OCEAN: Optional Capability-Based En Route Acknowledgement in Network Layer

Su Yao, *Senior Member, IEEE*, Songtao Fu, Qi Li, *Senior Member, IEEE*, Zhuotao Liu, Yi Cai, Yinchao Zhang, and Ke Xu, *Fellow, IEEE*

***Abstract*—High security and low latency are important in mission-critical data transmission, such as the end-to-end transmission in Industrial IoT (IIoT). However, existing schemes often struggle to simultaneously meet these demanding requirements due to hardware limitations and the lack of a packet lossless forwarding protocol in the network layer data plane. To address this challenge, we propose OCEAN (Optional Capability-based En route Acknowledgement in Network layer). OCEAN includes (1) an in-network caching hardware, which is a programmable Application Specific Integrated Circuit (ASIC) integrated with a Field Programmable Gate Array (FPGA), and (2) a packet lossless forwarding protocol in the network layer data plane. In OCEAN, each packet was generated by an authorized end device, while each en route node verifies the packet, and caches it until receiving the acknowledgment from the next en route node. It incurs negligible latency to packet forwarding when there is no packet loss while retransmitting the packet at the en route node after a short timeout, which reduces the packet forwarding latency. Besides that, the per-packet verification guarantees that the adversary could not subvert the forwarding protocol. Our simulation in the BMv2 environment confirms its functionality, and the hardware implementation demonstrates that it can process packets at line rate with a total processing latency ranging from 2519 ns to 6160 ns, which is negligible in end-to-end transmission.**

***Index Terms*—Industrial IoT, capability-based networking, packet verification, packet lossless forwarding.**

Received 8 August 2024; revised 22 July 2025; accepted 9 September 2025. Date of publication 24 September 2025; date of current version 14 January 2026. This work was supported by the National Key Research and Development Program of China under Grant 2022YFB3102301, in part by the National Natural Science Foundation of China under 62472240, under Grant 62394322, under Grant U22B2031, under Grant 62132011, in part by the Beijing National Research Center for Information Science and Technology under Grant BNR2025RC01010, in part by the National Science Foundation for Distinguished Young Scholars of China under Grant 62425201, and in part by the Taishan Scholar Foundation of Shandong Province under Grant tstp20250724. *(Corresponding authors: Su Yao; Ke Xu.)*

Su Yao is with the Beijing National Research Center for Information Science and Technology (BNRist), Tsinghua University, Beijing 100084, China (e-mail: yaosu@tsinghua.edu.cn).

Songtao Fu, Yi Cai, and Yinchao Zhang are with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: fust18@tsinghua.org.cn; afireswallow@gmail.com; caiyicdd@163.com).

Qi Li is with the Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing 100084, China, and also with the State Key Laboratory of Internet Architecture, Tsinghua University, Beijing 100084, China (e-mail: qli01@tsinghua.edu.cn).

Zhuotao Liu is with the Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing 100084, China.

Ke Xu is with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China, and also with Zhongguancun Laboratory, Beijing 100194, China (e-mail: xuke@tsinghua.edu.cn).

Digital Object Identifier 10.1109/TDSC.2025.3613841

I. Introduction

As THE Internet is increasingly utilized for critical services, ensuring correct and uninterrupted operations becomes paramount. For example, with the rising popularity of the Industrial IoT (IIoT) [1], low security and high latency are not just an inconvenience, they are potentially life threatening.

However, the current state of the Internet does not meet the requirements for end-to-end transmission with high security and low latency [2]. Achieving high security involves implementing mechanisms that can verify packets at routers to defend against source address spoofing and other attacks. These mechanisms require powerful source and destination devices capable of handling complex operations such as stateful operations and multiple Message Authentication Code (MAC) operations [3], [4], [5], [6], [7], [8]. However, this may not be feasible in mission-critical applications where source and destination users in these scenarios (e.g., IIoT) may have limited power, computation, or storage resources [9].

On the other hand, to achieve low latency and reliable transmission, Peter et al. propose ARROW (Advertised Reliable Routing Over Waypoints), which allows end users or enterprises to establish reliable end-to-end paths in the control plane [10]. This improves the Internet's resilience in the control plane. However, it does not address how to ensure packet forwarding with both security guarantees and bounded latency in the data plane.

The Time-Sensitive Networking (TSN) task group is dedicated to designing a mechanism that enables "guaranteed packet transport with bounded latency, low packet delay variation and low packet loss"[1] over IEEE 802 networks (i.e., Ethernet). We call a coordinated TSN network or network segment a TSN domain. It is important to note that TSN cannot guarantee transmission in an inter-domain context. DetNet [11] devotes to guarantee the packet was delivered on the network nodes (e.g., router) as schemed in the inter-domain context. However, it has two drawbacks. Firstly, each en route node[2] does not verify the packet, which creates the possibility for an adversary to launch DoS attacks on a specific node. Secondly, DetNet cannot guarantee immediate retransmission of packets by each

[1]https://1.ieee802.org/tsn

[2]We name the network node (e.g., router) the en route node.

1545-5971 © 2025 IEEE. All rights reserved, including rights for text and data mining, and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 02:57:41 UTC from IEEE Xplore. Restrictions apply.

TABLE I
COMPARISON OF MECHANISMS FOR HIGH-VALUE TRAFFIC TRANSMISSION

Mechanism	Control/Data	WAN	Efficiency	Security
ARROW	Control Plane	Yes	N/A	N/A
TSN	Data Plane	No	High	Limited
DetNet	Data Plane	Yes	Medium	Limited

en route node. In cases where packets are lost en route,[3] the source must retransmit the lost packet after an end-to-end timeout. This approach cannot meet the delivery requirements for mission-critical applications. An example of such a scenario is in space edge computing for advanced 6G services, Re-sending lost packets from on-path routers located in space in close proximity to the receiver is necessary [12].

Table I provides a comparative analysis of existing mechanisms for high-value traffic transmission in network layers, focusing on their operation in the control plane or data plane, support for wide-area network (WAN), efficiency, and security. This comparison highlights the limitations of current approaches and underscores the need for a new mechanism that can address these shortcomings. While TSN provides high efficiency for time-sensitive flows within a domain, its security is limited because it lacks encryption/authentication in base standards. Additional protocols like MACsec (IEEE 802.1AE) must be explicitly deployed to address this gap. DetNet offers WAN-capable deterministic networking but its security also relies on external protocols (e.g., IPsec). ARROW operates exclusively in the control plane and thus fails to ensure efficiency and security in the data plane.

Therefore, a mechanism ensuring high security and low latency is critical for mission-critical applications. Note that end-to-end delivery may occur within one or across multiple Autonomous Systems (ASes); we exclude same-network scenarios (e.g., Ethernet) where TSN guarantees performance.

This paper designs a practical mechanism for low-latency, high-security transmission with lossless forwarding and en route verification. Lossless forwarding ensures intermediate nodes cache packets until receiving an acknowledgment (ACK) from the next node, retransmitting on timeout—neighbor-level timeouts reduce latency compared to end-to-end ones. En route verification filters invalid packets via data-plane checks at each node. Caching and verification are key to security and latency goals. However, caching all packets and performing data-plane verification at en route nodes (e.g., AS border routers) is challenging due to programmable ASICs' limited memory and computation [13].

Fortunately, we observe that the mission-critical applications on the Internet has two characteristics: ① The mission-critical traffic is just the low-volume, high-value traffic, which is quite low compared to normal traffic on the Internet [10]. Crucially, the study demonstrates that even with the high-data-volume nature of autonomous driving (e.g., 8 Gbps from sensors), strategic bandwidth allocation can enable cloud offloading of high-value services. For example, the EfficientDet 3 (ED3) model requires a bandwidth of 19.0 Mbps to meet latency requirements when executed in the cloud [14]. ② The packet of mission-critical applications (e.g., the packet in IIoT) only traverses specific nodes, unlike general traffic. Thus, re-architecting the entire Internet is unnecessary. Instead, we propose OCEAN, which extends programmable ASICs with FPGAs [15] to enhance storage/computation, and implements a data-plane protocol for verification and lossless forwarding. OCEAN is an optional capability-based mechanism that has two characteristics: ① **High security.** En route nodes verify packets (first node checks source; others check the prior node). ② **Low latency.** Lossless forwarding via caching and retransmission on neighbor-based timeout. This paper's contributions are:

1) *Adversary Model and Design Goal:* We formally define the adversary model for mission-critical Internet applications in an inter-domain context, detailing the capabilities and limitations of potential attackers in IIoT environments. Additionally, we present OCEAN's design goals, which focus on establishing a secure and efficient transmission framework (Section II).
2) *OCEAN Hardware and Protocol Design:* We introduce the OCEAN hardware architecture, which integrates Intel Tofino2 and FPGA to support in-network caching. We also detail the packet lossless forwarding protocol implemented in the network layer, designed to ensure high security and low latency (Section III).
3) *Security and Performance Analysis:* We conduct a thorough analysis of OCEAN's security and performance properties. This analysis demonstrates its resilience against various attack vectors. Furthermore, we explain how OCEAN mitigates DoS attacks in the control plane using a neighbor-based counter, making it well-suited for mission-critical IIoT applications (Section IV).
4) *Simulation and Hardware Evaluation:* We validate OCEAN's functionality and performance under partial deployment of OCEAN-supported nodes in the BMv2 environment. We further evaluate its performance on the OCEAN hardware, demonstrating that it can process packets at line rate with negligible latency compared to end-to-end transmission. This evaluation confirms OCEAN's ability to meet the stringent requirements of IIoT deployments (Sections V and VI).

II. ADVERSARY MODEL AND DESIGN GOALS

A. Adversary Model

As IIoT is an important mission-critical application on the Internet, we design OCEAN based on the requirements of IIoT. We consider the end devices of IIoT in two remote networks connected to the Internet. The links between the two remote networks may be inter-domain (e.g., which has several transit ASes) [16], [17] or intra-domain. In an intra-domain context, a network model in IIoT consists of these entities: (1) key distribution server (KDS), (2) end device, (3) en route node and link. These entities are all managed by a network administrator. While in an inter-domain context, besides the entities in IIoT,

[3]http://cisco.num.edu.mn/CCNA_R&S1/course/module6/6.1.2.3/6.1.2.3.html

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 02:57:41 UTC from IEEE Xplore. Restrictions apply.

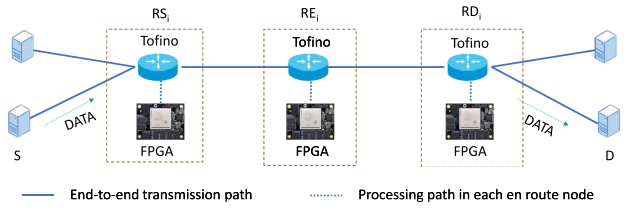


Fig. 1. End-to-end packet transmission in our design, the RE_i is not necessarily support OCEAN.

we assume that each transit AS on the Internet has a KDS [8] and several en route nodes (as well as links between these nodes).

We first describe the honest assumption. We assume the network administrator and the KDS in IIoT are honest and also assume that the KDS in the transit AS is also honest. This is a strong assumption on the Internet, but the compromised KDS in the transit AS only affects the traffic it received or forwarded, since each KDS only negotiates the key with the KDS of its neighbor ASes.

Based on the assumption, we consider a network-based adversary which may have compromised a subset of end devices, en route nodes, and links between the nodes. The adversary could utilize any method (including physical access) to have full control over the compromised entities. Moreover, the adversary can communicate out-of-band with each other.

The adversary aims to expand the reach of its attack, considering OCEAN's objective to establish a trusted communication channel for end devices. We focus on attacks that aim to degrade or undermine the OCEAN system. The potential strategies employed by the adversary include: (1) Sending spoofed packets to initiate DoS attacks, such as impersonating end devices and overwhelming the storage capacity of en route nodes with spoofed packets. (2) Subverting the forwarding protocol by either dropping or replaying the packet.

B. Design Goals

As shown in Fig. 1, we consider the transmission path between two end devices (S and D) to have one or more en route nodes. RS_i represents the en route nodes (e.g., one or more routers) in source AS, RE_i represents the en route nodes in transit ASes, and RD_i represents the en route nodes in destination AS (in intra-domain context, the source AS is the destination AS). To guarantee the caching performance, we utilize FPGA to extend the storage space. RS_i and RD_i support OCEAN since they are usually administrate by the administrator of a network (e.g., IIoT), and not all the $RE_i s$ support OCEAN. Each en route node supported OCEAN considers the legacy node transparent during the packet forwarding. In the extreme case, all the en route nodes between RS_i and RD_i may be legacy nodes. It is just like a tunnel between RS_i and RD_i. To guarantee the performance and robustness in end-to-end transmission, we present the design goals of OCEAN as follows:

High security: OCEAN aims to achieve a high level of security by assigning each packet to its own capability-based path. This approach minimizes both the exposure and access of the packet within the network. We have defined the following goals: ① Authentication: Preventing external malicious traffic from entering the network. Each en route node has the ability to authenticate each packet using the MACs in the packet header, effectively preventing the occurrence of spoofed packets. It is worth noting that, considering the high computational resources required for asymmetric encryption, we utilize symmetric encryption for packet verification. ② Isolation: Preventing internal malicious traffic from subverting the network. Each en route node can negotiate forwarding policies (such as the number of packets during a specific interval) with other nodes. If any node violates the policy (e.g., overwhelming the next node by sending huge number of packets), it is reported to the administrator in the control plane. It prevents compromised elements from subverting the network.

Low latency: The packet processing incurs little additional latency compared with IP packet forwarding. That means the en route node processes the packet at line rate. Furthermore, each en route node can cache and retransmit the lost packet promptly, thereby reducing the end-to-end transmission latency in the case of retransmission.

Compatibility: OCEAN does not hurt legacy packet forwarding, while the OCEAN packet has relatively high priority.

III. Design

A. Overview

At a high level, OCEAN is an optional capability-based mechanism to support high security and low latency for high-value traffic. It includes a hardware that supports in-network caching and a packet lossless forwarding protocol that supports each en route node to retransmit the verified packet as early as possible.

Based on the OCEAN hardware, we design a packet lossless forwarding protocol to achieve high security and low latency transmission, which includes a packet verification mechanism and hop-by-hop acknowledgment mechanism.

To support high security, we utilize **Double Verification** mechanism to only verify the OCEAN packet. The first verification was achieved at programmable ASIC, it verifies each packet with a lightweight verification at line rate. It then forwards the packet to the FPGA after successful verification, or else, it drops the packet to prevent the adversary from overwhelming the computation and storage resources of the FPGA. The FPGA utilizes another MAC operation with a strong security guarantee (e.g., AES) to achieve the second verification. The bandwidth between the ASIC and FPGA is 400 Gbps, which is sufficient for the low-volume traffic of IIoT.

To support low latency, we design a hop-by-hop acknowledgment mechanism in the network layer. Particularly, each en route node caches the packet until receiving the acknowledgment from the next en route node, or retransmits the packet after a short timeout.

B. OCEAN Hardware

In-network caching for stateful processing is a complex task to accomplish on the Internet. According to the research paper

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 02:57:41 UTC from IEEE Xplore. Restrictions apply.

[13], it is highlighted that a chip utilizing 7nm technology with an area of 826 mm^2 can only store a maximum of 5GB of SRAM. In practical scenarios, a 16 nm ASIC switch typically contains 10–100 MB SRAM on a 6.5 Tbps ASIC. Therefore, our hardware design integrates a programmable ASIC and FPGA (with a cache) in the data plane. This integration enables the verification and forwarding of low-volume, high-value traffic at line rate. Additionally, in the control plane, our OCEAN hardware employs a CPU to control the ASIC and FPGA through the management interface.

C. Packet Lossless Forwarding Protocol

We first detail the dynamic key for the **Double Verification** and the additional packet header, then describe the packet verification and hop-by-hop acknowledgment mechanism.

1) Dynamic Key: Considering that the end device in IIoT (or other mission-critical networks) has constrained computation and storage resources, and these traffic generally traverse specific domains, we design a neighbor-based dynamic key mechanism. The first en route node ($ingress$ node, e.g., a gateway) verifies the source (end device) with a specific verification mechanism in intra-domain context [8]. Then each en route node uses the dynamic key as its capability tokens to generate MAC for each packet [9]. The most crucial aspect of OCEAN is the dynamic key between two neighbor en route nodes [18].

We first detail the dynamic key for the first verification. As the ASIC has limited computation resources, we utilize a lightweight MAC operation to achieve the verification, thus needing to change the dynamic key frequently (e.g., several minutes). We do not utilize the on-path routers recreates the dynamic key on the fly, as it incurs the heavy computation overhead to calculate the dynamic key with a strong security guarantee [19]. For instance, to achieve the AES operation, the recirculation bandwidth and the connection between the data plane and dedicated extern encryption hardware could be the bottleneck of the forwarding [20]. Fortunately, presetting the neighbor-based dynamic keys incurs little storage overhead of each router. Therefore, we utilize a time-slot-based dynamic key deriving mechanism, ensuring the confidentiality of the dynamic key with no computation overhead and constant storage overhead. We assume that a long-term root key (LK_{ij}, change once a day or more) could be confidentially shared between the entity i and entity j (LK_{ij} is the same as LK_{ji}). Then the dynamic key DK_{ij} between entity i and j are derived by each entity at the beginning of each time slot, which calculates as:

$$DK_{ij} = MAC_{LK_{ij}}(id_i||id_j||TS) \tag{1}$$

In which '||' represents the concatenation operation, id_i and id_j are the identifiers (e.g., IP address) of entity i and j (assume that $id_i < id_j$ to ensure that the two entities could derive the same DK_{ij}), TS is the number of time slot. We assume that neighbor nodes, including routers and end devices, maintain coarse-grained time synchronization at the minute level. Each en route node calculates the dynamic key for M (e.g., 2) slots using the LK_{ij} in the control plane with its neighbor nodes, including end devices. The end devices obtain the DK_{ij} for M (e.g., 2) slots from the Key Distribution System (KDS). Subsequently, they can utilize a single bit in the $flag$ field in Fig. 3 to instruct each router to switch the dynamic key.

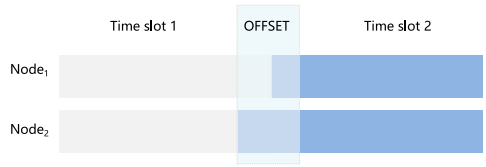


Fig. 2. The OFFSET of the time slot.

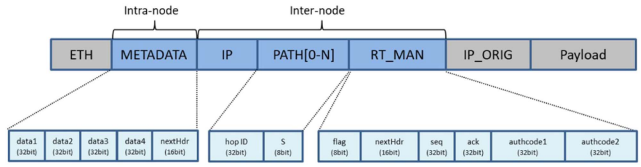


Fig. 3. OCEAN packet header, the METADATA is only used between Tofino and FPGA within a single node.

In practical implementation, it is feasible to customize the timing of key switching for two neighboring nodes. For instance, if these nodes switch the dynamic key every 2-minute time slot, they can predefine the dynamic keys for two consecutive time slots in the control plane. Each node uses the value of the first bit of the $flag$ 0 to indicate the first time slot, allowing the subsequent en route node to use the corresponding relative dynamic key for verification. When the $flag$ value switches to 1, the next node can use the dynamic key assigned to the second time slot for verification. To accommodate slight synchronization discrepancies, we introduce an OFFSET. As shown in Fig. 2, during this OFFSET period, each node retains the dynamic keys for two time slots. After the OFFSET, the node presets the dynamic key for the third time slot and clears the dynamic key for the first time slot. The subsequent $flag$ value of 0 indicates the third time slot. With this design, maintaining time synchronization at the minute level is sufficient for the two neighboring nodes.

The dynamic key (key_{ij}) for the second verification changes infrequently since the FPGA could utilize a MAC operation with a strong security guarantee. It could utilize the (1) to generate the key_{ij} except that it has a relatively larger timeslot and different long-term root key.

2) Additional Packet Header in En Route Node: The additional (OCEAN) packet header is shown in Fig. 3. The $METADATA$ (Intra-node) contains information field ($data1, data2, data3, data4$), which is used to share operational data between ASIC and FPGA. $nextHdr$ indicates the type of the following field. Each ASIC adds $METADATA$ when receiving a new packet and removes it before sending it to the next node. The Inter-node packet header includes the IP, $PATH[0-N]$, and RT_MAN. The IP is the new IP header, in which the destination IP address is the next en route node. The $PATH[0-N]$ is a stack that records the IP address from the current node to the destination, with $PATH[i]$ containing $hopID$ and s, where $hopID$ denotes the IP address of a specific

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 02:57:41 UTC from IEEE Xplore. Restrictions apply.

Algorithm 1: Double Verification at En Route Node.

input : pkt, DK, key

1 **while** receive a packet **do**
2 **if** *is ingress node* **then**
3 *verify the pkt with source address*
4 **else**
5 *verify the pkt with* ***authcode1***
6 *update the METADATA and send to FPGA*
7 **if** *is not ingress node* **then**
8 *verify the pkt with* ***authcode2***
9 **if** *is not egress node* **then**
10 **authcode2** $\leftarrow AMAC_{FP_{ij}}$
11 *update the METADATA and send to ASIC*
12 **if** *is not egress node* **then**
13 **authcode1** $\leftarrow AMAC_{AS_{ij}}$

en route node, and s signifies a bottom-of-the-stack identifier. The RT_MAN header records the packet status, including packet type, packet identification, and packet verification code. The $flag$ field records the packet type, each node will perform different operations on the packet based on different types, we also utilize 1 bit in flag to indicate the switching of the dynamic key. The seq field and ack field are the sequence numbers to identify the $DATA$ and ACK packet, respectively. The $authcode1$ (for the first verification) and $authcode2$ (for the second verification) fields are used for packet verification. IP_ORIG is the original IP packet header. The first en route node adds the OCEAN packet header, and the last en route node removes it to guarantee that the destination could receive an IP packet.

3) Packet Verification and Hop-by-Hop Acknowledgment: As shown in Algorithm 1, each en route node achieves the double verification as follows: Each node keeps the DK_{ij} and key_{ij} in the control plane. When receiving an OCEAN packet (line 1), each en route node needs to process the packet in three stages: ASIC-in, FPGA-processing, and ASIC-out. In the ASIC-in stage, the ingress node first verifies the $DATA$ packet with an intra-domain source address verification mechanism, e.g., in SAVI, each ingress node verifies the packet with the source address, which is a lightweight verification mechanism [21], [22] (line 3), and adds the OCEAN packet header.

For the other nodes, in the ASIC-in stage, the ASIC first verifies the packet with **authcode1** (line 5).

Then each en route node queries the forwarding table, updates the IP header, and adds the $METADATA$ header. After these operations, the ASIC forwards the packet to FPGA (line 6).

In the FPGA-processing stage, the FPGA caches and processes the packet based on its state. If the en route node is not the ingress node, it verifies the packet using the MAC calculated by the previous en route node (line 8). Then, each node, except the egress node, updates the MAC for the subsequent en route node (line 10). Then it forwards the packet to ASIC (line 11).

During the ASIC-out stage, all nodes, except the egress node, must update the authentication code in the header (line 13). Following this, each node removes the first element in the $PATH$, eliminates the $METADATA$ header, and forwards the packet to the next node. The calculations for the double verification are as follows:

First verification on ASIC: For each node except the egress node, in the ASIC-out stage, the ASIC generates the packet's MAC based on (2). The **nextHopID** is the second element in the $PATH$ and the **Seq** is the **seq** in RT_MAN header.

$$AMAC_{AS_{ij}} = MAC_{DK_{ij}}(nextHopID||Seq) \quad (2)$$

After the next en route node receives the OCEAN packet, in the ASIC-in stage, the ASIC calculates the packet's MAC based on (3). The **currHopID** is the first element in the *PATH* and the **Seq'** is recorded in the **seq** field in RT_MAN. The results of the $VMAC_{AS_{ij}}$ will be used to compare with the **authcode1** field in the RT_MAN. Each node could count the packets that failed the first verification, and report to the network operator (e.g., the administrator of the IIoT) in the control plane.

$$VMAC_{AS_{ij}} = MAC_{DK_{ij}}(currHopID||Seq') \quad (3)$$

Second verification on FPGA: To improve security, as the ASIC cannot verify the packet payload, we nest the MAC in **authcode2** to verify the validity and integrity of the packet. As shown in (4) and (5), the algorithm is similar to the operation on ASIC. The difference is that the input of the MAC operation on the FPGA adds the hash of the packet payload (HA in (4) and (5)), and the MAC operation on FPGA could have more security guarantees (e.g., AES). From this perspective, the key_{ij} does not need to be changed frequently as DK_{ij}.

$$AMAC_{FP_{ij}} = MAC_{key_{(i,j)}}(nextHopID||Seq||HA) \quad (4)$$

$$VMAC_{FP_{ij}} = MAC_{key_{(i,j)}}(currHopID||Seq'||HA) \quad (5)$$

Packet acknowledgement: OCEAN packets utilize the hop-by-hop ACK method. In this method, a sending node assigns a sequence number to each $DATA$ packet, which serves as a unique identifier for the packet. The sending node keeps track of the packet's status and retrieves the cached packet using this sequence number. The sequence number information is embedded in the **seq** field of RT_MAN in the packet header. To facilitate communication with multiple neighbor nodes, each node maintains a counter for assigning sequence numbers to each peer node. This counter acts as a form of proof to safeguard against potential protocol subversion by faulty or compromised entities. Notably, each node shares its counters with the network operator, enabling the operator to identify any faulty or compromised entities within the network.

Packet acknowledgement refers to the process where the receiving node responds to the sending node by sending an acknowledgement packet (ACK). This acknowledgement indicates that the corresponding $DATA$ packet has been successfully received. To construct the ACK packet, the receiving node uses the header of the $DATA$ packet. It swaps the values of the source address and destination address and sets the value of the **ack** in the RT_MAN to the value of **seq**.

Upon receiving the ACK packet, the sending node recognizes that the corresponding $DATA$ packet has been received successfully. It locates the relevant entry in the *status table* based

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 02:57:41 UTC from IEEE Xplore. Restrictions apply.

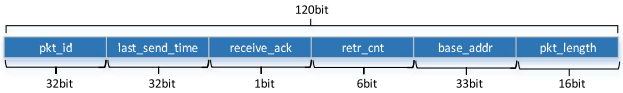


Fig. 4. The data structure of the table entry.

on the **ack** information and updates the status of the $DATA$ packet recorded in that entry.

OCEAN sets a *status table* on the data plane to maintain the status of $DATA$ packets. The *status table* is implemented in the SRAM of the FPGA. The table entry's data structure includes the following fields: pkt_id (32-bit), $last_send_time$ (32-bit), $receive_ack$ (1-bit), $retr_cnt$ (3-bit), $base_addr$ (33-bit), pkt_length (16-bit). These fields are illustrated in Fig. 4. pkt_id represents the unique identifier of the packet, which is derived from the hash of $nextHopID||Seq$. $last_send_time$ indicates the most recent time the packet was sent. $receive_ack$ records whether the corresponding ACK packet for the $DATA$ packet has been received (0 denotes not received, while 1 denotes received). $retr_cnt$ keeps track of the number of times the $DATA$ packet has been retransmitted. $base_addr$ stores the starting address of the packet in DDR4 memory, and pkt_length indicates the length of the packet.

Before transmitting a $DATA$ packet to the next node, the sending node must first store the packet in DDR4 memory and then log the packet's status in the *status table*. Once the receiving node receives the $DATA$ packet, it responds to the sending node with the corresponding ACK packet. Upon receiving an ACK packet, the sending node updates the status of the corresponding packet using the packet identification provided by the ACK packet and sets the $receive_ack$ bit of the corresponding entry to 1.

Packet caching in the data plane: OCEAN utilizes FPGA to implement a caching mechanism in its pipeline. This mechanism leverages the high capacity and fast I/O performance of DDR4 memory to store packets within the data plane. Before forwarding the $DATA$ packet to the next destination, the sending node saves the packet in DDR4 memory for caching purposes. If a packet timeout occurs, the node retrieves the packet from DDR4 memory using the corresponding entry in the *status table* and proceeds with retransmission.

We allocate a fixed and continuous memory area in DDR4 and design it as a ring buffer. The FPGA maintains a global write pointer that indicates the first address of the unallocated memory block or the inactive cached data. When a packet needs to be cached, the node requests the base address of a contiguous memory area using the packet length as a parameter. Then, the node initiates a write request to the DDR4 controller using this address, the number of memory blocks calculated based on the packet length, and the packet itself as parameters.

The caching module reads the memory base address from the memory write pointer register before caching the current DATA packet. After caching, the memory write pointer is updated as follows: $(p \leftarrow p + packet_length)$.

Timeout detection and retransmission: The OCEAN node utilizes the information stored in the *status table* to determine if a $DATA$ packet has timed out. It does this by periodically checking the table and extracting one entry at a time. By comparing the sending time (recorded as $last_send_time$ in the table entry) with the current time, the OCEAN node can infer if it exceeds a specific threshold. If the node has not received an ACK packet, the packet sending time exceeds the threshold, and the number of retransmissions is within a specific threshold, the OCEAN node will retransmit the packet. After the retransmission, it updates the corresponding entry in the *status table* by setting $last_send_time$ to the current time and incrementing $retr_cnt$ by 1.

Packet state and action: The packets require different actions based on their processing stage and packet types, such as $DATA$ packets and ACK packets. As a result, we have organized the packet states and categorized the corresponding operations for each state. Within each en route node, there are multiple packet states, namely *ingress* (gateway of the source), *inter*, and *egress* (gateway of the destination). Additionally, the forwarding direction within each node can be ASIC-to-FPGA, FPGA-to-ASIC, and ASIC-to-ASIC. As shown in Table II, each packet has 9 states, e.g., state ① $ingress_send_asic_to_FPGA$ indicates that a $DATA$ packet is currently at *ingress* node, and will be sent from ASIC to FPGA. The corresponding state $flag$, recorded in the $flag$ field of the RT_MAN structure, is 82. ② $send_fpga_to_asic$ indicates that a $DATA$ packet is currently at *inter* node and will be sent from ASIC to FPGA. Its state $flag$ is 10.

The ASIC and FPGA process different actions according to the packet states, which are shown in Table III. The **Action** column lists the atomic actions, while the **Packet State** column lists the set of actions that need to be processed in a specific state. For example, a packet in State ③ arrives at a node, the ASIC adds $METADATA$ to the packet header (C), verifies the packets (E), matches the forwarding table, updates IP headers (F), and sends the packet to the FPGA (I).

The packet state transition is illustrated in Table IV. The ASIC and FPGA execute the appropriate actions according to the packet state, subsequently updating the packet state and forwarding it to the next stage. For example, when a packet in state ③ undergoes actions C, E, F, I, if the packet is at the $egress$ node, the state transitions to ⑧; otherwise, the state transitions to ④.

IV. Analysis

In this section, we analyze the security and the performance of our design, to testify that OCEAN could achieve our design goals with sustainable overhead.

A. Security Analysis

As described in Section II-A, a network model in IIoT consists of an administrator and the following entities: (1) KDS, (2) end device, (3) en route node, and link. It is assumed that the administrator and the KDS are trustworthy, while the attacker can be either an outsider or an insider. With each en route node capable of verifying packets, the first en route node can filter out packets from outsiders. In other words, if the end device and en route nodes are trustworthy and the packet header contains

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 02:57:41 UTC from IEEE Xplore. Restrictions apply.

TABLE II
PACKET STATE

state	Ingress node	Egress node	ASIC-FPGA [a]	FPGA-ASIC	ASIC-ASIC	$DATA$	ACK	flag
① $ingress_send_asic_to_fpga$	✓[b]	-	✓	-	-	✓	-	82
② $send_fpga_to_asic$	-	-	-	✓	-	✓	-	10
③ $send_asic_to_asic$	-	-	-	-	✓	✓	-	6
④ $send_asic_to_fpga$	-	-	✓	-	-	✓	-	18
⑤ $ack_fpga_to_asic$	-	-	-	✓	-	-	✓	9
⑥ $ack_asic_to_asic$	-	-	-	-	✓	-	✓	5
⑦ $ack_asic_to_fpga$	-	-	✓	-	-	-	✓	17
⑧ $egress_send_asic_to_fpga$	-	✓	✓	-	-	✓	-	50
⑨ $egress_send_fpga_to_asic$	-	✓	-	✓	-	✓	-	42

[a] **ASIC-FPGA** represents the packet sent from the ASIC to the FPGA, **FPGA-ASIC** represents the packet sent from the FPGA to the ASIC, **ASIC-ASIC** represents the packet sent to next en route node, **DATA** and **ACK** represent the DATA packet and acknowledgment packet.
[b] '✓' indicates that the packet has corresponding state characteristics, and '-' indicates that the packet does not have corresponding state characteristics.

TABLE III
PACKET STATE AND ACTION LIST

Action	Packet State								
	①	②	③	④	⑤	⑥	⑦	⑧	⑨
A.$add_RT_MAN_hdr/PATH[0-N]_hdr$	✓[a]	-	-	-	-	-	-	-	-
B.$remove_RT_MAN_hdr/PATH[0-N]_hdr$	-	-	-	-	-	-	-	-	✓
C.$add_METADATA_hdr$	✓	-	✓	-	-	✓	-	-	-
D.$remove_METADATA_hdr$	-	✓	-	-	-	-	-	-	✓
E.$validation_Tofino$	-	-	✓	-	-	✓	-	-	-
F.$lookup_tbl_path_table$ & $update_IP_hdr$	-	-	✓	-	-	-	-	-	-
G.$add_authcode_Tofino$	-	✓	-	-	✓	-	-	-	-
H.$forward_to_next_node$	-	✓	-	-	✓	-	-	-	-
I.$forward_to_FPGA$	✓	-	✓	-	-	✓	-	✓	-
J.$validation_FPGA$	-	-	-	✓	-	-	✓	✓	-
K.$generate_seq$ & $add_authcode_FPGA$	✓	-	-	✓	-	-	-	-	-
L.$cache_packet$ & $record_packet_sending_status$	✓	-	-	✓	-	-	-	-	-
M.$forward_to_Tofino$	✓	-	-	✓	-	-	-	✓	-
N.$generate_ACK_packet$	-	-	-	✓	-	-	-	✓	-
O.$ACK_packet_process$	-	-	-	-	-	-	✓	-	-

[a] '✓' represents the actions that need to be performed on the ASIC and FPGA for packets with the current state, '-' represents the actions that do not need to be performed on the ASIC and FPGA for packets with the current state, or the results obtained from performing this action will not be used for updated the packet header.

TABLE IV
STATE TRANSITION

Current State	Action-List	Next State [a]
①	A,C,I,K,L,M	start and ②
②	D,F,H	③
③	C,E,F,I	④ or ⑧
④	J,K,L,M,N	② and ⑤
⑤	G,H	⑥
⑥	C,E,I	⑦
⑦	O	end
⑧	J,M,N	⑤ and ⑨
⑨	B,D	end

[a] **start** indicates that the packet starts performing the OCEAN protocol. **end** indicates that the packet has completed the OCEAN protocol and will be forwarded as an IP packet or directly dropped in this node.

MACs, OCEAN can ensure that malicious traffic is prevented from entering (**Authentication**) and internal traffic is isolated (**Isolation**). Consequently, we analyze the system's ability to defend against attacks aimed at compromising the end device, en route node, or link, with the intention of degrading or subverting the OCEAN system.

End devices: The gateway ($ingress$ node) could filter the spoofed packet with intra-domain verification. However, if an adversary compromises a source end device, it could send spoofed packets toward the gateway. The gateway cannot filter these packets. Fortunately, since one end device contributes negligible traffic, it does not adversely affect other entities.

When an adversary has compromised a huge number of end devices, the compromised end devices can generate and send a huge number of spoofed packets toward the network. The gateway allows all the packets from these devices and sends them along the forwarding path. It could launch DoS attacks by simply overwhelming a specific en route node's bandwidth resources between the ASIC and FPGA. However, each en route node could negotiate the threshold of the packet number with the gateway as well as the neighbor node in the control plane, and it updates the ACL in ASIC from the control plane if a specific gateway generates too much traffic in a certain interval. It could also report to the network operator in the control plane. Then the network could mitigate the attacks from these end devices.

En route node and link: An adversary who has access to an en route node and link has the capability to inject spoofed packets or potentially drop and replay packets.

We first detail the attacks from the link. With the MACs calculated by the last en route node, each node verifies each packet and filters the packet if verification fails. The probability of detecting and dropping each spoofed packet is $1/2^{L-1}$, where L represents the length of MACs (two 32-bit in OCEAN).

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 02:57:41 UTC from IEEE Xplore. Restrictions apply.

This process ensures that the security goals are not violated. Additionally, since each packet has a sequence number (seq) and en route nodes cache the packet until receiving an ACK from the next node, dropping packets does not compromise the security goals. Regarding replay attacks, as the hash of the packet payload is the input of the $AMAC_{FP_{ij}}$, the next en route node can identify this type of attack when it caches the packet. Therefore, an adversary cannot violate the security goals by accessing the link.

When it comes to the attacks from en route nodes, the next en route node can identify replay attacks from the last node as attacks originating from the link. However, in the event that an adversary compromises one or more en route nodes, they can drop the packet after responding with an ACK to the last node. Fortunately, with the presence of a counter in each node, the first honest node after the compromised node can identify this dropping attack and report it to the network operator. Additionally, the next as well as other en route nodes cannot detect spoofed packets injected by the compromised nodes, each en route node can keep a counter for the verified packets received from the last node and report to the network operator when the counter exceeds a specific threshold [23]. This mechanism allows the system to continue functioning even under attacks from compromised en route nodes and the links.

Besides that, the inter-node packet header includes PATH[0 - N] which is list of the IP address from the current node to the destination. it could enable targeted DoS attacks by allowing an adversary to congest a specific link. Moreover, such attacks can be launched more easily than traditional link-flooding attacks, as the adversary no longer needs to perform network reconnaissance to identify critical links. Nevertheless, each en route node can negotiate a packet quantity threshold with neighboring nodes via the control plane. It updates the Access Control List (ACL) in ASIC via the control plane if a specific neighbor node generates too much traffic in a certain interval.

B. Performance Analysis

Storage overhead: OCEAN presets the dynamic key (DK_{ij}) in the control plane. The first en route ($ingress$) node caches the dynamic key of the end devices (if it needs to verify the source with a MAC) and the neighbor nodes, assuming the length of the key is 16 B, the storage overhead of the first en route node takes $(M + N) \times 16\,B$ for M end devices and N neighbor nodes. The last en route ($egress$) node takes the same storage overhead if the destination needs to verify the packet, while other en route nodes takes $N \times 16\ B$ for N neighbor nodes. Therefore, as a neighbor-based mechanism, the storage overhead incurred by the dynamic key is sustainable with today's hardware. As Key_{ij} does not change frequently, the neighbor based Key_{ij} takes no more space than DK_{ij}.

Besides that, the en route node in OCEAN caches the packet until receiving the ACK from next node. Which is extremely high in the traditional hardware. But as detailed in Section VI, our FPGA could cache up to $8\,GB$. It is sustainable for the traffic in IIoT.

Computation overhead: Computation overhead plays a critical role in determining the ability of the en route node to process the packet at line rate. Each en route node is required to verify the packet using the MACs calculated by the dynamic key. This task proves challenging for traditional hardware such as routers or switches. However, our OCEAN hardware, equipped with lightweight MAC operations on programmable ASIC and secure MAC operations on FPGA, significantly reduces the computation overhead. In Section VI, we will provide evidence to demonstrate that each node is capable of processing the packet at line rate.

Control Plane Key Sharing: We have designed a neighbor-based dynamic key mechanism, where key sharing is analogous to that among brother Address Domains (ADs) in [24], involving direct sharing of the LK_{ij} between neighboring nodes. The process for key distribution when new nodes join or leave the network is as follows: (1) Adding a New Node: When a new en route node joins the network, it only needs to establish a shared key with its neighboring nodes. For instance, if a node has k neighbors, the new node will update its key with these k neighbors. Simultaneously, the k neighbors will obtain the key from the new node. This process ensures that the key sharing remains localized and efficient. (2) Removing a Node: When a node leaves the network, the remaining $k - 1$ neighbors will delete the key associated with the departing node. This ensures that the key space is updated without requiring a global key redistribution. Given that the scale of k is based on the number of neighbors, the complexity of key sharing remains manageable.

V. Implementation

In this section, we first test the correctness of the design in the BMv2[4] simulation environment. Then, we use the Tofino+FPGA hardware platform to develop support for the OCEAN design.

A. Simulation in BMv2

In the BMv2 simulation experiment, a network topology is created based on the diagram shown in Fig. 1. The source and destination hosts are labeled as S and D, respectively. Individual routers are represented by RS_i, RE_i, and RD_i. To simulate the functionality of the FPGA module, virtual terminals (simulated FPGA) are utilized. The program logic of these simulated FPGAs aligns with the FPGA module designed in OCEAN and is implemented using Python language and Scapy 2.4.3. This simulation includes data plane processing, hop-by-hop packet acknowledgment mechanism, and packet caching function of the FPGA. Additionally, independent packet timeout detection and retransmission programs are set up to identify timeout packets based on our timeout detection method. These timeout packets are retrieved from the cache and retransmitted.

1) Hop-by-Hop Delivery: In the hop-by-hop delivery scenario, S sends packets (with UDP destination port number 11223) that require reliable transmission to D. The simulated FPGA at each en route node (RS_i-RD_i) prints the received packet information. The printed packet information shows: (a)

[4]bmv2:https://github.com/p4lang/behavioral-model.

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 02:57:41 UTC from IEEE Xplore. Restrictions apply.

The number of header elements in the range of $PATH[0-N]$ from RS_i to RD_i decreases sequentially. (b) The source and destination address fields in the IP header of the packet are updated at each hop. It is used to verify that the hop-by-hop transmission mechanism of the simulated routers aligns with our design goals.

2) Packet Status Scheduling: The packet status scheduling mechanism involves sending $DATA$ packets (with UDP destination port number 11223) from S to D, which also require reliable transmission. The received and sent DATA packet information is printed by the simulated FPGAs at RS_i-RD_i. The printing information shows:

a) The simulated FPGAs at RS_i-RD_i have received $DATA$ packets from the simulated Tofino, with $DATA$ packet statuses of ①, ④, and ⑧, respectively. After processing by the simulated FPGAs, the $DATA$ packet statuses sent to the simulated Tofino are ②, ②, and ⑨, and the ACK packet status sent to the simulated Tofino is ⑤.

b) The simulated FPGAs at RS_i and RE_i respectively received ACK packets from RE_i and RD_i, with $DATA$ packet status of ⑦.

It verifies that the packet status scheduling mechanism of the simulated environment is consistent with the design goals.

3) Acknowledgment and Packet Caching Mechanism: The acknowledgment and packet caching mechanism is tested by sending $DATA$ packets (with UDP destination port number 11223) from S to D, which require reliable transmission. The simulated FPGAs at RS_i - RD_i are used to print the received $DATA$ packet information. The printing information shows:

a) The simulated FPGAs at RS_i - RD_i have received $DATA$ packets from the simulated Tofino, with $DATA$ packet statuses of ①, ④, and ⑧, respectively.

b) The simulated FPGAs at RS_i and RE_i have received ACK packets from RE_i and RD_i, respectively.

c) By querying the specified storage area of the cache terminal, the result shows that the $DATA$ packets have been successfully cached.

It verifies that the acknowledgment and packet caching mechanism of the simulated FPGA is consistent with the design goals.

4) Timeout Retransmission Mechanism: In the timeout retransmission mechanism, S sends $DATA$ packets (with UDP destination port number 11223) to D for reliable transmission. Packet loss is simulated by intentionally dropping a portion of the packets, which are discarded by the simulated Tofino at RE_i and RD_i. When retransmitting the packets using the simulated FPGA, the TTL field value is set to 10 to differentiate them from other unretransmitted packets. The received $DATA$ packet information is printed by the simulated FPGA at RS_i - RD_i. The printing information shows:

a) The simulated FPGA at RE_i and RD_i will not receive $DATA$ packets with packet loss.

b) After the timeout retransmission time, the simulated FPGA at RE_i and RD_i will receive $DATA$ packets with a TTL field value of 9. This confirms that RS_i and RE_i perform retransmission of cached packets when detecting packet timeouts and not receiving the corresponding ACK packets..

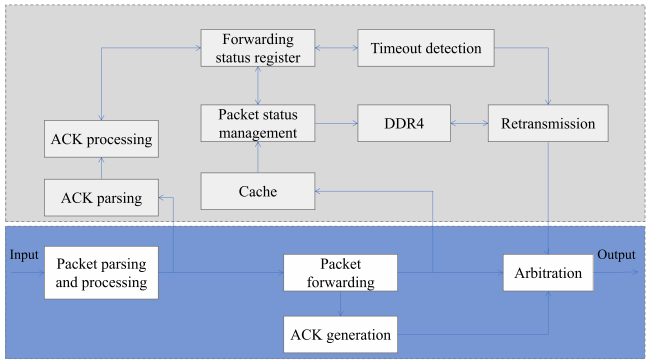


Fig. 5. The processing in FPGA.

It verifies that the timeout retransmission mechanism of the simulated FPGA is consistent with the design goals.

After verifying the functionality in the simulation environment, the next step is to implement the OCEAN software and hardware on the Tofino+FPGA platform.

B. OCEAN Software

The OCEAN software includes 738 lines of P4 code in Tofino2 and 8000 lines of Verilog code in FPGA. Particularly, the processing in FPGA is shown in Fig. 5, it has the following modules:

Packet parsing and processing: This module performs packet parsing, verification, sequence number assignment, MAC operation, and packet type classification. The output signals from this module include the original packet, additional packet header, packet type, and packet verification result.

Packet forwarding: After updating the packet header, the module places the new packet into the packet forwarding queue (**Arbitration** module). It assesses whether the current packet should be discarded or forwarded, whether it needs to be cached (**Cache** module), and whether it should reply with an ACK to the last en route node based on the packet type and packet verification result (**ACK generation** module).

ACK generation: This module is connected to the packet forwarding module. It assesses whether an ACK needs to be generated and generates it if required. The generated ACK is then sent to the ACK queue (**Arbitration** module).

Arbitration: One end of this module is connected to the output port, while the other end is connected to three queues: the $DATA$ forwarding queue, ACK queue, and Retransmission queue. In this paper, a priority-based resource allocation strategy is implemented. $DATA$ packets are assigned the highest priority, followed by ACK packets, and retransmission packets are given the lowest priority.

Cache: The packet forwarding module serves as the input for this module. It assesses whether the packet should be cached and, if so, caches the packet, awaiting the **packet status management** module to read it.

Packet status management: If a packet is present in the cache, the module sends a storage request to the DDR4 controller. Once the $DATA$ packets are written to the **DDR4**, it utilizes the packet

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 02:57:41 UTC from IEEE Xplore. Restrictions apply.

identifier (consisting of the next node's IP address and packet serial number) to calculate and generate the physical address for accessing the **Forwarding status register**. Subsequently, it sends the write command to the register. The information written to the register includes the packet ID, packet forwarding time, ACK status, retransmission time, DDR4 storage address, and packet length.

DDR4: One end of this module is connected to the **Packet status management** module to write the packet to **DDR4**, while the other end is connected to **Retransmission** module to read the packet from **DDR4**.

Forwarding status register: This module stores the packet forwarding status, and arbitrates the request to access the register from the **packet status management**, the **ACK processing**, and the **timeout detection** module.

ACK parsing: This module receives the packet type and packet verification results from the **packet parsing and processing** module. It then extracts the IP address of the ACK (which corresponds to the IP address of the next en route node) and the packet's serial number. These two pieces of information are stored in the ACK information and are subsequently awaited by the **ACK processing** module for reading and further processing.

ACK processing: This module initially reads the ACK information and utilizes a specific algorithm (e.g., CRC32) to calculate the physical address of the register. This address is then used to read the forwarding status register. The module sets the flag indicating whether to receive the corresponding ACK packet in the entry to 1 and writes the updated data back to the corresponding register entry.

Timeout detection: This module accesses the **Forwarding status register** to determine if the packet requires retransmission. If retransmission is necessary, the module extracts the DDR4 storage address and length of the packet from the current register. This information is then passed on to the **retransmission module** for further processing.

Retransmission: The module retrieves timeout packet information that requires retransmission from the **Timeout detection** module. It then sends a request to the **DDR4** to read the packets and stores them in the Retransmission queue. These packets are held in the queue, awaiting processing by the **Arbitration** module.

C. OCEAN Hardware

As depicted in Fig. 6, the OCEAN hardware design comprises an ASIC (Intel Tofino2) and an FPGA.

The COME (COM Express CPU Module) serves as the hardware's management unit, featuring an Intel Pentium D1517 CPU and 16 GB DDR4 SDRAM. It facilitates internal management of TOFINO 2 (Intel Tofino2) and FPGA via PCIE, while external management is possible through Console and Ethernet Ports.

TOFINO 2 (Intel Tofino2) is a programmable ASIC supporting P4, with 4 internal pipelines each housing a Programmable Parser, MAU (Match-Action Unit), and De-Parser. The traffic Manager and Packet Buffer units are shared resources. It supports 10/100/400-Gbps interface. Meanwhile, it supports two 10 Gbps interfaces to connect with the COME.

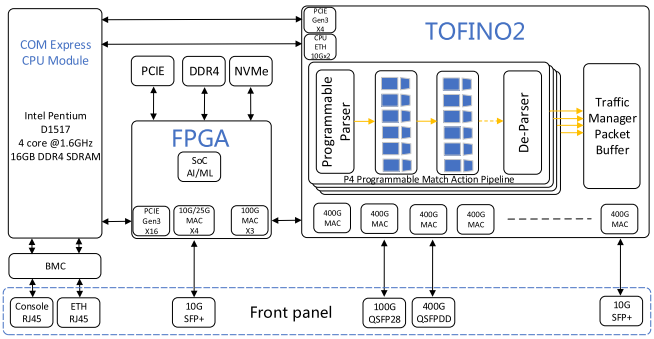


Fig. 6. OCEAN hardware design.

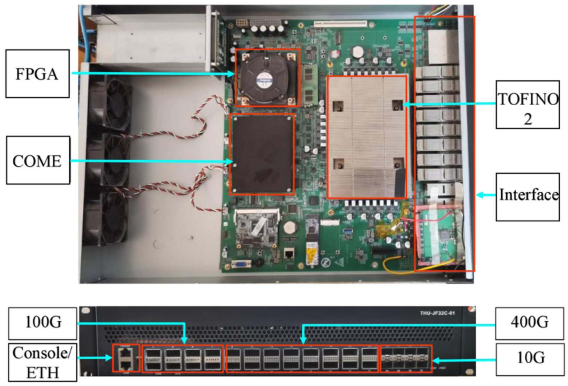


Fig. 7. OCEAN hardware photo.

The FPGA (ZU19EG series of Xilinx) is the key component to implementing our OCEAN protocol. The interface between FPGA and Tofino2 can achieve up to 400 Gbps, and the interface between FPGA and COME is PCIE Gen3 X16. We utilize DDR4 as the cache in this paper. And we could utilize NVMe to extend the cache capability.

The front panel features ports such as Console, Ethernet, and Optical ports on the OCEAN hardware.

We have implemented the hardware, as shown in Fig. 7, we integrate the OCEAN hardware device with 2U chassis. It has 32 Optical Ports, which include 8 x 10-Gbps ports, 8 x 100-Gbps ports, and 16 x 400-Gbps ports. The FPGA has 10 MB on-chip SRAM resources and can access 8 GB DDR4 storage resources. Besides that, the FPGA could connect with NVMe to further extend the storage space. The COME connects with a 120 GB SSD, which supports storage expansion in the control plane.

VI. Evaluation

In BMv2, we evaluated the performance with nine REs, as shown in Fig. 1. First, we measured the end-to-end latency for transmissions with partial OCEAN-supported RE deployment. Then, we analyzed the number of retransmitted packets and the maximum retransmission times under varying packet loss rates. The results demonstrate that OCEAN enhances performance and reliability, even with partial OCEAN-supported RE deployment.

We then leverage our OCEAN hardware to run the P4 and Verilog code. The P4 code consists of 6 stages, whereas TOFINO2

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 02:57:41 UTC from IEEE Xplore. Restrictions apply.

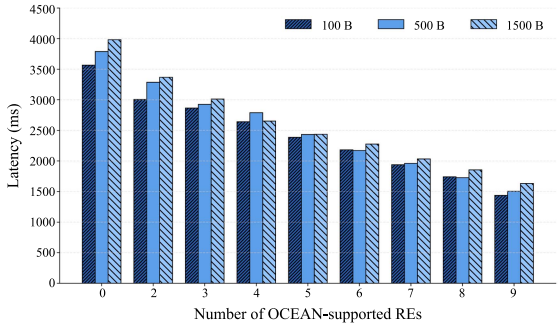


Fig. 8. End-to-end transmission latency under partial OCEAN-supported RE deployment.

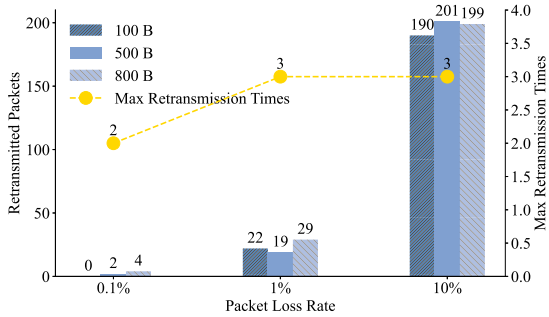


Fig. 9. Number of retransmitted packets and max retransmission times.

comprises 20 stages. Additionally, it occupies only 2.08% of TOFINO2's SRAM capacity. In terms of the FPGA, the Verilog code only occupies 7.91% of LUT (Look-Up Table), 1.49% of LUTRAM (Look-Up Table RAM), 4.64% of FF (Flip-Flop), and 50.41% of BRAM (Block RAM). It does not impact the packet processing. To the best of our knowledge, OCEAN stands as the pioneering programmable hardware integrated with a packet lossless forwarding protocol, enabling caching of every packet at each en route node. In light of this, our initial focus was on testing the processing latency and comparing it to IP packet forwarding. This was done to demonstrate that our solution incurs minimal latency during end-to-end transmission. Subsequently, we conducted an evaluation of the caching ability to confirm the sustainability of our design in conjunction with our hardware.

A. *Performance Under Partial OCEAN-Supported RE Deployment*

We evaluated the impact of partial deployment of OCEAN-supported REs. Specifically, in BMv2, we designed a 10-hop link between RS_i and RD_i where 0 to 9 intermediate nodes were configured to support the OCEAN mechanism. The results show that end-to-end transmission latency decreases with increasing OCEAN-supported RE deployment ratio. In the BMv2 environment, our measurements show that each link has a transmission latency of 40 ms. The timeout while waiting for 3 ACK retransmissions per link is calculated as $40 \times 2 \times 3 = 240$ ms. When all nodes support OCEAN, the timeout latency is 240 ms. When 1 node does not support OCEAN, the latency between the two neighbors becomes 480 ms, and so on. We tested the end-to-end delay of data packets when 0 to 9 intermediate nodes do not support the OCEAN mechanism. As shown in Fig. 8, with 0 to 9 intermediate nodes supporting OCEAN, the end-to-end latency is 3566.2 ms, 3788.9 ms, 3982.6 ms with packet sizes of 100 B, 500 B, and 1500 B when no node supporting OCEAN. While the delay is 1437.2 ms, 1504.5 ms, 1631.6 ms when all the 9 nodes supporting OCEAN. Packet size has minimal impact on latency. The more nodes that lack OCEAN support, the greater the latency. The OCEAN mechanism reduces transmission latency by 60% in full-support scenarios compared to the scenario with no OCEAN support. Note that in BMv2, latency is significantly affected by packet processing, for example, our FPGA-emulated program incurs about 30 ms processing latency. Nevertheless, the results demonstrate that the OCEAN mechanism can effectively reduce transmission latency. In particular, when the source node is an energy-constrained IoT node, retransmitting data packets through en route nodes that support OCEAN can significantly reduce the energy consumption of the IoT node.

Secondly, we tested the number of retransmitted packets. We conducted 100 packets per turn and took the average after multiple turns. As shown in Fig. 9, at 0.1% packet loss rate, the number of retransmitted packets in the whole link is 0, 2, and 4 for 100 B, 500 B and 1500 B respectively. When the packet loss rate is 1%, the numbers of retransmitted packets become 22, 19 and 29 for 100 B, 500 B and 1500 B. At 10% packet loss rate, the corresponding retransmission counts are 190, 201 and 199. The results show that as the link packet loss rate increases, the number of retransmitted packets also increases. When the packet loss rate is below 0.1%, retransmissions are almost unnecessary, while it exceeds 1%, the total number of packet retransmissions required across the entire link increases significantly. Performing retransmissions is therefore highly meaningful. When it comes to the number of max retransmission times required in the same scenario, as the packet loss rate increases from 0.1% to 10%, the results shown in Fig. 9 indicate that at a packet loss rate of 0.1%, only 1 instance requiring 2 retransmissions occurred with 1500 B packets, while all other cases achieved 100% successful transmission with just 1 retransmission, and the max retransmission times is 3 to achieve a data packet delivery success rate of 100% with packet loss rate of 1%. Even at a packet loss rate of 10%, setting 3 retransmissions per link is sufficient to achieve a data packet delivery success rate of 100%. The results demonstrate that packet size has little impact on retransmission times, and 3 retransmissions are adequate to ensure reliable transmission in most of the cases.

B. *Processing Latency*

We test the processing latency of each node. The parameter settings have 3 sizes of packet length (100B, 800–900B, 1300–1400B) and 3 sizes of match-action table (MAT) (3K, 30K, 90K).

We utilize the timestamp that Tofino2 nested in the packet header to get the processing latency of Tofino2 and utilize Xilinx Vivado2021.1 to calculate the processing latency of FPGA. The processing latency in the ingress node is shown in Fig. 10. The latency of Tofino2 ($Tofino\ IN$ or $Tofino\ OUT$) is between 447 ns and 962 ns, the latency of FPGA ($FPGA$)

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 02:57:41 UTC from IEEE Xplore. Restrictions apply.

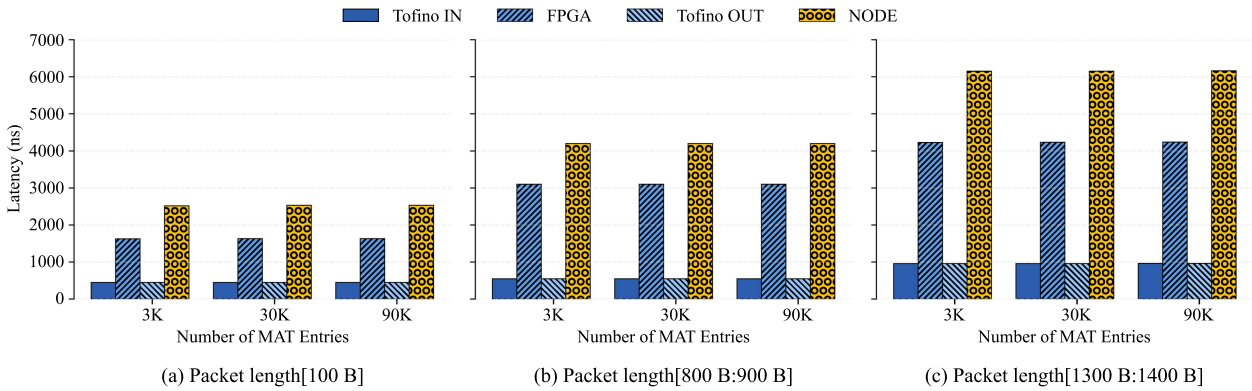


Fig. 10. Processing latency with different numbers of MAT entries in ingress router.

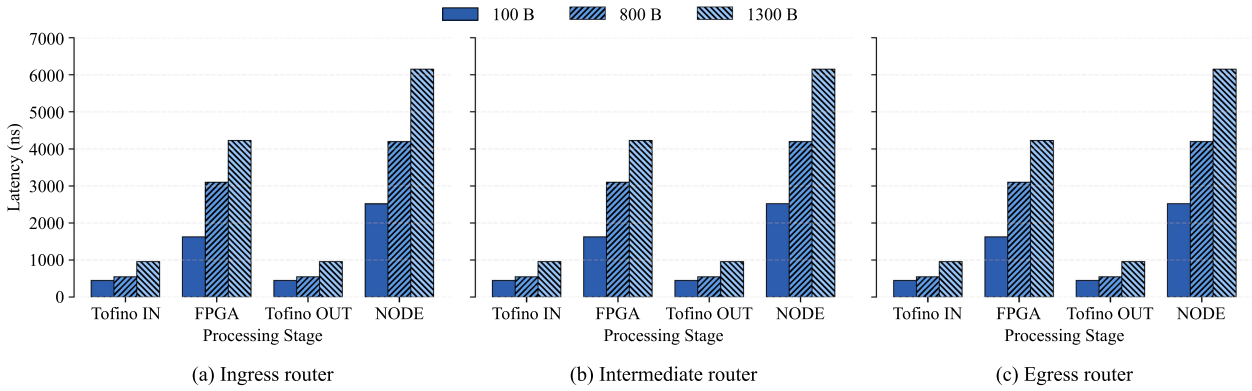


Fig. 11. Processing latency in different nodes.

is between 1625 ns and 4236 ns (including the forwarding latency between the ASIC and FPGA), and the total latency ($NODE$) is between 2519 ns ($447 * 2 + 1625 = 2519$) and 6160 ns ($962 * 2 + 4236 = 6160$). The intermediate and egress nodes have nearly the same latency, while the baseline of IPv4 forwarding is between 364 ns and 920 ns. We could learn that the longer the packet length, the longer the processing latency, and the size of the MAT has little impact on packet processing latency. It is worth noting that we use CRC32 to generate the MAC. In practice, we can opt for a MAC operation with robust security assurances, such as completing 2 AES operations in just 18 ns with dedicated hardware [25].

As shown in Fig. 11, with the scale of 30K MAT entries, different packet sizes have different processing latency. The same result is reflected in the IPv4 packet forwarding (between 364 ns and 920 ns). Our design incurs negligible latency compared with the RTT on the Internet (The RTT on the Internet peaks at 100 ms and 200 ms[5]).

C. Caching Ability

We begin by analyzing the caching capability of a single node using the current hardware resources. Subsequently, we propose a scheme to expand the storage capacity of the node. Finally, we examine the storage space requirements for OCEAN at the network layer across various network configurations.

The caching requirements of each en route node encompass two aspects: (1) the on-chip storage space (SRAM) of the FPGA for recording packet status, and (2) the DDR4 memory for caching packets. Our hardware's FPGA is equipped with 10 MB of SRAM and 8 GB of DDR4 storage space. Considering that the forwarding state of one packet occupies 15 B of storage space, the SRAM can accommodate the forwarding state of 600,000 packets. Furthermore, the total storage space requirements for DDR4 amount to approximately 900 MB for packets with a length of 1500 B.

Additionally, the DDR4 controller provides a read/write clock of 266.5 MHz and supports a maximum of 512 read/write bits. Consequently, the theoretical read/write bandwidth of the DDR4 memory reaches 34 GBps, enabling a bandwidth exceeding 100 Gbps.

The caching ability requirements were determined based on parameters such as forwarding bandwidth (BA), timeout threshold (TO), and the number of retransmissions (N). In Section VI-B, it is detailed that the round-trip time (RTT) on the Internet reaches its peak at 100 ms and 200 ms. Considering that the average path length at the AS level is approximately 4, and the majority of path lengths are less than 10,[6] we have set the default TO to 50 ms and compared it with timeouts

[5]https://www.caida.org/catalog/software/walrus/rtt

[6]https://bgp.potaroo.net/as6447

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 02:57:41 UTC from IEEE Xplore. Restrictions apply.

TABLE V
THE STORAGE SPACE REQUIREMENTS

Timeout	Bandwidth	$SRAM_{sp}$	DDR_{sp}
50ms	10Gbps	3.52MB	0.1875GB
	100Gbps	35.2MB	1.875GB
100ms	10Gbps	7.04MB	0.375GB
	100Gbps	70.4MB	3.75GB
200ms	10Gbps	14.08MB	0.75GB
	100Gbps	140.8MB	7.5GB

of 100 ms and 200 ms. The forwarding bandwidth is set to 10 Gbps and 100 Gbps, the maximum number of retransmissions is set to 3, and the size of each packet is 800 B.[7] As shown in Table V, the storage space of DDR4 is calculated as $DDR_{sp} = TO * BA * N$, and the storage space of SRAM is calculated as $SRAM_{sp} = (DDR_{sp}/800) * 15$. The results indicate that a 10 MB SRAM can support a bandwidth of 10 Gbps with a timeout of 100 ms. Since OCEAN is designed for low-volume, high-value traffic, we can conclude that our hardware is sustainable.

To enhance the caching capability, we propose connecting one ASIC with multiple FPGAs, thereby enabling the utilization of multiple SRAMs. In our design, upon receiving a packet, the Tofino device determines whether it should be forwarded to the next hop or an FPGA. In the case of packet forwarding to an FPGA, different strategies are employed. For $DATA$ packets, a specific FPGA is selected based on the IP address of the next hop. Conversely, for ACK packets, a specific FPGA is chosen based on the IP address of the previous hop. This innovative design allows for the expansion of caching capabilities within a single node, enabling efficient support for high-volume traffic.

D. Throughput

OCEAN is designed to operate effectively in scenarios where most background traffic is forwarded at line rate, while only high-value traffic undergoes OCEAN processing. Nevertheless, to ensure scalability, handling traffic at the 10 Gbps level remains a fundamental processing capability that the OCEAN mechanism must achieve. To validate this, we have conducted throughput tests, which demonstrate that OCEAN can reach a throughput of 10 Gbps. Specifically, our FPGA-based test results show that across different packet sizes (64 B, 500 B, and 1500 B), the system can achieve 10 Gbps, with 14.20 Mpps for 64 B packets (88 B on wire). These results indicate that OCEAN maintains robust performance across varying packet sizes. Such performance ensures that the system remains highly effective even in real-world environments with substantial traffic demands.

VII. DISCUSSION

A. Incremental Deployment

Each en route node selects the next node that supports OCEAN to ensure reliable forwarding. However, as described in Fig. 1, the RE_i is not necessarily support OCEAN. The nodes between the two en route nodes that do not support OCEAN act as transparent links. From this perspective, OCEAN enables incremental deployment with two key characteristics: (1) certain OCEAN nodes can enhance forwarding performance, and (2) early adopters of OCEAN can attract traffic towards them.

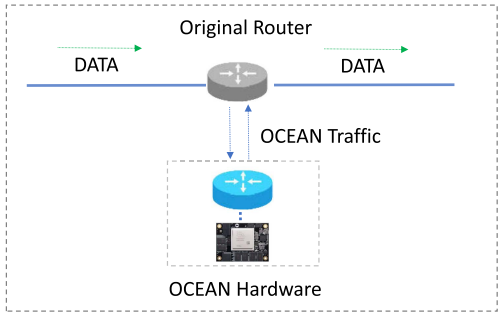


Fig. 12. A deployment example for early adopters.

During the initial deployment phase, adopters have the option to transfer OCEAN traffic from the original router to the OCEAN hardware. The OCEAN hardware is capable of processing the OCEAN protocol and returning the traffic back to the router. As shown in Fig. 12, the original router forwards all received OCEAN packets to a designated port connected to the OCEAN hardware, and subsequently forwards packets originating from the OCEAN hardware based on their IP addresses. This process does not disrupt normal Internet traffic. Additionally, adopters have the option to utilize the OCEAN hardware for more efficient updates to the original router.

Besides that, OCEAN is not limited to supporting mission-critical services in IIoT. Since the source user does not need to participate in retransmission after receiving the ACK from the first en route node, OCEAN can also cater to other scenarios. For instance, in a satellite network with a dynamic topology [26], [27] or the Cloud-Edge-Device (CED) computation system [28], [29], each en route node can retransmit the packet to the next node after a timeout. If the next en route node has changed, it can forward the packet to another neighboring node after a certain number of failures. Additionally, OCEAN, as a network layer protocol, is intentionally designed to be interoperable with other network protocols and can function effectively in the application layer. This adaptability enables OCEAN to meet the changing demands of vertical industries like Virtual Reality and Augmented Reality.

B. Applications in Other Scenarios

Our OCEAN hardware, equipped with FPGA, enables efficient MAC operations such as AES and enhances the packet verification mechanism. Specifically, it can perform calculations for each packet in SCION [30] (as well as EPIC), which is challenging to implement directly in a programmable switch [31]. Additionally, the OCEAN hardware improves the cache ability in the data plane and the fault localization mechanism [18], [32], [33], [34], which require storing the testimony in the

[7] http://mawi.wide.ad.jp/mawi

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 02:57:41 UTC from IEEE Xplore. Restrictions apply.

data plane. This enhancement contributes to building a more secure data plane in the network layer.

VIII. Related Work

Capability-based Networking: SIFF [35] and TVA [36] consider the use of capabilities for DoS protection, which could prevent volumetric attacks, it could not authenticate each packet. SANE uses capabilities in the form of authenticated source routes [37]. But it does not provide packet verification. Hopper constructs a capability mechanism based on a hierarchical key space that is used to uniquely bind each packet to an authorized flow in IIoT [9]. However, the en route node does not cache the packet for the low latency transmission.

In-network caching: In-network caching is critical in Named data networking (NDN) [38]. However, most available NDN router solutions are achieved with software-based in-network caching, and even the highly-optimized designs tailored to specific hardware platforms present limited performance, thus hindering adoption. With the high-throughput and low-latency requirements of large-scale Internet services, in-memory caching for key-value storage was proposed [39], [40]. NetCache achieves fast in-network caching by exploiting the capabilities of programmable switch ASICs. It could achieve 2+ billion queries per second for 64K items with 16-byte keys and 128-byte values. However, it could not satisfy the in-network caching for the whole packets. OCEAN achieves the packet cache with the FPGA. It could also facilitate the in-network caching in clean-slate designs (e.g., NDN).

End-to-end transmission with high security: ARROW is a solution designed to enable users to establish reliable and secure end-to-end paths through participating providers [10]. It provides low latency and high security end-to-end transmission for ARROW traffic with the cooperating ISPs. Other solutions such as BGPsec [41] and SCION [30] can also enhance routing security. SBAS designs and deployes an architecture to bootstrap secure routing [42]. They do not specify the deterministic packet forwarding in the data plane. TSN is a family of standards developed in the TSN Task Group of the IEEE 802.1 group. It works in layer 2 and leads to different requirements for the connected TSN domains, which in turn leads to different employed TSN mechanisms in heterogeneous networks. DetNet is tailored for the deterministic end-to-end forwarding in the network layer. It could facilitate the connection of different TSN domains. Wüsteney et al. propose a model that allows calculating best-case and worst-case latencies for time-critical communication across different DetNet-connected TSN networks. It enables validating end-to-end communication requirements in open, multi-purpose industrial networks [43]. OCEAN could improve the performance of DetNet since the en route node retransmits packets as soon as possible.

Packet verification at en route node: ICING [3], OPT [4] and EPIC [6] authenticate each packet at each en route node. PPV [5] and MASK [8] decrease the overhead of packet verification at the en route router by probabilistically verifying. Hummingbird only sample packets probabilistically and based on symmetric keys shared between neighboring routers [44]. These mechanisms were not tailored for mission-critical applications, they need a powerful source and destination. OCEAN is a packet verification mechanism in which each trusted en route node builds a secure network for the users, the source needs not to take part in the packet verification in each en route node.

Programmable network data plane: The emerging programmable switch hardware, which supports packet processing using domain-specific languages (e.g., P4 [45]), provides an exciting opportunity to design protocols that manipulate arbitrary parts of packet header at line rate [46]. Researchers have looked at accelerating various applications in networking, which includes DDoS defense [47], [48], load balancing [49], [50], packet verification [7], blocking large-scale SYN floods [51], deploying machine learning (ML) models [52], [53], key-value stores [54], network monitoring and measurement tasks [55]. These applications deliver significantly better performance at lower cost than their software counterparts on commodity servers. Menshen extends the Reconfigurable Match Tables (RMT) pipeline that enforces isolation between different packet-processing modules [56]. However, implementing the complex processing and in-network caching with the programmable switch still poses challenges since the ASIC constraints and space limitation even with some new extension of P4 have been proposed [57]. FENIX is a hybrid in-network ML system that performs feature extraction on programmable switch ASICs and deep neural network inference on FPGAs [15]. Based on this concept, our OCEAN innovatively addresses the existing limitations by leveraging the FPGA-enhanced approach to extend the pipeline and increase the caching space.

IX. Conclusion

We have developed and implemented OCEAN with the objective of achieving packet lossless forwarding in the network layer data plane. OCEAN leverages the deep integration of $Intel\ Tofino2$ and $FPGA$ to enable a programmable data plane and in-network caching. By caching packets within the network until receiving the ACK from the next node, OCEAN enhances end-to-end transmission in various networks, including IIoT and other networks (e.g., satellite networks). OCEAN serves as an optional forwarding strategy that does not compromise packet forwarding in the data plane. It not only reduces forwarding latency but also enhances security through the MAC calculated with the dynamic keys. Additionally, OCEAN is designed to be compatible with other network protocols and functions, making it adaptable to the evolving requirements of vertical industries such as Virtual Reality and Augmented Reality.

References

[1] Y. Zhao, K. Xu, H. Wang, B. Li, M. Qiao, and H. Shi, "MEC-enabled hierarchical emotion recognition and perturbation-aware defense in smart cities," *IEEE Internet Things J.*, vol. 8, no. 23, pp. 16933–16945, Dec. 2021.

[2] P. Cong et al., "A deep reinforcement learning-based multi-optimality routing scheme for dynamic IoT networks," *Comput. Netw.*, vol. 192, 2021, Art. no. 108057.

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 02:57:41 UTC from IEEE Xplore. Restrictions apply.

[3] J. Naous, M. Walfish, A. Nicolosi, D. Mazières, M. Miller, and A. Seehra, "Verifying and enforcing network paths with icing," in *Proc. 2011 Conf. Emerg. Netw. Experiments Technol.*, 2011, pp. 1–12.

[4] T. H. Kim, C. Basescu, L. Jia, S. B. Lee, Y. Hu, and A. Perrig, "Lightweight source authentication and path validation," in *Proc. ACM SIGCOMM Conf.*, 2014, pp. 271–282.

[5] B. Wu et al., "Enabling efficient source and path verification via probabilistic packet marking," in *Proc. 26th IEEE/ACM Int. Symp. Qual. Service*, 2018, pp. 1–10.

[6] M. Legner, T. Klenze, M. Wyss, C. Sprenger, and A. Perrig, "EPIC: Every packet is checked in the data plane of a path-aware Internet," in *Proc. 29th USENIX Secur. Symp.*, 2020, pp. 541–558.

[7] S. Fu et al., "MASK: Practical source and path verification based on multi-as-key," in *Proc. 29th IEEE/ACM Int. Symp. Qual. Service*, Tokyo, Japan, 2021, pp. 1–10.

[8] S. Fu et al., "MASK: Practical source and path verification based on multi-as-key," *IEEE/ACM Trans. Netw.*, vol. 31, no. 4, pp. 1478–1493, Aug. 2023.

[9] P. D. Vaere, A. Tulimiero, and A. Perrig, "Hopper: Per-device nano segmentation for the industrial IoT," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, Nagasaki, Japan, 2022, pp. 279–293.

[10] S. Peter, U. Javed, Q. Zhang, D. Woos, T. E. Anderson, and A. Krishnamurthy, "One tunnel is (often) enough," in *Proc. ACM SIGCOMM 2014 Conf.*, Chicago, IL, USA, 2014, pp. 99–110.

[11] N. Finn, P. Thubert, B. Varga, and J. Farkas, "Deterministic networking architecture," *RFC*, vol. 8655, pp. 1–38, 2019.

[12] Z. Lai et al., "Futuristic 6G pervasive on-demand services: Integrating space edge computing with terrestrial networks," *IEEE Veh. Technol. Mag.*, vol. 18, no. 1, pp. 80–90, Mar. 2023.

[13] M. Scazzariello, T. Caiazzi, H. Ghasemirahni, T. Barbette, D. Kostic, and M. Chiesa, "A high-speed stateful packet processing approach for tbps programmable switches," in *Proc. 20th USENIX Symp. Networked Syst. Des. Implementation*, Boston, MA, 2023, pp. 1237–1255.

[14] A. Krentsel, P. Schafhalter, J. E. Gonzalez, S. Ratnasamy, S. Shenker, and I. Stoica, "Managing Bandwidth: The key to cloud-assisted autonomous driving," 2024, *arXiv:2410.16227*.

[15] X. Gao et al., "FENIX: Enabling in-network DNN inference with FPGA-enhanced programmable switches," 2025, *arXiv:2507.14891*.

[16] P. Cong et al., "DIT and beyond: Interdomain routing with intradomain awareness for IIoT," *IEEE Internet Things J.*, vol. 10, no. 23, pp. 20602–20616, Dec. 2023.

[17] M. Shen et al., "Blockchain-assisted secure device authentication for cross-domain industrial IoT," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 5, pp. 942–954, May 2020.

[18] S. Fu et al., "D3: Lightweight secure fault localization in edge cloud," in *Proc. 42nd IEEE Int. Conf. Distrib. Comput. Syst.*, Bologna, Italy, 2022, pp. 515–525.

[19] L. Wang, H. Kim, P. Mittal, and J. Rexford, "Programmable in-network obfuscation of DNS traffic," in *Proc. Annu. Netw. Distrib. Syst. Secur. Symp.: DNS Privacy Workshop*, 2021, pp. 1–10.

[20] X. Chen, "Implementing AES encryption on programmable switches via scrambled lookup tables," in *Proc. ACM SIGCOMM Workshop Secure Program. Netw. Infrastructure*, 2020, pp. 8–14.

[21] J. Wu, J. Bi, M. Bagnulo, F. Baker, and C. Vogt, "Source address validation improvement (SAVI) framework," *RFC*, vol. 7039, pp. 1–14, 2013.

[22] J. Bi, G. Yao, J. M. Halpern, and E. Levy-Abegnoli, "Source address validation improvement (SAVI) for mixed address assignment methods scenario," *RFC*, vol. 8074, pp. 1–12, 2017.

[23] X. Zhang, C. Lan, and A. Perrig, "Secure and scalable fault localization under dynamic traffic patterns," in *Proc. IEEE Symp. Secur. Privacy*, San Francisco, California, USA, 2012, pp. 317–331.

[24] X. Wang et al., "Toward practical inter-domain source address validation," *IEEE/ACM Trans. Netw.*, vol. 32, no. 4, pp. 3126–3141, Aug. 2024.

[25] B. Rothenberger, D. Roos, M. Legner, and A. Perrig, "PISKES: Pragmatic internet-scale key-establishment system," in *Proc. 15th ACM Asia Conf. Comput. Commun. Secur.*, Taipei, Taiwan, 2020, pp. 73–86.

[26] S. Yao, J. Guan, Y. Wu, K. Xu, and M. Xu, "Toward secure and lightweight access authentication in SAGINs," *IEEE Wirel. Commun.*, vol. 27, no. 6, pp. 75–81, Dec. 2020.

[27] S. Yao, J. Guan, Z. Yan, and K. Xu, "SI-STIN: A smart identifier framework for space and terrestrial integrated network," *IEEE Netw.*, vol. 33, no. 1, pp. 8–14, Jan./Feb. 2018.

[28] B. Lin, Y. Huang, J. Zhang, J. Hu, X. Chen, and J. Li, "Cost-driven offloading for DNN-based applications over cloud, edge and end devices," *IEEE Trans. Ind. Inform.*vol. 16, no. 8, pp. 5456–5466, Aug. 2019.

[29] S. Yao et al., "Blockchain-empowered collaborative task offloading for cloud-edge-device computing," *IEEE J. Sel. Areas Commun.*, vol. 40, no. 12, pp. 3485–3500, Dec. 2022.

[30] A. Perrig, P. Szalachowski, R. M. Reischuk, and L. Chuat, *SCION: A Secure Internet Architecture*. Berlin, Germany: Springer, 2017.

[31] J. de Ruiter and C. Schutijser, "Next-generation internet at terabit speed: SCION in P4," in *Proc. The 17th Int. Conf. Emerg. Netw. EXperiments Technol.*, Virtual Event, Munich, Germany, 2021, pp. 119–125.

[32] B. Barak, S. Goldberg, and D. Xiao, "Protocols and lower bounds for failure localization in the Internet," in *Proc. 27th Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, Istanbul, Turkey, 2008, pp. 341–360.

[33] C. Basescu, Y. Lin, H. Zhang, and A. Perrig, "High-speed inter-domain fault localization," in *Proc. IEEE Symp. Secur. Privacy*, San Jose, CA, USA, 2016, pp. 859–877.

[34] B. Wu et al., "RFL: Robust fault localization on unreliable communication channels," *Comput. Netw.*, vol. 158, pp. 158–174, 2019.

[35] A. Yaar, A. Perrig, and D. X. Song, "SIFF: A stateless internet flow filter to mitigate DDoS flooding attacks," in *Proc. 2004 IEEE Symp. Secur. Privacy*, Berkeley, CA, USA, 2004, pp. 1–14.

[36] X. Yang, D. Wetherall, and T. E. Anderson, "TVA: A DoS-limiting network architecture," *IEEE/ACM Trans. Netw.*, vol. 16, no. 6, pp. 1267–1280, Dec. 2008.

[37] M. Casado, T. Garfinkel, A. Akella, M. J. Freedman, D. Boneh, and N. McKeown, "SANE: A protection architecture for enterprise networks," in *Proc. 15th USENIX Secur. Symp.*, Vancouver, BC, Canada, 2006, pp. 137–151.

[38] L. Zhang, "Evolving Internet to the future via named data networking," in *Proc. Asian Internet Eng. Conf.*, Bangkok, Thailand, 2011, Art. no. 72.

[39] A. Dragojevic, D. Narayanan, M. Castro, and O. Hodson, "FaRM: Fast remote memory," in *Proc. 11th USENIX Symp. Networked Syst. Des. Implementation*, Seattle, WA, USA, 2014, pp. 401–414.

[40] Y. Lee, H. A. Maruf, M. Chowdhury, A. Cidon, and K. G. Shin, "Hydra: Resilient and highly available remote memory," in *Proc. 20th USENIX Conf. File Storage Technol.*, Santa Clara, CA, USA, 2022, pp. 181–198.

[41] M. Lepinski and K. Sriram, "BGPsec protocol specification," *RFC*, vol. 8205, pp. 1–45, 2017.

[42] H. Birge-Lee et al., "Creating a secure underlay for the Internet," in *Proc. 31st USENIX Secur. Symp.*, Boston, MA, USA, 2022, pp. 2601–2618.

[43] L. Wüsteney et al., "Analyzing and modeling the latency and jitter behavior of mixed industrial TSN and DetNet networks," in *Proc. 18th Int. Conf. Emerg. Netw. EXperiments Technol.*, 2022, pp. 91–109.

[44] A. He et al., "Hummingbird: Dynamic path validation with hidden equal-probability sampling," *IEEE Trans. Inf. Forensics Secur.*, vol. 18, pp. 1268–1282, 2023.

[45] P. Bosshart et al., "P4: Programming protocol-independent packet processors," *Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, 2014.

[46] N. Foster, N. McKeown, J. Rexford, G. Parulkar, L. Peterson, and O. Sunay, "Using deep programmability to put network owners in control," *SIGCOMM Comput. Commun. Rev.*, vol. 50, no. 4, pp. 82–88, Oct. 2020.

[47] G. Li et al., "Enabling performant, flexible and cost-efficient DDoS defense with programmable switches," *IEEE/ACM Trans. Netw.*, vol. 29, no. 4, pp. 1509–1526, 2021.

[48] Z. Zhao et al., "Effective DDoS mitigation via ML-driven in-network traffic shaping," *IEEE Trans. Dependable Secur. Comput.*, vol. 21, no. 4, pp. 4271–4289, Jul./Aug. 2024.

[49] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu, "SilkRoad: Making stateful layer-4 load balancing fast and cheap using switching ASICs," in *Proc. Conf. ACM Special Int. Group Data Commun.*, Los Angeles, CA, USA, 2017, pp. 15–28.

[50] M. Zhang et al., "Poseidon: Mitigating volumetric DDoS attacks with programmable switches," in *Proc. 27th Annu. Netw. Distrib. System Secur. Symp.*, San Diego, CA, USA, 2020, pp. 1–18.

[51] S. Yoo, X. Chen, and J. Rexford, "SmartCookie: Blocking large-scale SYN floods with a split-proxy defense on programmable data planes," in *Proc. 33rd USENIX Secur. Symp.*, 2024, pp. 1–18.

[52] G. Zhou, Z. Liu, C. Fu, Q. Li, and K. Xu, "An efficient design of intelligent network data plane," in *Proc. 32nd USENIX Secur. Symp.*, Anaheim, CA, USA, 2023, pp. 6203–6220.

[53] J. Yan et al., "Brain-on-switch: Towards advanced intelligent network dataplane via NN-driven traffic analysis at line-speed," in *Proc. 21th USENIX Symp. Networked Syst. Des. Implementation*, 2024, pp. 419–440.

[54] M. Liu, L. Luo, J. Nelson, L. Ceze, A. Krishnamurthy, and K. Atreya, "IncBricks: Toward in-network computation with an in-network cache," in *Proc. 22nd Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2017, Xi'an, China, 2017, pp. 795–809.

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 02:57:41 UTC from IEEE Xplore. Restrictions apply.

[55] J. Sonchack, O. Michel, A. J. Aviv, E. Keller, and J. M. Smith, "Scaling hardware accelerated network monitoring to concurrent and dynamic queries with ∗Flow," in *Proc. 2018 USENIX Annu. Tech. Conf.*, Boston, MA, USA, 2018, pp. 823–835.
[56] T. Wang, X. Yang, G. Antichi, A. Sivaraman, and A. Panda, "Isolation mechanisms for high-speed packet-processing pipelines," in *19th USENIX Symp. Networked Syst. Des. Implementation*, Renton, WA, USA, 2022, pp. 1289–1305.
[57] M. Hogan, S. L. Feibish, M. T. Arashloo, J. Rexford, D. Walker, and R. Harrison, "Elastic switch programming with P4All," in *Proc. 19th ACM Workshop Hot Topics Netw.*, Virtual Event, USA, 2020, pp. 168–174.



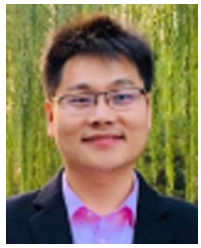
Su Yao (Senior Member, IEEE) received the PhD degree from the National Engineering Laboratory for Next Generation Internet, Beijing Jiaotong University. Currently, he is an associate researcher with Beijing National Research Center for Information Science and Technology (BNRist), Tsinghua University. His research interests include future network architecture and the IoT security.



Songtao Fu received the PhD degree from Tsinghua University, Beijing, China, in 2023. His research interests include Internet architecture and network security.



Qi Li (Senior Member, IEEE) received the PhD degree from Tsinghua University, Beijing, China. He is currently an associate professor with the Institute for Network Sciences and Cyberspace, Tsinghua University. He worked at ETH Zurich and The University of Texas at San Antonio. His research interests include network and system security, particularly in internet and cloud security, mobile security, and big data security. He is an Editorial Board Member of the *IEEE Transactions on Dependable and Secure Computing* and ACM DTRAP.



Zhuotao Liu received the PhD degree from the University of Illinois at Urbana-Champaign, USA. He is currently an associate professor with Tsinghua University. Before joining Tsinghua, he was a Technical Lead with Google, managing one of world's largest software-defined datacenter networks. His research interests include data/AI security and privacy, blockchain and applied cryptography, and secure internet architecture.



Yi Cai received the ME degree from Tsinghua University, Beijing, China, in 2023. His research interests include Internet architecture and network security.



Yinchao Zhang is currently working toward the master's degree with the Department of Computer Science and Technology, Tsinghua University, Beijing, China. His primary research interst includes intelligent programmable switch architectures.



Ke Xu (Fellow, IEEE) received the PhD degree from the Department of Computer Science and Technology, Tsinghua University, Beijing, China. He is currently a Full Professor with the Department of Computer Science and Technology, Tsinghua University. He has authored or coauthored more than 200 technical articles and holds more than ten U.S. patents in the research areas of next generation internet, blockchain systems, the Internet of Things, and network security. He is also a member of ACM. He has won the IWQoS 2024 Best Paper Award, the Distinguished Paper Award from USENIX Security 2023, and 2024 and NDSS 2025.

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 02:57:41 UTC from IEEE Xplore. Restrictions apply.