (FN_Num and FN_LocLen);
parse FN[] according to FN_Num;
extract FN_Loc according to FN_LocLen;
for $i \leftarrow 1$ to FN_Num do
    if FN[i].tag == 1 then
        /* skip host operation */
        continue;
    else
        target_field ← FN_Loc(FN[i].FieldLoc, FN[i].FieldLen);
        switch FN[i].key do
            ...
            case 4 : $\mathcal{F}_{\text{FIB}}(target\_field)$;
            case 5 : $\mathcal{F}_{\text{PIT}}(target\_field)$;
            case 6 : $\mathcal{F}_{\text{parm}}(target\_field)$;
            case 7 : $\mathcal{F}_{\text{MAC}}(target\_field)$;
            case 8 : $\mathcal{F}_{\text{mark}}(target\_field)$;
            ...
end processing;
```

Available FNs. After the host is connected to an accessed AS, it uses bootstrapping mechanisms (similar to DHCP) to get the set of available FNs. To illustrate the workflow of DIP, here we assume that all network devices are DIP-capable and have the same FN configuration. We discuss the heterogeneous deployment case in § 2.4. Table 1 summarizes the FNs discussed in this paper. We recommend the ASes should agree upon a dedicated operation standard for enabling new field operations. One readily deployable mechanism to globally propagate supported FNs among ASes is relying on BGP communities, which is part of our future work.

Host Constructions. Before sending the data packets, the host needs to formulate appropriate FNs in the packet header considering both the required network services and the supported FNs. Here, we consider the host wants to request a file named "hotnets.org" and it retrieves the relevant FNs defined in Table 1, including $\mathcal{F}_{32\_match}$, $\mathcal{F}_{128\_match}$, $\mathcal{F}_{\text{FIB}}$, $\mathcal{F}_{\text{PIT}}$, $\mathcal{F}_{\text{parm}}$, $\mathcal{F}_{\text{MAC}}$, $\mathcal{F}_{\text{mark}}$ and $\mathcal{F}_{\text{ver}}$. The host requests this file using the $\mathcal{F}_{\text{FIB}}$, which is a forwarding information base and performs the longest prefix match with the content name. It sets the file name (*i.e.,* "hotnets.org") in the FN locations, and takes the $\mathcal{F}_{\text{FIB}}$, with the FN key as 4 and the FN tag as 0, as the desired operation on the router. After the host finds this file source (*e.g.,* a server), it wishes to verify the authenticity

Table 1: Field operations in the DIP prototype

operation	notation	key
32-bit address match	$\mathcal{F}_{32\_match}$	1
128-bit address match	$\mathcal{F}_{128\_match}$	2
source address	$\mathcal{F}_{source}$	3
forwarding information base match	$\mathcal{F}_{FIB}$	4
pending interest table match	$\mathcal{F}_{PIT}$	5
load parameters	$\mathcal{F}_{parm}$	6
calculate MAC	$\mathcal{F}_{MAC}$	7
mark update	$\mathcal{F}_{mark}$	8
destination verification	$\mathcal{F}_{ver}$	9
parse the directed acyclic graph	$\mathcal{F}_{DAG}$	10
handle intent	$\mathcal{F}_{intent}$	11

of the source and the network path. Towards this end, the server sets some authentication tags in the FN locations and uses the $\mathcal{F}_{MAC}$, which is a cryptographic computing module (*e.g.,* 2EM [2]), to instruct the on-path routers to update the authentication tags. The exact packet header construction for integrating content delivery and path authentication is detailed in § 3.

Router Processing. We present the packet processing logic in Algorithm 1 and the operation modules involved are described in Table 1. Upon receiving a packet, the router will first parse the basic DIP header to get the number of FNs (*FN_Num*) and the length of FN locations (*FN_LocLen*). Then, the router parses the FN definitions (*FN[]*) carried in the packet header in the form of triples according to the *FN_Num* and extracts the FN locations (*FN_Loc*) according to the *FN_LocLen*. After that, the router operates on the target fields (*target_field*) according to the FN definitions (*i.e.,* field location, field length, and operation key). In our example, the router will perform $\mathcal{F}_{FIB}$ with the target field (*i.e.,* "hotnets.org") to forward the packet and find the file source. After that, the router will perform $\mathcal{F}_{FIT}$, which is a pending interest table and performs the longest prefix match with the content name, to transfer the file to the host. Besides, the router will perform three authentication operation (*i.e.,* $\mathcal{F}_{parm}$, $\mathcal{F}_{MAC}$, $\mathcal{F}_{mark}$) one by one to update the authentication tags. Finally, the host receives and verifies the packet by performing $\mathcal{F}_{ver}$. The detailed authentication operation instruction is in § 3.

2.4 Additional Design Concerns

Incremental deployment. In the early stage of deployment, two DIP domains may not be directly connected. One could use tunneling technology [6, 8] to build end-to-end path across DIP-agnostic domains.

Backward compatibility and heterogeneous configuration. We can make lightweight changes on the DIP packet header to be compatible with backward protocols. In particular, the existing network protocol header can be viewed as an FN location in the DIP. For example, when a DIP host connects to another host using IPv6. We set the IPv6 header in the FN location part and define the corresponding forwarding operations. Afterward, the border router can remove the basic header and FN definitions, so that the packet is routed only based on the FN operations that are recognized by the legacy devices. Similarly, to process packets from a legacy domain, the inbound border router needs to add back the DIP basic header and FN definitions. In case of heterogeneous FN configurations among ASes, the inbound router may receive a DIP packet carrying an FN that the AS has not supported yet. If this FN requires all on-path ASes to particulate (*e.g.,* the FN designed for path authentication [16]), the router should return *an FN unsupported message* to notify the source through a mechanism similar to ICMP. Otherwise, the router can simply ignore this FN.

Security. Here we discuss two major security concerns of DIP. First, the processing of the packet is dynamically customized according to the FNs in the packet header, so we should prevent packet processing from exhausting the router state. Enforcing a hard limit for packet processing time and per-packet state consumption is enough to prevent such attacks. Second, an adversary may strategically combine FNs to launch attacks. For instance, an attacker can use both $\mathcal{F}_{FIB}$ and $\mathcal{F}_{PIT}$ in one packet and carry maliciously constructed data to pollute the node's content cache. Nodes can enable source label verification designs (*e.g.,* [15], implemented as a new FN $\mathcal{F}_{pass}$) to defend against this attack. Although enabling $\mathcal{F}_{pass}$ all the time is expensive, DIP allows the network operators to dynamically adjust security policies based on network conditions. For instance, $\mathcal{F}_{pass}$ can be enabled on the fly upon detecting content poisoning attacks.

3 PROTOCOL REALIZATION USING DIP

We describe how to realize various L3 protocols based on DIP by properly constructing DIP headers.

IP Forwarding. We first use DIP to achieve the canonical IPv4 and IPv6 forwarding. In the FN definition part of the DIP header, we use $\mathcal{F}_{128\_match}$ and $\mathcal{F}_{32\_match}$ to instruct the router to perform 128-bit/32-bit address matching and forwarding and use $\mathcal{F}_{source}$ to specify the source address. We set the destination address in the lower 128/32 bits of the FN locations and the source address in the upper 128/32 bits, so the FN triples used in our prototype are (*loc*: 0, *len*: 128, *key*: 1)/(*loc*: 0, *len*: 32, *key*: 2) and (*loc*: 128, *len*: 128, *key*: 3)/(*loc*: 32, *len*: 32, *key*: 3).

NDN [41]. NDN is a content delivery network, in which there are two types of packets: interest packets and data packets. For the first type of packets, according to the NDN routing protocol [41], the router records its receiving port in the PIT (pending interest table) and matches it in the

FIB (forwarding information base) with the content name to determine the forwarding port[2]. For the data packets, the router looks up the content name in the PIT and forwards it to the recorded request port (match hit) or discards the packet (match miss). We abstract the packet processing of NDN into two operation modules: $\mathcal{F}_{\text{FIB}}$ and $\mathcal{F}_{\text{PIT}}$, and use the following two FN triples (*loc*: 0, *len*: 32, *key*: 4) and (*loc*: 0, *len*: 32, *key*: 5) to explicitly customize NDN packet processing and set the content name in the FN locations.

OPT [16]. In addition to basic packet forwarding functions, the research community has proposed various related functions (*e.g.,* source and path verification [11, 16, 38]). Here we use OPT [16] as an example to demonstrate how to use FNs to explicitly customize packet processing that supports source and path verification functions. Packet processing in OPT mainly involves MAC calculation and updates of cryptographically-signed tags. After receiving a packet, the router will derive a dynamic key form session ID[3] in the packet header with its local key. Then the router uses the dynamic key, which is shared with the host, to recalculate and update the tags. Finally, the host receives the packet and recalculates these tags to verify the legitimacy of the source and path used. We abstract this process into four operation modules: $\mathcal{F}_{\text{parm}}$, $\mathcal{F}_{\text{MAC}}$, $\mathcal{F}_{\text{mark}}$ and $\mathcal{F}_{\text{ver}}$. We set the OPT header in the FN locations and use the triple (*loc*: 128, *len*: 128, *key*: 6) to instruct the router to generate the key and load other parameters (*e.g.,* previous validator node label, which will be used in the MAC operation). Then we use the FN triples (*loc*: 0, *len*: 416, *key*: 7) and (*loc*: 288, *len*: 128, *key*: 8) to recalculate and update the tags. Finally, we use the triple (*loc*: 0, *len*: 544, *key*: 9) to instruct the destination host to verify the packet source and path.

NDN+OPT. OPT is a security-oriented protocol and uses authentication tags to achieve source validation and path authentication in the IP network. With FNs, we can integrate OPT with NDN to derive a secure content delivery network, which is called NDN+OPT. In particular, we compose the following FN modules ($\mathcal{F}_{\text{FIB}}$, $\mathcal{F}_{\text{PIT}}$, $\mathcal{F}_{\text{parm}}$, $\mathcal{F}_{\text{MAC}}$, $\mathcal{F}_{\text{mark}}$ and $\mathcal{F}_{\text{ver}}$) to construct the DIP packet header for NDN+OPT.

XIA. We use the $\mathcal{F}_{\text{DAG}}$ and $\mathcal{F}_{\text{intent}}$ FN modules to realize the complex packet processing logic in XIA. We set the header of XIA in the FN locations and use these two operation modules to parse the directed acyclic graph and handle the intent.

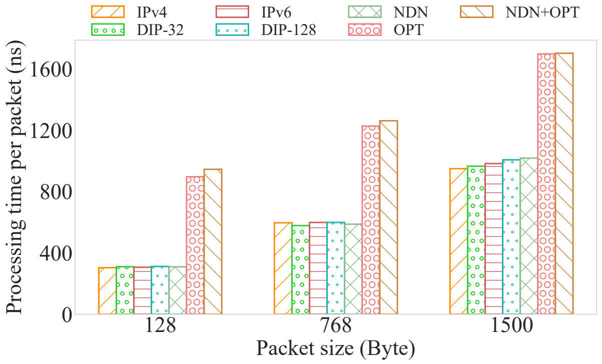

Figure 2: Packet processing time in the DIP prototype.

4 EVALUATION

4.1 Implementation

We implement a prototype DIP with the commodity Barefoot Tofino programmable switch S9180-32X. On account of the current hardware limitations of the commodity Barefoot Tofino programmable switch, it was challenging to implement a loop to invoke the operation modules. We use the simple "if-else" statement with FN_Num to determine how many field operations to perform. The field slices in Barefoot Tofino are restricted to not using variables, therefore we preset some fixed field slices and use some tables to match the target field. Runtime programmability has not yet been implemented on Barefoot Tofino, so we pre-write the required operation modules on the data plane and use the operation key to match these operation modules.

The header length of OPT [16] packet varies with the path length and we use one hop for evaluation. For the MAC operation in OPT, we take 2EM [2] instead of AES, since 2EM is more friendly to Barefoot Tofino and can be completed without resubmitting the packet, while the AES needs to resubmit the packet. As for the NDN [41] packet, we take the 32-bit content name for the packet forwarding with $\mathcal{F}_{\text{FIB}}$ and $\mathcal{F}_{\text{PIT}}$.

4.2 Evaluation

Packet processing time. For the IP, NDN, OPT, and NDN +OPT packets, we test their processing time with 128-byte, 768-byte, and 1500-byte packet sizes. The forwarding times of IPv4 and IPv6 packets are used as baselines. We carried out 1000 forwarding tests for each size of the packet and the evaluation results are shown in Figure 2. The results show that the processing times of DIP packets are close to the baseline. The OPT and NDN+OPT packets take more processing time since the MAC operations are expensive.

Packet header size overhead. The packet header size overhead is shown in Table 2. The basic DIP header occupies 6 bytes and the remaining parts (*i.e.,* FN definitions and FN

[2] In our prototype the router has no cached data, so there is no matching content store. For the forwarding devices that support caching, the FIB matching module can be slightly modified to first match the local content store and then match the FIB.

[3] The session ID is a flow tag and is generated during the key negotiation process in OPT.

Table 2: The packet header size overhead

Network function	Total header size (Byte)
IPv6 forwarding	40
IPv4 forwarding	20
DIP-128 forwarding	50
DIP-32 forwarding	26
NDN forwarding	16
OPT forwarding	98
NDN+OPT forwarding	108

locations) vary in different L3 protocols. Therefore, DIP has slightly more packet header size overhead than IP forwarding. The packet headers of OPT and NDN+OPT are significantly larger than the IP packet header due to the extra source and path authentication tags.

5 DISCUSSION

Addressable Market. The long-cherished desire of our community is making the Internet support diverse network functions and protocols. To this end, the community considered to build new clean-slate Internet architectures (*e.g.,* [12, 41]), and then switched gears to gradually, not surely, improve the Internet via incrementally deployable protocols like Trotsky [22]. DIP is the successor to these efforts: DIP proposes a new primitive, name FN, as the basic building block to enable customizable network functions, as demonstrated in § 3. We do not claim that FN is the only option. Yet, it follows the recent research trend of Runtime Programmable Networks [10, 40].

Opportunities with DIP. DIP is beneficial for both network providers and end users. On the one hand, users are allowed to construct their desired network protocols. On the other hand, the network providers can now support new services by only upgrading FNs, instead of replacing the underlying hardware. Further, the flexibility of FNs also enables the providers to offer differentiated services to different users, according to their subscription models. Other opportunities of DIP include implementing stateless guaranteed services [29, 30] and efficient network telemetry [14, 33].

Hardware Limitations. The commodity switch in our lab (*i.e.,* Barefoot Tofino programmable switch S9180-32X) has some limitations and we made some compromises in implementing the DIP prototype, as described in § 4.1. Considering that runtime programmable device is currently a hot research topic in our community and some exciting results have been achieved [9, 39], we believe that DIP can also embrace the advances in the foreseeable future.

6 RELATED WORK

Research on network architecture is protracted. There are many notable achievements [4, 5, 12, 24, 25, 35] and the Trotsky [22] is one of the currently hotly discussed schemes. The main idea of Trotsky is to add a new protocol layer (*i.e.,* L3.5) to shield the various L3 protocols, as the IP protocol shields different link-layer technologies, and decide the Internet's service model at L3.5. Our work is orthogonal to Trotsky and focuses on virtually integrating various L3 protocols.

Active network [34] is an advanced network vision, which allows users to inject customized programs into the network nodes and replaces packet with "capsule". Its research has laid the foundation for many important developments in network programmability [1, 7, 26, 37]. However, the extreme flexibility of active network makes it difficult to address issues such as resource isolation and security. DIP allows the host to customize the packet processing with FNs, which provides constrained and appropriate flexibility.

Programmable Network Infrastructure [3, 27] is a hot research topic in our community and the runtime programmability has been studied in several contexts. The architecture community has extensively explored the capability of live, partial reconfiguration of FPGAs [36]. Recently, switch vendors are increasingly exposing runtime programmability in their ASICs [40] and our community has also made some academic achievements [9, 10, 39] to programming the network to add, remove, and modify functions at runtime.

Network Function Virtualization [13, 18, 21, 32] is a long-researched topic in our community, which is a network architecture for virtualizing network functions (VNFs) on commodity general-purpose hardware. DIP takes the FN as the building block and some operation modules without delay requirements can be implemented with VNFs.

7 CONCLUSION

DIP leverages the development of network programmable devices to provide a new design space for the use and customization of network functions. The preliminary implementation and evaluation have shown that DIP can realize various L3 protocols with good performance. We are working on improving the DIP scheme and implementing more L3 protocols with DIP in an extended paper.

This work does not raise any ethical issues.

ACKNOWLEDGMENTS

This work was in part supported by the China National Funds for Distinguished Young Scientists under Grant 61825204; in part by NSFC Project under Grant 61932016 and 62132011; in part by Beijing Outstanding Young Scientist Program under Grant BJJWZYJH01201910003011.

REFERENCES

[1] D Scott Alexander, William A Arbaugh, Michael W Hicks, Pankaj Kakkar, Angelos D Keromytis, Jonathan T Moore, Carl A Gunter, Scott M Nettles, and Jonathan M Smith. 1998. The SwitchWare active network architecture. *IEEE network* 12, 3 (1998), 29–36.

[2] Andrey Bogdanov, Lars R Knudsen, Gregor Leander, François-Xavier Standaert, John Steinberger, and Elmar Tischhauser. 2012. Key-alternating ciphers in a provable setting: encryption using a small number of public permutations. In *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 45–62.

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

[4] Dave Clark, Bill Lehr, Steve Bauer, Peyman Faratin, Rahul Sami, and John Wroclawski. 2006. Overlay Networks and the Future of the Internet. *Communications and Strategies* 63 (2006), 109.

[5] Jon Crowcroft, Steven Hand, Richard Mortier, Timothy Roscoe, and Andrew Warfield. 2003. Plutarch: an argument for network pluralism. *ACM SIGCOMM Computer Communication Review* 33, 4 (2003), 258–266.

[6] Jason A Donenfeld. 2017. WireGuard: Next Generation Kernel Network Tunnel.. In *NDSS*. 1–12.

[7] Nick Feamster, Jennifer Rexford, and Ellen Zegura. 2014. The road to SDN: an intellectual history of programmable networks. *ACM SIGCOMM Computer Communication Review* 44, 2 (2014), 87–98.

[8] Markus Feilner. 2006. *OpenVPN: Building and integrating virtual private networks*. Packt Publishing Ltd.

[9] Yong Feng, Zhikang Chen, Haoyu Song, Wenquan Xu, Jiahao Li, Zijian Zhang, Tong Yun, Ying Wan, and Bin Liu. 2022. Enabling In-situ Programmability in Network Data Plane: From Architecture to Language. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 635–649.

[10] Yong Feng, Haoyu Song, Jiahao Li, Zhikang Chen, Wenquan Xu, and Bin Liu. 2021. In-situ programmable switching using rp4: Towards runtime data plane programmability. In *Proceedings of the Twentieth ACM Workshop on Hot Topics in Networks*. 69–76.

[11] Songtao Fu, Ke Xu, Qi Li, Xiaoliang Wang, Su Yao, Yangfei Guo, and Xinle Du. 2021. MASK: practical source and path verification based on Multi-AS-Key. In *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*. IEEE, 1–10.

[12] Dongsu Han, Ashok Anand, Fahad Dogar, Boyan Li, Hyeontaek Lim, Michel Machado, Arvind Mukundan, Wenfei Wu, Aditya Akella, David G Andersen, et al. 2012. {XIA}: Efficient support for evolvable internetworking. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. 309–322.

[13] Georgios P Katsikas, Tom Barbette, Dejan Kostic, Rebecca Steinert, and Gerald Q Maguire Jr. 2018. Metron:{NFV} service chains at the true speed of the underlying hardware. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. 171–186.

[14] Changhoon Kim, Anirudh Sivaraman, Naga Katta, Antonin Bas, Advait Dixit, and Lawrence J Wobker. 2015. In-band network telemetry via programmable dataplanes. In *ACM SIGCOMM*, Vol. 15.

[15] Dohyung Kim, Jun Bi, Athanasios V Vasilakos, and Ikjun Yeom. 2017. Security of cached content in NDN. *IEEE Transactions on Information Forensics and Security* 12, 12 (2017), 2933–2944.

[16] Tiffany Hyun-Jin Kim, Cristina Basescu, Limin Jia, Soo Bum Lee, Yih-Chun Hu, and Adrian Perrig. 2014. Lightweight source authentication and path validation. In *Proceedings of the 2014 ACM Conference on SIGCOMM*. 271–282.

[17] Markus Legner, Tobias Klenze, Marc Wyss, Christoph Sprenger, and Adrian Perrig. 2020. {EPIC}: Every Packet Is Checked in the Data Plane of a {Path-Aware} Internet. In *29th USENIX Security Symposium (USENIX Security 20)*. 541–558.

[18] Bojie Li, Kun Tan, Layong Luo, Yanqing Peng, Renqian Luo, Ningyi Xu, Yongqiang Xiong, Peng Cheng, and Enhong Chen. 2016. Clicknp: Highly flexible and high performance network processing with reconfigurable hardware. In *Proceedings of the 2016 ACM SIGCOMM Conference*. 1–14.

[19] Xin Liu, Xiaowei Yang, and Yong Xia. 2010. Netfence: preventing internet denial of service from inside out. *ACM SIGCOMM Computer Communication Review* 40, 4 (2010), 255–266.

[20] Zhuotao Liu, Hao Jin, Yih-Chun Hu, and Michael Bailey. 2016. MiddlePolice: Toward enforcing destination-defined policies in the middle of the Internet. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 1268–1279.

[21] Antonis Manousis, Rahul Anand Sharma, Vyas Sekar, and Justine Sherry. 2020. Contention-aware performance prediction for virtualized network functions. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. 270–282.

[22] James McCauley, Yotam Harchol, Aurojit Panda, Barath Raghavan, and Scott Shenker. 2019. Enabling a permanent revolution in internet architecture. In *Proceedings of the ACM Special Interest Group on Data Communication*. 1–14.

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

[24] David Naylor, Matthew K Mukerjee, Patrick Agyapong, Robert Grandl, Ruogu Kang, Michel Machado, Stephanie Brown, Cody Doucette, Hsu-Chun Hsiao, Dongsu Han, et al. 2014. XIA: Architecting a more trustworthy and evolvable Internet. *ACM SIGCOMM Computer Communication Review* 44, 3 (2014), 50–57.

[25] Norbert Niebert, Stephan Baucke, Ibtissam El-Khayat, Martin Johnsson, Borje Ohlman, Henrik Abramowicz, Klaus Wuenstel, Hagen Woesner, Jurgen Quittek, and Luis M Correia. 2008. The way 4ward to the creation of a future internet. In *2008 IEEE 19th International Symposium on Personal, Indoor and Mobile Radio Communications*. IEEE, 1–5.

[26] Beverly Schwartz, Alden W Jackson, W Timothy Strayer, Wenyi Zhou, R Dennis Rockwell, and Craig Partridge. 1999. Smart packets for active networks. In *1999 IEEE Second Conference on Open Architectures and Network Programming. Proceedings. OPENARCH'99 (Cat. No. 99EX252)*. IEEE, 90–97.

[27] Haoyu Song. 2013. Protocol-oblivious forwarding: Unleash the power of SDN through a future-proof forwarding plane. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. 127–132.

[28] Ion Stoica, Daniel Adkins, Shelley Zhuang, Scott Shenker, and Sonesh Surana. 2002. Internet indirection infrastructure. In *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. 73–86.

[29] Ion Stoica and Hui Zhang. 1999. Providing guaranteed services without per flow management. *ACM SIGCOMM Computer Communication Review* 29, 4 (1999), 81–94.

[30] Ion Stoica, Hui Zhang, S Shenker, R Yavatkar, D Stephens, A Malis, Y Bernet, Z Wang, F Baker, J Wroclawski, et al. 1999. Per hop behaviors based on dynamic packet states. *draft-stoica-diffserv-dps-00. txt* (1999).

[31] Nik Sultana, John Sonchack, Hans Giesen, Isaac Pedisich, Zhaoyang Han, Nishanth Shyamkumar, Shivani Burad, André DeHon, and

Boon Thau Loo. 2021. Flightplan: Dataplane disaggregation and placement for p4 programs. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. 571–592.

[32] Chen Sun, Jun Bi, Zhilong Zheng, Heng Yu, and Hongxin Hu. 2017. NFP: Enabling network function parallelism in NFV. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. 43–56.

[33] Lizhuang Tan, Wei Su, Wei Zhang, Jianhui Lv, Zhenyi Zhang, Jingying Miao, Xiaoxi Liu, and Na Li. 2021. In-band network telemetry: A survey. *Computer Networks* 186 (2021), 107763.

[34] David L Tennenhouse and David J Wetherall. 2002. Towards an active network architecture. In *Proceedings DARPA Active Networks Conference and Exposition*. IEEE, 2–15.

[35] Christian Tschudin and Richard Gold. 2003. Network pointers. *ACM SIGCOMM Computer Communication Review* 33, 1 (2003), 23–28.

[36] Kizheppatt Vipin and Suhaib A Fahmy. 2018. FPGA dynamic and partial reconfiguration: A survey of architectures, methods, and applications. *ACM Computing Surveys (CSUR)* 51, 4 (2018), 1–39.

[37] David Wetherall and David Tennenhouse. 2019. Retrospective on" towards an active network architecture". *ACM SIGCOMM Computer Communication Review* 49, 5 (2019), 86–89.

[38] Bo Wu, Ke Xu, Qi Li, Zhuotao Liu, Yih-Chun Hu, Martin J Reed, Meng Shen, and Fan Yang. 2018. Enabling efficient source and path verification via probabilistic packet marking. In *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*. IEEE, 1–10.

[39] Jiarong Xing, Kuo-Feng Hsu, Matty Kadosh, Alan Lo, Yonatan Piasetzky, Arvind Krishnamurthy, and Ang Chen. 2022. Runtime programmable switches. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 651–665.

[40] Jiarong Xing, Yiming Qiu, Kuo-Feng Hsu, Hongyi Liu, Matty Kadosh, Alan Lo, Aditya Akella, Thomas Anderson, Arvind Krishnamurthy, TS Eugene Ng, et al. 2021. A Vision for Runtime Programmable Networks. In *Proceedings of the Twentieth ACM Workshop on Hot Topics in Networks*. 91–98.

[41] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, KC Claffy, Patrick Crowley, Christos Papadopoulos, Lan Wang, and Beichuan Zhang. 2014. Named data networking. *ACM SIGCOMM Computer Communication Review* 44, 3 (2014), 66–73.

[42] Xin Zhang, Hsu-Chun Hsiao, Geoffrey Hasker, Haowen Chan, Adrian Perrig, and David G Andersen. 2011. SCION: Scalability, control, and isolation on next-generation networks. In *2011 IEEE Symposium on Security and Privacy*. IEEE, 212–227.