Hindawi
Journal of Computer Networks and Communications
Volume 2021, Article ID 5599254, 13 pages
https://doi.org/10.1155/2021/5599254

Research Article

Performance Analysis of Identifier Locator Communication Cache Effects on ILNPv6 Stack

Mohsen Kadi, Maher Suleiman, and Samih Jammoul

Faculty of Information Technology, Higher Institute for Applied Sciences and Technology, Damascus, Syria

Correspondence should be addressed to Mohsen Kadi; mohsen.kadi@hiast.edu.sy

Received 15 February 2021; Revised 11 April 2021; Accepted 3 May 2021; Published 13 May 2021

Academic Editor: Cong Pu

Identifier-locator network protocol (ILNP) is a host-based identifier/locator split architecture scheme (ILSA), which depends on address rewriting to support end-to-end mobility and multihoming. The address rewriting is performed by hosts using a network layer logical cache that stores state information related to the communicated hosts, which is called identifier-locator communication cache (ILCC). Since address rewriting is executed on a packet basis in ILNP, ILCC lookups are required at each packet reception and transmission. This leads to a strong correlation between the host's network stack performance and ILCC performance. This paper presents a study of the effect of ILCC size on network stack performance. Within this paper, a direct comparison of the performance of two ILNP prototypes that differ by ILCC management mechanism is conducted. We present ILCC size measurements and study their effects on the host's network stack performance. The results show that ILCC growth caused by correspondents increase has a significant effect on the latency of both network and transport layers. The obtained results show that controlling ILCC size through an effective policy strongly enhances ILNP network stack performance.

1. Introduction

In the last two decades, as a result of many research efforts, there is a common recognition that Internet routing and addressing architecture is encountering challenges in scalability, multihoming, and interdomain traffic engineering [1–4]. To overcome these challenges and cope with the present Internet growth levels without inducing prohibitive increases in network operating expenses, several solutions have been proposed, and many of them are based on ILSA [5–7].

One of the host-based ILSA schemes is ILNP. The architectural concept of ILNP replaces the present address space (IP) with two address spaces, specifically the locators and identifiers. The architectural concept of ILNP is independent of the IP version, but it adapts the same packet format, and thus, there are two variants of ILNP: ILNPv4, which is based on IPv4 packet format, and ILNPv6, which is based on IPv6 packet format [8–10].

ILNP provides an enhancement to current Internet architecture via enriching the set of used namespaces instead of adopting a clean slate namespace. It adapts the evolutionary approach, which assumes that IP will remain the namespace for the Internet. Based on that assumption, it detaches the semantics overloading of IP by splitting between the node identifier (NID), which is a nontopological name uniquely identifying a node, and the locator (Loc), which is a topologically bound name for an IP subnetwork. Therefore, in IPv6 packet headers, each 128 bit IPv6 address is separated into 64 bits for the Loc and 64 bits for NID [11].

As RFC6741 [12] indicates, the ILNP network stack implementations stores state information of the communicated hosts at the network layer in ILCC, which keeps information like current valid NIDs, Locs with their precedence, Loc lifetime and nonce values for the local host, and current and recent correspondents. Therefore, ILNP presented the notion of network layer session to facilitate ILNP address rewriting in order to support end-to-end mobility and multihoming.

As a protocol address rewriting facility, ILCC must always provide up-to-date mapping information among Locs and NIDs. Consequently, ILCC's lookups became a critical

part of every packet reception and transmission in the ILNP protocol stack. In ILNP transmission, the network layer receives only the local and remote NIDs from the upper layer. After that, it chooses the most favorable Locs for both remote and local hosts from the ILCC. Likewise in reception, when network layer receives a packet for local delivery, it extracts two pairs of NIDs and Locs belonging to remote and local host from packet header, and afterwards, the network layer conducts a validity check on the extracted pairs and decides either to proceed in packet reception or drop it. After passing the former check, the network layer will only handle NIDs to the transport layer. More details about this process can be found in RFC6740 [11].

In this paper, we investigate the performance of two ILNPv6 prototypes when user datagram protocol (UDP) is used as a transport layer protocol. The first prototype is developed by the University of St. Andrews and available at [13], and the second is proposed by us in [14]. The main difference between the two prototypes, which this paper focuses on, is the policy used to manage ILCC.

The key contributions of this paper are as follows:

(i) We conducted a comprehensive comparison between two ILNPv6 prototypes in terms of ILCC size and packet serving and dropping rates

(ii) We measured the latency introduced by both prototypes at different network stack layers

(iii) We also performed a system profiling for the prototypes and analyzed the internal load distribution of each one

(iv) Based on the previous results, we showed how ILCC uncontrolled growth can introduce a significant kernel latency, and how a worst-case scenario, for example, by a malicious or bugged host, can increase the latency by several milliseconds and how to avoid this worst-case latency

The rest of the paper is organized as follows: a review of related work is presented in Section 2. Section 3 discusses the current ILNP caching policies. The test environment, developed traffic generator, used toolset, and evaluation scenarios are introduced in Section 4. Section 5 presents the obtained results including performance metrics and analysis. Finally, Section 6 concludes this work.

2. Related Work

ILNP is one of the evolutionary solutions recommended by the Internet research task force (IRTF) routing research group to enhance current Internet routing architecture. It was the subject of many types of researches. For instance, Abid et al. [15] developed an analytical cost model to study the performance of three ILSA schemes, namely, ILNP and host identity protocol (HIP) as a representative for host-based ILSA scheme, and the locator/identifier separation protocol (LISP) as a network-based ILSA scheme. Multiple performance metrics were noted in the analytical model, like connection establishment cost, data packet delivery cost, and impacts of the number of correspondents. The analysis results indicate that a network with a large number of correspondents decreases the performances of both host-based and network-based ILSA schemes dramatically. The work in [16] presents the first implementation of UDP over ILNPv6 in Linux kernel. The introduced performance evaluation of UDP over ILNPv6 using this implementation shows that ILNPv6 handoff performance is better than Mobile IPv6 in terms of throughput, packet loss, and handoff delay. Our relevant previous work [14] focuses on performance implications resulting from ILCC uncontrolled growth, and we proposed a policy with three-way handshake as an enabling mechanism to control ILCC growth. The preliminary evaluation focuses on the network layer performance, and the results show that the proposed policy can avoid network layer latency via avoiding unnecessary ILCC entries adoption.

While the previous work on ILNP either provides an analytical model to study the ILNP performance [15] or demonstrates this protocol support for mobility [16], the significant contribution in this paper, beyond the results of the previous work in [14], is that a comprehensive comparison between the two ILNPv6 prototypes is introduced. Moreover, the related work [14] measures only the kernel latency at the network layer under different loads. We measure both ILCC size at different loads and the kernel latency at different network stack layers for different ILCC sizes and study their effects on packet serving and dropping rates. Besides that, system profiling is presented to point out reasons for performance degradation. The test setup is also improved over the work in [14] as we used a custom traffic generator, which will be described later in this paper, in addition to using a set of tools to monitor the internal and external ILNPv6 host's behavior.

3. Current ILNP Caching Policies

RFC6741 [12] introduced the concept of ILCC as a required local cache of state information that enables ILNP operations. However, it does not include any description of how to manage ILCC and the rules to add new records to it. Moreover, the prototype implementation of ILNPv6 [13] adopts a simple policy to manage ILCC. In this policy, the host adds a new ILCC entry for each new correspondent based on its network layer state without any restriction. In this implementation, for each new local delivery packet, it performs an ILCC lookup, and if the correspondent does not exist, it simply adds a new entry to ILCC, without taking into consideration the state of the host's higher layers or the requirements of the flow, to which this packet belongs. Hence, there is a bijection between the number of ILCC entries and the number of correspondents.

In order to overcome the problem of uncontrolled ILCC growth when hosts exchange data over UDP, we introduced in a previous work [14] a new policy for ILCC entries adoption. Following this policy, adding a new entry to ILCC is not a pure network layer decision, but rather a decision taken in an integrated manner between the network layer and the transport layer. Using this policy, the decision of adding a new entry to ILCC depends on multiple criteria like

the added value provided by this entry to the communication ends, mutual approval, and availability of the resources at both ends. The added value of ILCC entry is derived from the type of UDP sockets at both ends. Based on the nature of UDP sockets running on those ends, ILCC entry is only useful when one or both sockets at the ends are connected. Table 1 summarizes the added values of ILCC entries and the effect of different UDP socket types on ILCC size. The introduced policy adopts a three-way handshake as an enabling mechanism for both ends mutual approval and depends on new Internet control message protocol for IPv6 (ICMPv6) messages as a dedicated and independent control plan that maintains the nature of the data exchange provided by the UDP. The proposed policy takes into consideration many factors, which may affect the communicating hosts' approval, such as the count of network layer sessions and available resources. If the communicated hosts do not reach a mutual approval, the data exchange continues between them, but this exchange will not benefit from ILCC's services (more details in [14]).

Later on, we will refer to the Linux kernel that runs the prototype of the University of St. Andrews as "without policy" kernel and "with policy" kernel to the kernel that runs our prototype.

4. Test Methodology and Setup

The following section describes the used test environment, developed traffic generator, evaluation scenarios, and used toolset.

4.1. Test Environment. In the conducted experiments, data were sent over ILNPv6 in one direction between two hosts. As shown in Figure 1, the source and target are each a separate virtual machine and hosted on the same VMWare enterprise-class hypervisor (ESXI). Both hosts are located on the same virtual local area network (VLAN) and connected to ESXI's VMware's virtual switch (vSwitch) using a 1 Gbps link as labeled. The features of source and target are shown in Table 2. The ethernet adapter at the target features four receive and transmit ring buffers, and each of them has 256 descriptors.

4.2. Traffic Generator. The developed traffic generator uses PF_PACKET sockets to generate specially constructed UDP over ILNPv6 packets, as this type of sockets provides a very powerful way to send data packets by directly accessing a network device [17]. It generates IPv6 packets with nonce destination option, so the target, as ILNP capable host, will handle these packets as ILNPv6 packets. All generated packets have the same size (76 bytes), and the generator codebase is written in C and Bash. Figure 2 shows the architecture of the generator. At start-up, it runs a master sh file, which controls packet generation volume, type, and duration via user-provided parameters. In addition, it controls the monitoring processes. Based on the user-provided parameters, it spawns several processes responsible for generating UDP packets destined to the target.

The generator has two kinds of generation processes to generate two different types of UDP datagrams. The first represents identical echo requests from a fixed source {NID, Loc} pair to an echo service running on the target. As the packet fields are constant in this case, the generation process of this kind is simple. After start-up, each process depends on a predefined packet template to initialize and fill all data structures required for the packets. Then, it enters a loop that depends on PF_PACKET sockets to send identical packets to the target for the duration specified by the master sh file. In the second type of UDP datagrams, each produced UDP packet has a randomly generated source NID. This type of UDP datagrams will be used to emulate an increase in the number of correspondents at the target. As a result, the generation processes of this kind are more complicated compared with the previous kind, and we will call them the load process. After start-up, each load process initializes and fills some data structures required for the packets. After that, it enters a loop to perform four operations: generating a random source NID, filling packets structure, calculating UDP pseudoheader checksum for ILNP using only source and destination NIDs as stated in [11], and finally sending the packet to the target using PF_PACKET socket.

Using the features listed in Table 2, the generator generates the traffic volume shown in Figure 3 using twenty first-kind generation processes and a variable number of load processes. However, this volume is sufficient to show the bottlenecks of the analyzed systems.

During experiments, the number of load processes is changed from 0 to 10. The naming convention used in this work is as follows: the count of load processes represents the test rank. We will refer to each test with its rank and "Test" prefix. Test0 is the test with zero load process, and so on. Each test round lasts for 60 seconds.

4.3. Measurement Tools. As this work focuses on performance evaluation of ILNPv6 stack, data about the internal and external behavior of the target was gathered. To collect information about the external behavior of the ILNPv6 network stack during experiments *tcpdump* and *ethtool* were used. *tcpdump* was used to record all on-wire traffic, while *ethtool* was used to get the statistics of network interface controller (NIC) driver. The information about internal ILNPv6 network stack behavior was collected using *trace-cmd* and *perf*. *trace-cmd* is a user-space front-end tool that interacts with the Ftrace tracer built inside the Linux kernel [18], while *perf* is a performance analysis tool for Linux capable of lightweight profiling [19].

4.4. Test Setup for Transport Layer. In transport layer test scenario, the target only runs an echo service that simply sends back to the originating source any data it receives. In this test scenario, an evaluation was conducted for the transport layer of the two ILNPv6 prototypes at two stages. In each stage, eleven tests were performed in order to fulfill the stage objective. The objective of the first stage is to measure ILCC size, delay added by the kernel, and packet serving and dropping rate, while the objective of the second

Table 1: ILCC entries with different types of UDP socket.

		Server	
		Connected UDP socket	Nonconnected UDP socket
Client	Connected UDP socket	(i) ILCC entry required (if available resources allow) (ii) One entry per socket at each end	(i) LCC entry required (if available resources allow) (ii) One ILCC entry at the end with connected socket (iii) Nonconnected socket may lead to unlimited growth in ILCC at its host
	Nonconnected UDP socket	(i) ILCC entry required (if available resources allow) (ii) One ILCC entry at the end with connected socket (iii) Nonconnected socket may lead to unlimited growth in ILCC at its host	(i) ILCC entry adoption must be avoided (ii) Each socket may cause unlimited growth in ILCC

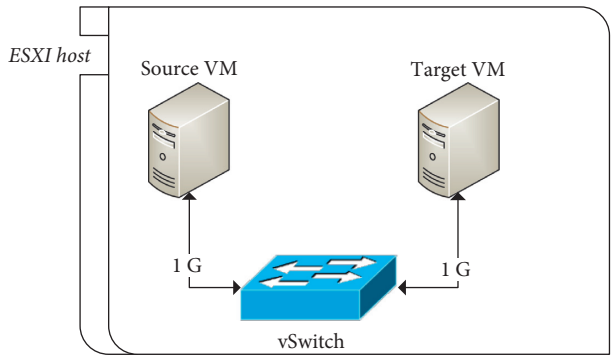

Figure 1: Experiment network and topology.

stage is performing the target's operating system profiling to obtain insight into the effect of the generated UDP traffic on ILNPv6 stack performance.

The performed eleven tests started with Test0 and ended with Test10. In Test0, only the first type of UDP packets was sent from source to target. This test was used to establish a baseline performance for each prototype. The percent of the second type of UDP datagram increased gradually from 0% in Test0 to 34% in Test10. In addition, the average number of UDP packets emitted by the source increased as tests progressed, starting with 1280 packets per second for Test0, and ending with 1940 packets per second at Test10 as shown in Figure 3.

During the tests of the first stage, *trace-cmd was* used to trace the execution time of the *udpv6_sendmsg* transport layer kernel function with *function_graph* tracer. This tracer is able to trace both the entry and exit of the traced function and its descendants. In addition, NIC's driver statistics were collected.

In the second stage tests, three levels of profiling of the target were performed. The first level is system-wide to monitor the overall system load. The second is also system-wide and sheds light on load distribution between the target system's three components: the user space, kernel space, and NIC's driver during tests. The third level of the profiling was done within the kernel. To clarify the distribution of loads among the kernel's functions, we divided these functions into two groups: the first includes just ILCC's function, and the rest of kernel's functions are located in the second group.

4.5. Test Setup for Network Layer. Using the same previous manner, the network layer tests are conducted in two stages, and each stage consists of eleven tests. These tests are carried out to demonstrate the security risk and performance deterioration resulting from adopting new ILCC entries in ILNP capable host only based on a pure network layer decision.

In the test scenario, the target is idle without any running service. When a host is in this state and receives UDP packets, as stated in RFC1122 [20], it should generate an Internet control message protocol (ICMP) destination unreachable messages with code 3 (port unreachable), because the designated transport protocol (UDP in our case) is unable to demultiplex the datagram and has no protocol mechanism to inform the sender. Taking the previous host requirement into consideration, this evaluation investigates how both prototypes handle the second type of UDP datagrams that may be generated by a malicious or bugged host and study the impact of this kind of traffic.

The previously described traffic generator is used for network layer tests. Figure 3 shows the generated traffic volumes for both types. In network layer tests, only the second type of UDP packets is generated. Based on that, Test0 represents the idle state of the source and target, where no packets destined to the target are sent. Starting from Test1, the generated traffic rate is 65 packets per second all over the test. After that, the number of sent packets increases to 125 packets per second at Test2. The gradual increase in the number of sent packets continues as the tests progress until reaching 660 packets per second at Test10.

At the first stage tests, the same toolset presented in previous section is used to monitor the internal and external behavior of the ILNPv6 network stack. During the conducted tests, *trace-cmd was used* to trace the execution time of the *ip6_input_finish* network layer kernel function with *function_graph* tracer. As mentioned earlier, this tracer is able to trace both entry and exit of the traced function and its descendants. This function was chosen because it is the last IP layer function in the Linux kernel receiving path, and it parses the packet's destination options extension header that includes the nonce option. Moreover, both prototypes process nonce option and perform ILCC look-up and housekeeping at it.

While the tests of the second stage are dedicated to perform a profiling of the target's kernel, the profiling is

Table 2: Features of testbed hardware and software.

	Source VM	Target VM
CPU	Intel(R) Xeon(R) silver 4110 CPU @ 2.10 GHz (4 virtual CPUs)	Intel(R) Xeon(R) silver 4110 CPU @ 2.10 GHz (4 virtual CPUs)
RAM	4 GB DDR4	4 GB DDR4
Operating system	Ubuntu 18.04.1 LTS	Debian GNU/Linux 9
OS kernel	Linux 4.15.0–45-generic	A modified Linux kernel v4.9 or v4.4
Ethernet adapter	Intel 82545EM Gigabit Ethernet controller	Intel 82545EM Gigabit Ethernet controller
Driver	vmxnet3	vmxnet3

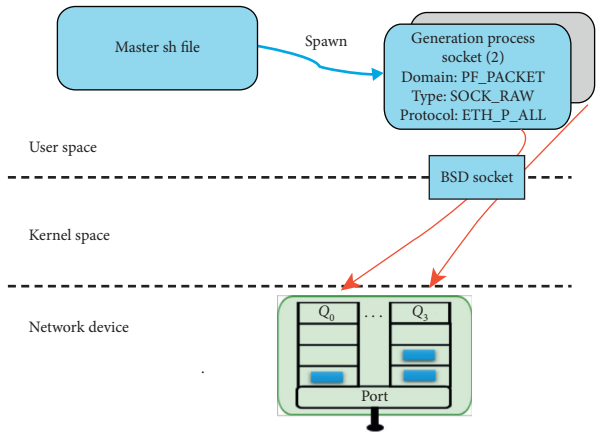

Figure 2: Traffic generator architecture.

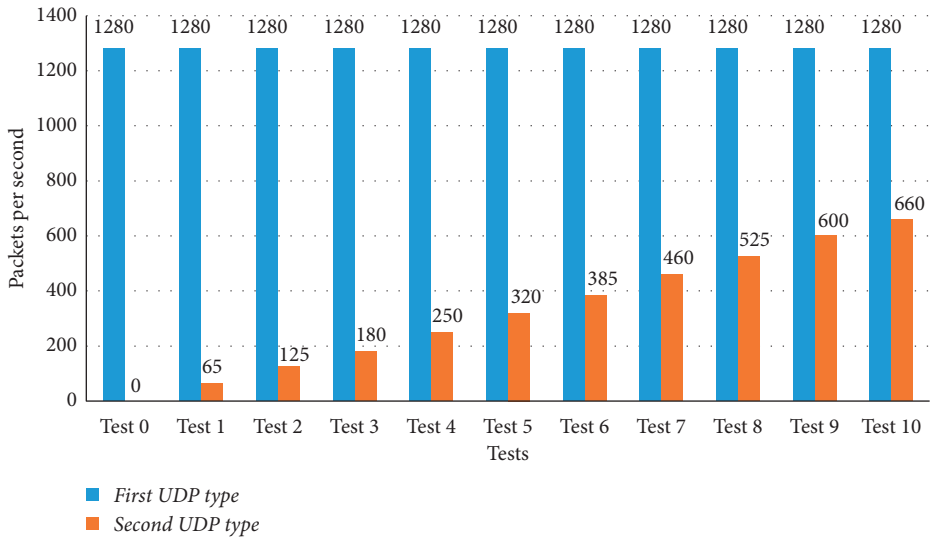

Figure 3: Generated traffic volume.

done to illustrate the distribution of loads among the kernel's functions. We divide these functions into two groups: the first includes just ILCC's functions and the rest of the kernel's functions are located in the second group.

4.6. Graphs. All graphs represent measurements for different aspects of the target in two test scenarios. The first scenario is the transport layer scenario, and the second is the network layer scenario. Every scenario includes two stages with eleven tests in each. In each scenario, the same eleven tests were conducted in every stage. Every test in each stage was repeated ten times to obtain more accurate results and exclude marginal values. After that, the average values were obtained by aggregating the results. Based on that, the graphed measurements of each test aspect are the average values of ten iterations of the same test unless otherwise mentioned. The standard deviation is shown by error bars.

5. Results and Analysis

Here, we present the results obtained from collecting external and internal target behaviors during the conducted tests. The used performance metrics are as follows.

(i) ILCC size: the size of ILCC at the target during tests was measured; values close to zero are better; zero is ideal.

(ii) Kernel latency: the delay added by the kernel, at both the network and transport layers, was measured at the target; lower values are better.

(iii) Packets serving and dropping rates: the number of served and dropped packets at the target was measured; serving rate values close to the offered loads are better; while lower dropping rates are better, zero is ideal.

Moreover, the target system profile is presented to give insights about the effects of the generated traffic.

5.1. ILCC Size. The following section presents ILCC size measurements for the conducted tests, as a first step towards understanding its effects on ILNP stack performance.

5.1.1. Transport Layer Tests. The kernel with policy kept a fixed ILCC size of zero entry overall tests. The reason, according to the adopted policy, is that as the data exchange between the two ends did not include any connected end, there was no need to create a record within ILCC to serve this exchange, while, in the kernel without policy, ILCC maintained a fixed size of one entry at Test0. However, in the rest of the tests, and with the presence of the second type of UDP datagrams, the ILCC size increased linearly with the number of emulated correspondents. Figure 4 shows ILCC size during some tests for the kernel without policy. As shown in Figure 4, the target developed an upper threshold for ILCC size. The first test that reached this threshold was Test7, and also, all subsequent tests did so. It is worth mentioning that Figure 4 does not include the ILCC growth curves for all conducted tests to maintain clarity of the figure. To further investigate the upper threshold of ILCC, the same tests were conducted on different target hardware specifications. The observation is that the upper threshold of ILCC size is strongly related to available hardware resources, and different hardware resources have different thresholds.

5.1.2. Network Layer Tests. The UDP packets of the second type do not have any effect on ILCC size at the kernel with policy. This kernel preserves an empty ILCC over all conducted tests since no transport layer sockets are waiting for those packets. Consequently, the decision of the policy is not to add ILCC entries for those packets.

Although the target was idle, the kernel without policy maintained an empty ILCC at Test0. However, in the rest of the tests, and with the presence of the second type of UDP datagrams, the ILCC size increased linearly with the number of emulated correspondents despite the lacking of UDP sockets to serve those packets. Consistent with the results in the previous section, target had developed an upper threshold for ILCC size about 15 k entries. Figure 5 shows the growth of ILCC in the kernel without policy over some of the conducted tests. As shown, this threshold was reached in Test5 and all subsequent tests.

5.2. Kernel Latency. Each network stack has a latency impact at the host level. In ILNP stack, although ILCC is a fundamental component, the stack, as an optimization, should avoid unnecessary delays within ILCC responsibility; that is, for ILCC size and lookups cost, the ILNP network stack should carefully decide which entries to adopt in ILCC to reduce latency, while keeping the ability to fulfill its objectives. To further understand ILCC size effects on ILNP stack performance, the following section presents the latency introduced by both prototypes for the conducted tests.

5.2.1. Transport Layer Tests. Figure 6 shows how the average latency changed with the increase of the second UDP datagrams for both kernels. In the performance baseline test (Test0), both kernels maintained the same average of latency. In later tests, the kernel with policy almost preserved the same latency. Moreover, the latency for this kernel was normally distributed at each test, with a very low standard deviation. However, the kernel without policy showed an increased latency as tests progressed. Starting from Test1, this kernel had a long-tail latency probability distribution with a considerable standard deviation.

5.2.2. Network Layer Tests. Figure 7 shows how the average latency changed with the progress of tests for both kernels. Although the target was idle, in network layer tests, the observation was that the second type of UDP traffic severely affected the latency at the network layer of the kernel without policy. As expected, at the performance baseline test (Test0), both kernels induced zero latency at the network layer. In subsequent tests, the kernel with policy almost preserved the same latency average, with normally distributed latency at each test and with a very low standard deviation. The observed function's average execution time ranged between 7 μseconds and 13.86 μseconds during all tests. For the kernel without policy, we observed the same previous behavior of long-tail latency probability distribution with a considerable standard deviation. In this kernel, the average function's execution time started at 267.75 μseconds in Test1 and ended at 1495.78 μseconds for Test10.

5.3. Packets Serving and Dropping Rates. Since latency at host is strongly associated with packets serving and dropping rates, packets serving and dropping rates are presented in the following section.

5.3.1. Transport Layer Tests. As a result of increased latency at the kernel without policy, it exhibited a decreasing packet serving rate as tests processed. As shown in Figure 8, at this

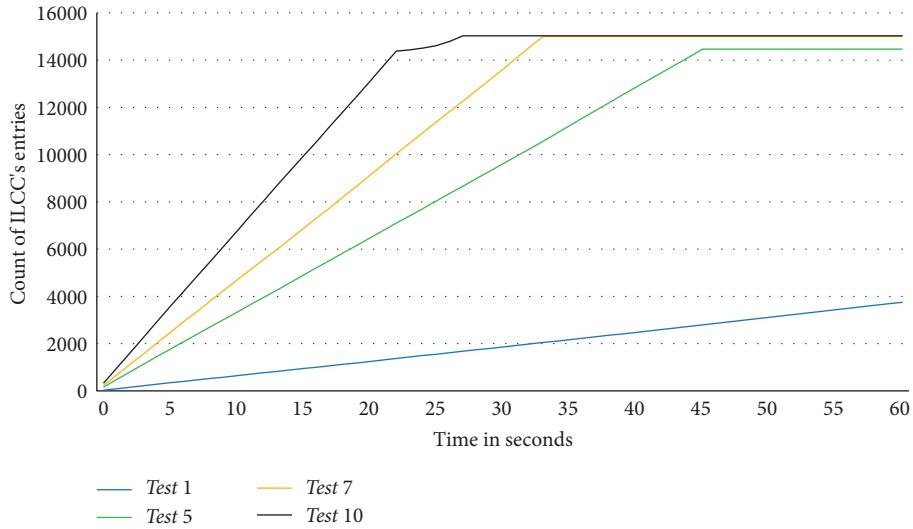

Figure 4: ILCC growth over transport layer tests.

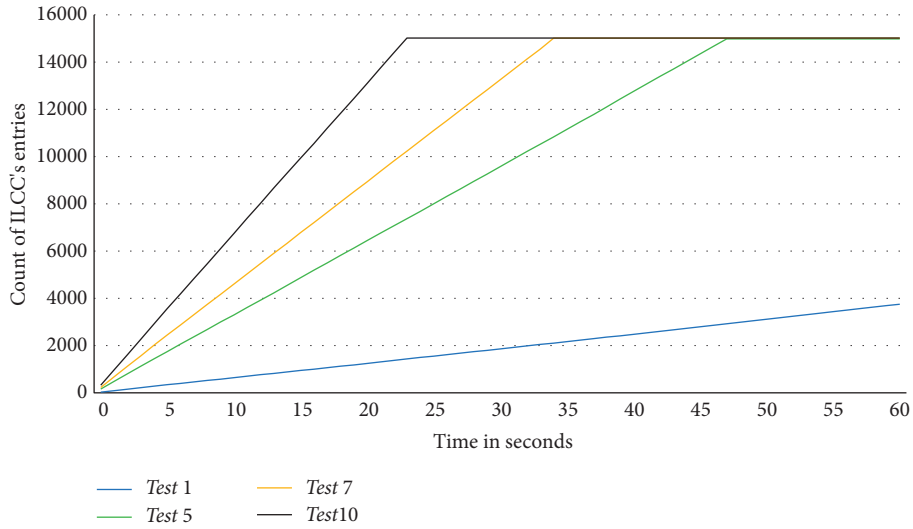

Figure 5: ILCC growth over network layer tests.

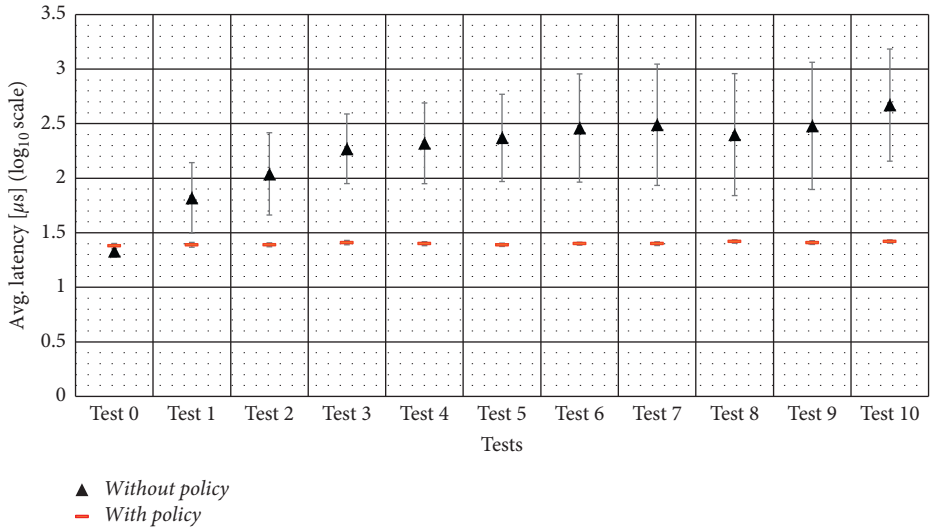

Figure 6: Average of transport layer latency.

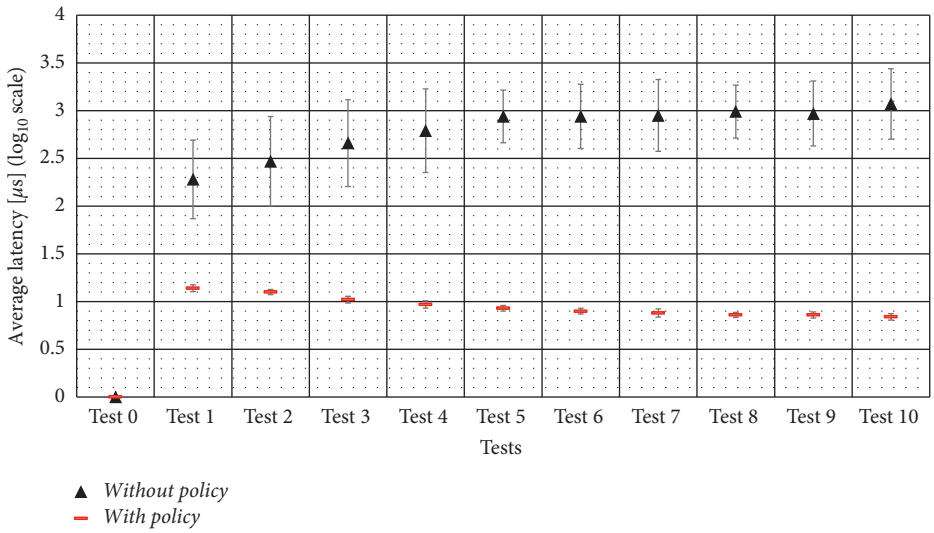

Figure 7: Average of network layer latency.

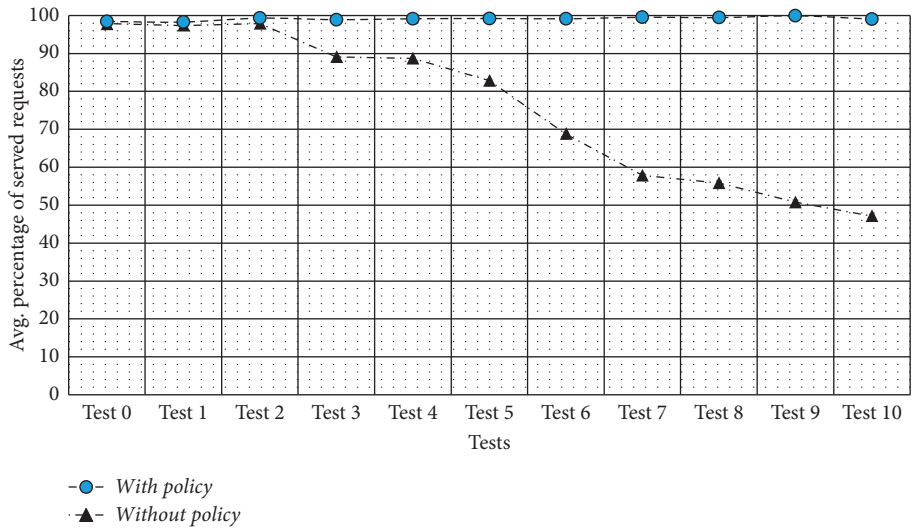

Figure 8: Average percentage of served echo requests.

kernel, the average percentage of serviced echo requests declined to less than 50% of total echo requests at both Test9 and Test10. Other than the previously mentioned, the kernel with policy maintained a steady service rate equal to the echo request arrival rate, which was consistent with the latency constancy exhibited by it.

Packets dropping rate was another aspect under monitoring in the evaluation. At each test, the number of dropped packets was obtained from NIC's driver. The reason behind this drop, as reported by the NIC, was that its ring buffers were full, so there were no ready descriptors for the new packets. This indicates that, during tests, at some points, the packets' arrival rate at NIC was greater than the rate at which the kernel reinitialized packet descriptors. This behavior was compatible with previous measures of transport layer latency and packet service rate.

Figure 9 shows the average percentage of dropped packets over conducted tests for both kernels. The kernel with policy maintained a constant service rate with no packets drop at all tests, while, in the kernel without policy, as packets' service rate decreased over tests, it experienced an increased packets' drop. The average percentage of dropped packets exceeded 50% starting from Test7. Figure 10 is used to clarify the observed drop pattern during some tests for the kernel without policy. The graphed tests were chosen as representative examples, as other tests followed a similar pattern.

5.3.2. Network Layer Tests. As the target was idle at network tests, there was no service rate to monitor, but only a drop rate. Figure 11 shows the average percentage of dropped packets over conducted tests for both kernels. The number of dropped packets was obtained from NIC's driver for each test. The cause for packet drop was full-ring buffers as in the previous section.

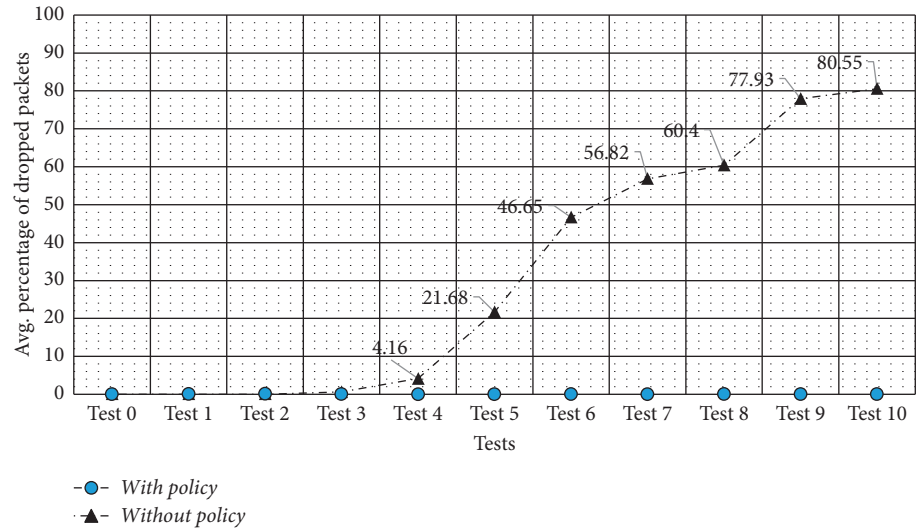

Figure 9: Average percentage of dropped packets.

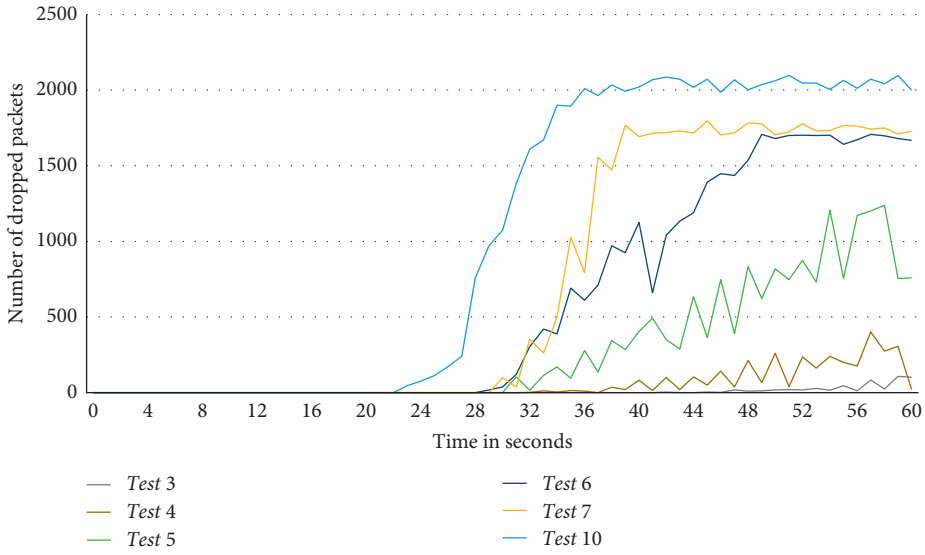

Figure 10: Number of dropped packets by the kernel without policy.

This behavior was expected based on the previous measures of the network layer latency. As shown in the figure, the kernel with policy had no packets drop at all tests, whereas the kernel without policy exhibited the same previous behavior of dropping packets. The average percentage of dropped packets exceeded 25% starting from Test8.

Figure 12 is used to render the noticed model of packet drop during some tests of the kernel without policy. Within Test9 and Test10, the target dropped all the packets destined to it and became unreachable from the fortieth second although it was not running any service. The plotted tests were chosen as representative examples, as other tests followed a similar pattern.

5.4. Target's System Profiling. To find out the main cause of performance degradation at the target, a profiling of the target's kernel at several levels was performed. The results are as follows.

5.4.1. Transport Layer Tests. The following figures are presented to indicate the source of load that caused the target's performance deterioration at the transport layer and the whole network stack, specifically when the target ran the kernel without policy. Figure 13 shows the average percentage of the target load for both kernels for only the first seventh tests. In the later tests, the measurements were not obtained as the target suffered from massive scheduling

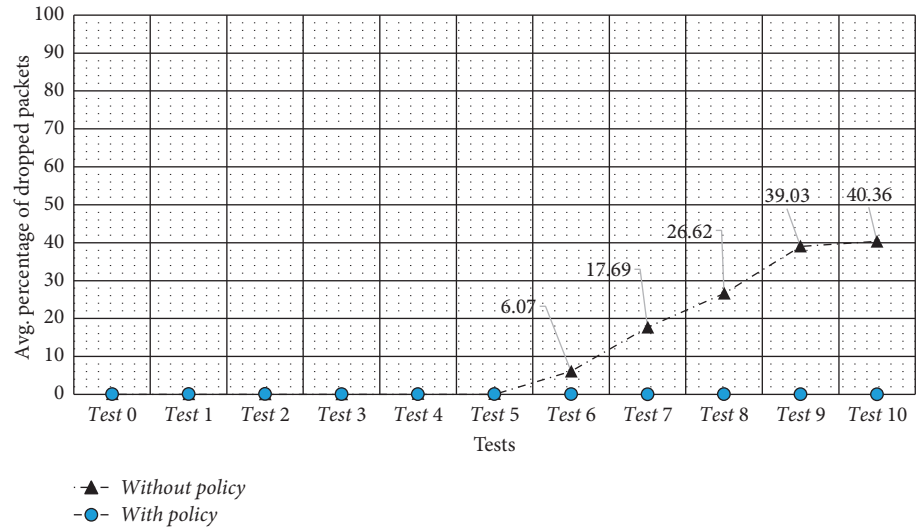

Figure 11: Average percentage of dropped packets.

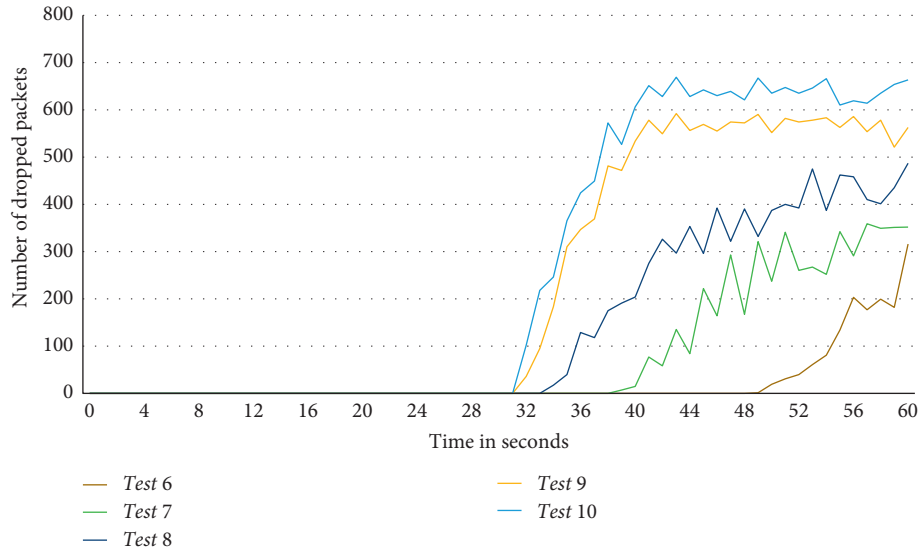

Figure 12: Number of dropped packets by the kernel without policy.

problems that prevented the monitoring thread from performing its job. As shown in the figure, the target had a gradual load increase as tests were processed when it used the kernel without policy.

In Figure 14, the results of the second level of the target profiling are presented. The loads resulting from the user space and driver were excluded because their values were stable for all kernels with policy tests and marginal compared with the load caused by the kernel without policy from Test3 onwards. As shown in Figure 14, the kernel without policy was the source of overload in the target, and again, it had an increasing load that exceeded 90% of the total target load starting from Test6.

The results of the third level of the target profiling are presented in Figure 15. This figure did not present the load caused by the second group (all kernel functions except ILCC's functions) as it was insignificant. While the load of the first group of functions did not exceed 3% of the entire kernel without policy load in the baseline test, a continuous increase in this percentage was noticed in the later tests, and starting from Test6, this percentage exceeded 90%.

Based on the previous results, it is clear that ILCC size has a significant impact on the performance of the transport layer and thus on all network stack. In the testbed, the impact was evident and led to poor performance at the kernel without policy. Moreover, these results are consistent with the conclusion in [15]. The kernel with policy has successfully minimized the impact on performance by effectively controlling ILCC size.

5.4.2. Network Layer Tests. Figure 16 shows the loads caused by the ILCC functions for both kernels without user space and driver, as their load was marginal and in order to

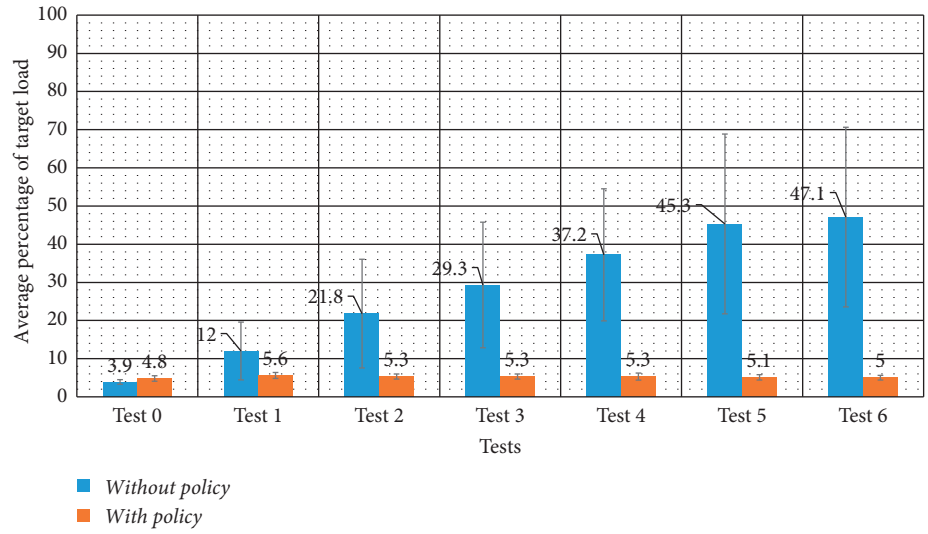

Figure 13: Average percentage of the target load.

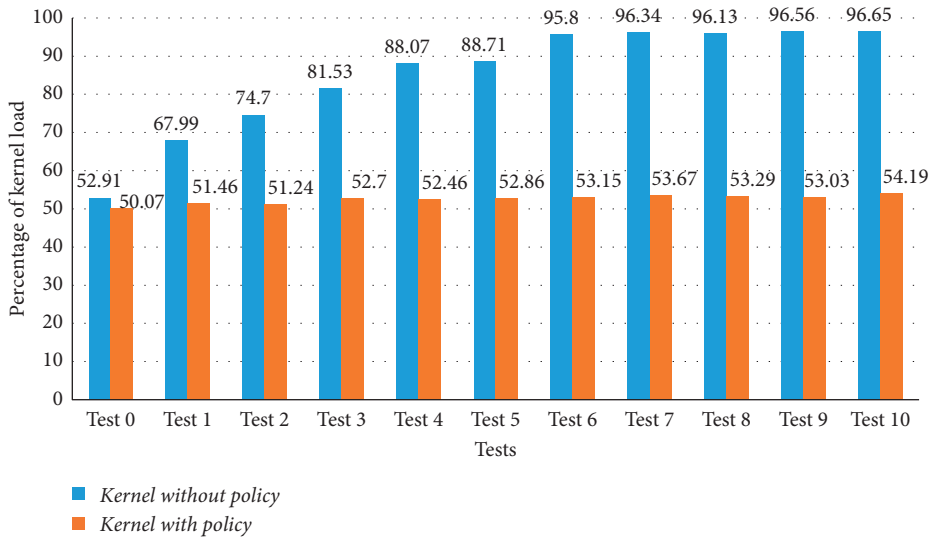

Figure 14: Percentage of kernel load.

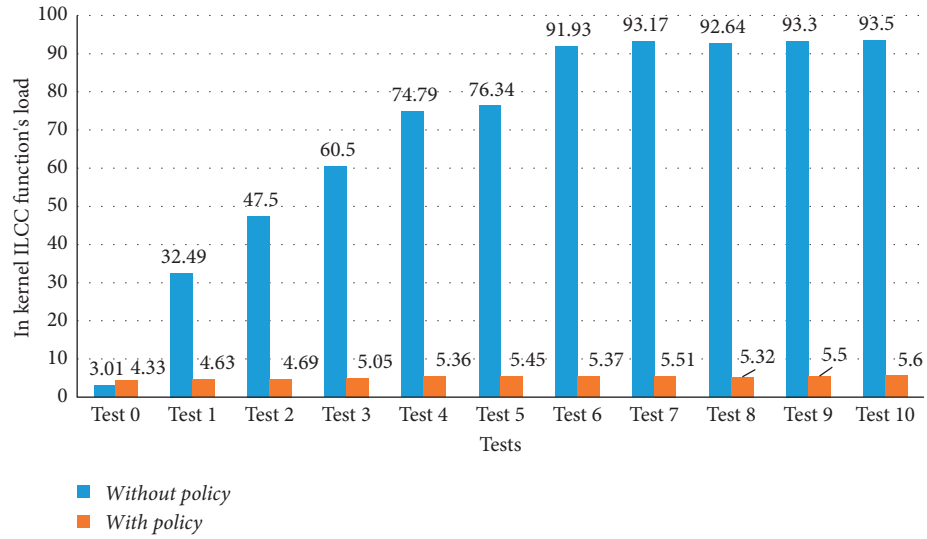

Figure 15: In kernel ILCC function's load.

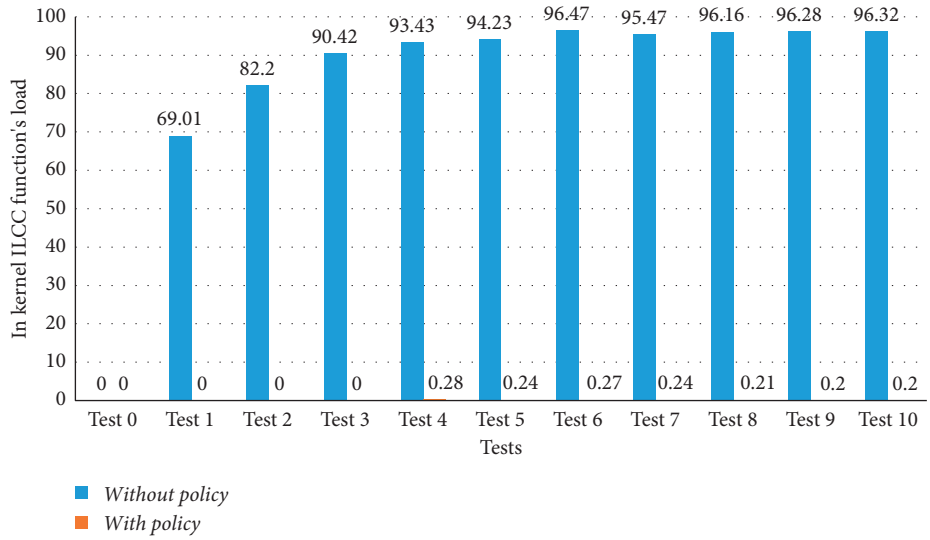

Figure 16: In kernel ILCC function's load.

maintain the figure's readability. The ILCC functions in with policy kernel kept a steady load level between 4.3% and 5.6% of the total kernel load overall tests, whereas the ILCC functions in the without policy kernel had an increasing load level with tests advancement. From Test3 onwards, these functions load represented more than 90% of the entire total kernel load. Therefore, almost all the target's processing resources were performing the kernel's tasks, letting no room to serve tasks that belong to both user space and NIC driver.

From these results, it is clear that although the target did not run any service, due to its adoption of new records within the ILCC based on a pure network layer decision, and not setting any restrictions on the growth of ILCC, it became vulnerable to DoS attack, taking into consideration that in our case the attack took less than 60 seconds.

Based on the previous results, it is evident that the kernel with policy outperforms the kernel without policy in terms of ILCC growth, the delay added by the kernel network stack at network layer, packet drop, and overhead resulted from the kernel's function that deals with ILCC operations.

6. Conclusion

In this work, the effects of ILCC on the ILNPv6 stack are studied in detail at different network stack layers. We conduct a performance evaluation of two ILNPv6 prototype implementations and illustrate the way they handle different types of UDP traffic. One of the key differences between these prototypes is the policy used to manage ILCC. The first prototype was introduced by the University of St Andrews, while the other was developed by us.

In the evaluation, ILCC size, delay added by the kernel network stack at both network and transport layers, packet service and drop rates, and overhead resulting from the kernel's function that deals with ILCC operations are measured. The results show that the St Andrews's prototype experience uncontrolled ILCC growth as a normal result for its ILCC entries adoption policy. Consequently, this prototype suffers from a severe performance degradation in terms of network stack latency, service rate, drop rate, and system loads. Moreover, the host running this prototype is vulnerable to DoS attacks, while our prototype effectively controls ILCC size by carefully deciding which entries to adopt in ILCC, and thus, it outperforms the St Andrews's prototype in all measured aspects.

Moreover, the results show that the introduced policy is a promising new technique to improve latency and performance for UDP traffic over ILNPv6 via avoiding unnecessary delays within ILCC responsibility. Therefore, we recommend adapting this policy within the protocol description and not leaving ILCC management as an engineering consideration up to the implementer.

However, this is the first work that presents a comprehensive performance evaluation of our prototype and its ILCC policy, and they should be subject to further studies with various scenarios and integration with other transport layer protocols. As future work, we plan to study ILCC growth effects on transmission control protocol (TCP) and host-based mobility.

Data Availability

No data were used to support this study.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] D. Meyer, L. Zhang, and K. Fall, "Report from the IAB workshop on routing and addressing," 2007.

[2] G. Huston, "Analyzing the internet's BGP routing table," *Internet Protocol Journal*, vol. 4, no. 1, pp. 2–15, 2001.

[3] F. Liu, C. Gong, R. Tu, and H. Tang, "Identifier and locator mapping service research," *Journal of Physics: Conference Series*, vol. 1693, no. 1, p. 12188, 2020.
[4] F. Liu, C. Gong, R. Tu, and H. Tang, "Identifier and locator separation based site multi-homing path failure recovery," *Journal of Physics: Conference Series*, vol. 1631, no. 1, p. 12096, 2020.
[5] A. Saleh and J. Simmons, "Technology and architecture to enable the explosive growth of the internet," *IEEE Communications Magazine*, vol. 49, no. 1, pp. 126–132, 2011.
[6] B. Li and J. Wang, "An identifier and locator decoupled multicast approach (ILDM) based on ICN," *Applied Sciences*, vol. 11, no. 2, p. 578, 2021.
[7] C.-H. Lee, J. S. Park, and J. G. Shon, "An SDN-based distributed identifier locator separation scheme for IoT networks," *Advances in Computer Science and Ubiquitous Computing*, vol. 5, pp. 349–355, 2021.
[8] B. Feng, H. Zhang, H. Zhou, and S. Yu, "Locator/identifier split networking: a promising future Internet architecture," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2927–2948, 2017.
[9] W. Ramirez, X. Masip-Bruin, M. Yannuzzi, R. Serral-Gracia, A. Martinez, and M. S. Siddiqui, "A survey and taxonomy of ID/locator split architectures," *Computer Networks*, vol. 60, pp. 13–33, 2014.
[10] M. Komu, M. Sethi, and N. Beijar, "A survey of identifier-locator split addressing architectures," *Computer Science Review*, vol. 17, pp. 25–42, 2015.
[11] R. J. Atkinson and S. N. Bhatti, "Identifier-Locator network protocol (ILNP) architectural description," 2012.
[12] R. Atkinson and S. N. Bhatti, "Identifier-locator network protocol (ILNP) engineering considerations," 2012.
[13] ILNP software version: ilnp-public-1 | ilnp-public-1." 2021, https://ilnp.github.io/ilnp-public-1/.
[14] M. Kadi, M. Suleiman, and S. Jammoul, "Caching in ILNP, identifier locator communication cache (ILCC) implementation using three-way handshake," *International Journal of Engineering Research and Technology*, vol. 10, no. 1, pp. 107–113, 2021.
[15] N. Abid, P. Bertin, and J.-M. Bonnin, "A comparative cost analysis of identifier/locator split approaches," in *Proceedings of the 2014 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 2946–2951, Istanbul, Turkey, April 2014.
[16] D. Phoomikiattisak and S. N. Bhatti, "Mobility as a first class function," in *Proceedings of the 2015 IEEE 11th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pp. 850–859, Abu Dhabi, United Arab Emirates, October 2015.
[17] C. Benvenuti, *Understanding Linux Network Internals*, O'Relly Media," Inc., Sebastopol, CA, USA, 2005.
[18] trace-cmd(1) - Linux man page, 2021, https://linux.die.net/man/1/trace-cmd.
[19] perf(1) - Linux manual page.2021, https://man7.org/linux/man-pages/man1/perf.1.html.
[20] R. Braden, "Requirements for internet hosts--communication layers," 1989.