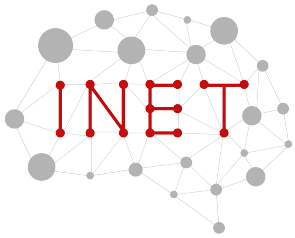

Bachelorthesis

Evaluating Dual-Stack against NAT64 deployment schemes with DNS64 and CLAT

Wodke, Daniel Jin

21. Mai 2025

Gutachter: Prof. Dr. Stefan Schmid
Prof. Dr. Stefan Tai
Betreuerin: Max Franke and Dr. Philipp Tiesel
Matrikelnr.: 456675

Technische Universität Berlin
Faklutät IV - Elektrotechnik und Informatik
Institut für Telekommunikationssysteme
Fachgebiet Intelligent Networks

Eidesstattliche Versicherung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Berlin, den Date

_____

Wodke, Daniel Jin

Zusammenfassung

Weit hinten, hinter den Wortbergen, fern der Länder Vokalien und Konsonantien leben die Blindtexte. Abgeschieden wohnen Sie in Buchstabhausen an der Küste des Semantik, eines großen Sprachozeans. Ein kleines Bächlein namens Duden fließt durch ihren Ort und versorgt sie mit den nötigen Regelialien. Es ist ein paradiesmatisches Land, in dem einem gebratene Satzteile in den Mund fliegen. Nicht einmal von der allmächtigen Interpunktion werden die Blindtexte beherrscht – ein geradezu unorthographisches Leben. Eines Tages aber beschloß eine kleine Zeile Blindtext, ihr Name war Lorem Ipsum, hinaus zu gehen in die weite Grammatik. Der große Oxmox riet ihr davon ab, da es dort wimmele von bösen Kommata, wilden Fragezeichen und hinterhältigen Semikoli, doch das Blindtextchen ließ sich nicht beirren. Es packte seine sieben Versalien, schob sich sein Initial in den Gürtel und machte sich auf den Weg. Als es die ersten Hügel des Kursivgebirges erklommen hatte, warf es einen letzten Blick zurück auf die Skyline seiner Heimatstadt Buchstabhausen, die Headline von Alphabetdorf und die Subline seiner eigenen Straße, der Zeilengasse. Wehmütig lief ihm eine rethorische Frage über die Wange, dann setzte es seinen Weg fort. Unterwegs traf es eine Copy. Die Copy warnte das Blindtextchen, da, wo sie herkäme wäre sie zigmal umgeschrieben worden und alles, was von ihrem Ursprung noch übrig wäre, sei das Wort "und" und das Blindtextchen solle umkehren und wieder in sein eigenes, sicheres Land zurückkehren.

Abstract

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi.

Contents

Chapter 1

Introduction

The transition from IPv4 to IPv6 has shifted from a far concern to an immediate challenge. With IPv4 address exhaustion and rising costs for public IPv4 addresses in cloud environments, organizations face increasing pressure to adopt IPv6. However, complete migration remains complicated by the reality that many applications and services still depend on IPv4 connectivity, whether due to legacy dependencies or third-party integrations. Two primary approaches have emerged. Dual-Stack maintains both IPv4 and IPv6 connectivity, offering wide compatibility but increasing operational overhead and continuous IPv4 allocation. In contrast, IPv6-only with NAT64/DNS64 and CLAT (464XLAT) enables a smaller IPv4 footprint while maintaining reachability to IPv4 services via translation. While this has worked well in mobile networks, the practical trade-offs for enterprise and cloud workloads remain a subject of debate. Industry collaboration This thesis was conducted in close cooperation with SAP SE and was supervised from the industry side by Dr. Philip Tiesel. This collaboration ensured that the research questions addressed are directly aligned with the real-world challenges faced by large-scale platform engineering teams. The primary motivation comes from the architectural decisions required by projects like SAP Gardener, an open-source initiative for managing Kubernetes clusters at scale. Within SAP's Gardeners ecosystem, teams decide between Dual-Stack clusters and IPv6-only clusters. The choice has implications for cost, operational complexity, and performance.

1.1 Problem Statement

While the conceptual trade-offs between Dual-Stack and IPv6-only with 464XLAT are understood, there is a lack of publicly available, empirical data that quantifies the performance overhead of common, open-source translation implementations. Network architects and platform operators are forced to make long-term architec-

tural decisions based on assumptions or data from differing environments, such as mobile networks. The central problem this thesis addresses is: Does a CLAT implementation create performance disadvantages compared to native Dual-Stack connectivity that would justify the continued reliance on IPv4 infrastructure? The problem is increased by the fact that performance data from mobile networks, where 464XLAT has seen widespread adoption, may not be directly applicable to data center or cloud environments. Mobile networks operate under different constraints, with specialized hardware and traffic patterns that differ significantly from the high-throughput, low-latency requirements common in enterprise applications. Moreover, the choice between translation mechanisms is not only about raw performance numbers. Different implementations operate at different layers of the network stack—some in kernel space, others in userspace—each with implications for CPU utilization and integration complexity. Without concrete performance data, teams cannot make informed decisions about which approach best fits their specific use case. The practical impact of this knowledge gap extends beyond technical considerations. In cloud environments where IPv4 addresses cause direct costs, the performance penalty of translation mechanisms directly influences the economic viability of IPv6-only deployments. If the overhead is minimal, organizations can confidently transition to IPv6-only architectures and reduce their IPv4 footprint. However, if translation introduces significant performance degradation, the operational costs may outweigh the savings from reduced IPv4 usage.

1.2 Objectives and Scope

The objective of this thesis is to produce empirical evidence on the performance trade-offs between native Dual-Stack connectivity and IPv6-only deployments that rely on NAT64/DNS64 with a client-side translator (CLAT) as defined by 464XLAT. In practical terms, the work aims to answer whether a CLAT-based path introduces a measurable performance disadvantage that would argue against replacing Dual-Stack in enterprise and cloud contexts. To address this question, the thesis focuses on quantifying the overhead of translation relative to a Dual-Stack baseline using two fundamental network metrics: throughput and round-trip time. Throughput is measured with iperf, and RTT is measured with ping, providing a straightforward and reproducible basis for comparison. The evaluation concentrates on open-source software components that reflect different implementation strategies in the network stack: Jool as a stateful kernel-space NAT64, Tayga as a stateless user-space NAT64, and Tundra as a stateless user-space NAT64. The measurements are carried out across three environments: an AWS cloud setup, a single Ubuntu machine where translators run in separate Linux network namespaces on one client host, and a two-host Ubuntu setup connected via Ethernet with the iperf

server on the second machine. The scope of the work is intentionally narrow to keep the measurements focused and comparable. The metrics are limited to TCP throughput and ICMP RTT. Tail latency, per-packet loss under sustained load, and application-level behaviors are not evaluated. Likewise, the study does not include a systematic analysis of CPU utilization, power consumption, or a cost model. DNS resolution performance itself is not measured. The translators under test are software components typical of general Linux systems. Other transition mechanisms such as SIIT-DC are not considered, since the goal is to compare Dual-Stack with the specific IPv6-only approach built around NAT64/DNS64 and CLAT. Baselines and roles are defined as follows. The Dual-Stack baseline represents native IPv4/IPv6 connectivity without any translation. The CLAT-based path represents an IPv6-only client that uses a local NAT46 function (CLAT). All translators are placed in separate network namespaces to ensure isolation and to make it possible to observe effects to the corresponding implementation. This layout is kept consistent across the three environments. Several limitations follow from this design. Because only iperf and ping are used, the results primarily capture bulk data throughput and a basic latency profile. They do not fully characterize behavior under bursty traffic patterns, short flows, or high packet rate scenarios. The choice of software-only translators on Linux narrows the conclusions to deployments that resemble this setup; organizations using hardware offload may see different results. Moreover, the study does not attempt to evaluate security aspects such as application-level gateways or resilience under failures. While these topics are important in practice, including them would shift the focus from the central performance question. A discussion of limitations and their implications is provided in Chapter 6 - Conclusion. The work is designed to be reproducible. Configurations, scripts, and environment details are documented and collected in the appendix so that others can repeat the measurements or extend them with additional metrics. Chapter 4 - Experiment Design details the setups and methodology, including the namespace configuration on the single-host setup and the role of the iperf server on the second host in the dual Ubuntu environment.

1.3 Thesis Structure

This thesis proceeds from motivation and context to reproducible measurement and evidence-based conclusions in a linear fashion. Following the introduction, which frames the problem and outlines the industry motivation in cooperation with SAP, Chapter 2 provides the technical background required to interpret the results. It revisits IPv4 exhaustion and the ongoing transition to IPv6, contrasts Dual-Stack with IPv6-only strategies, and introduces the principles of NAT64/DNS64 together with the role of client-side translation in 464XLAT to explain how IPv4-

only endpoints remain reachable from IPv6-only clients. The chapter closes with a description of the software components used in the experiments—Jool, Tayga and Tundra. Chapter 3 presents related work and identifies the specific gap this study addresses by reviewing key studies on IPv6 transition mechanisms, summarizing reported performance characteristics, before pinpointing the lack of publicly available empirical data for the selected open-source implementations in the environments considered here. Chapter 4 then explains the experiment design by introducing the three test environments—an AWS setup, a single Ubuntu host with translators running in separate network namespaces, and a dual-host Ubuntu setup connected via Ethernet with the iperf server on the second machine—documenting how Tayga, Tundra, and Jool are set up, how namespaces and routing are configured, and how measurements are taken with iperf for throughput and ping for RTT, and discussing practical challenges encountered during setup, such as clocksource differences and their effect on timing, together with the steps taken to mitigate them. Chapter 5 presents the test results and evaluation, showing throughput and RTT outcomes per environment, analyzing the translators against the Dual-Stack baseline, synthesizing findings across environments and providing a summary comparison of Jool, Tayga, and Tundra. Finally, Chapter 6 concludes the thesis by summarizing the main findings with respect to the central question, outlining practical implications engineering teams considering IPv6-only with NAT64/DNS64 and CLAT, reflecting on limitations arising from the chosen metrics, software focus, and environments, and suggesting directions for future work, while the document ends with references and an appendix containing configuration files and scripts, and result tables to support reproducibility.

Chapter 2

Background

To understand the choices we're making for the internet today, we have to look back at a long-standing problem: we've run out of the original internet addresses (IPv4). This has forced us into a decades-long transition to a new system (IPv6), where parts of the internet will use only the new addresses. The core challenge is well-known: the old and new systems don't speak the same language, yet both have to work together on a global scale. At the same time, we're trying to balance running out of old addresses, increasing costs, and the constant risk of things breaking[1], [2].

2.1 IPv4 exhaustion and transition to IPv6

Internet use has expanded to critical infrastructure across consumer, enterprise, and public sectors. Services running at any time of the day and the increase of connected devices have driven steady growth in traffic and endpoints, making address management a central concern[1], [2]. IP addresses perform two fundamental roles: identifying endpoints and enabling packet delivery across networks, and uniqueness at global scale is essential[2]. IPv4, standardized in 1981-1983, provides a 32-bit address space of roughly 4.3 billion addresses[3].In practice, not all addresses are usable on the public Internet due to special-use and private allocations, and early design choices further reduced the effectively usable pool[1], [2], [4]. Management techniques such as CIDR and NAT slowed the pace of consumption but could not eliminate the finite limit[1]. Exhaustion means that the pool of unused IPv4 addresses has run out, not that IPv4 connections themselves have stopped working. The process started at the global level and then moved downward: IANA allocated its final IPv4 blocks on 3 February 2011, after that, each Regional Internet Registry (RIR) moved into its final phase: APNIC in April 2011, RIPE NCC in September 2012, LACNIC in June 2014 while AFRINIC held on the longest[2]. A global policy

passed on 6 May 2012 established mechanisms for the recovery and redistribution of returned IPv4 address space, yet scarcity has continued to persist[1]. The impact has been uneven across regions because historical allocations left some operators with far fewer addresses per user than others[2]. The shortage was unavoidable because demand kept rising: by 2014, about 2.9 billion people were online, with more than 200 million new users joining each year after 2010. Internet use jumped from less than 1

IPv6 was standardized in 1995 as the long-term successor, expanding addressing to 128 bits—on the order of 3.4×10^{38} unique addresses—and introducing protocol-level improvements aimed at routing scalability, mobility support, and operational security[1], [2], [5]. The allocation data highlights how abundant IPv6 is compared to IPv4, even when large IPv6 distributions are considered[1]. However, IPv4 and IPv6 don't work together on their own, so bridging mechanisms are needed while both remain in use[2]. Even though the technical benefits of IPv6 were clear, its adoption lagged behind the urgency created by IPv4 shortages. IPv6 allocations reported by the RIRs lagged behind user growth, with especially low adoption in some regions, like Africa. At the same time, user-side measurements, such as those from Google, showed less than 10Operators and policymakers have followed three main approaches: making better use of IPv4, allocating the remaining IPv4 space more efficiently and transitioning to IPv6[2]. On the technical side, NAT, especially carrier-grade NAT, conserves public IPv4 addresses by multiplexing many private hosts behind a smaller set of public addresses[6]. By around 2014, a measurable fraction of users were estimated to traverse CGN[7]. While NAT is effective at saving IP addresses, it can make end-to-end connectivity more complicated and add to operational complexity[8]. Dual stack (running IPv4 and IPv6 in parallel) is broadly available and will remain common for years, but it does not address the decline of IPv4[2]. Policy measures like smaller allocations and efforts to reclaim unused addresses have helped ease IP address scarcity somewhat, but they don't eliminate the need for IPv6[2]. Strategically this means that IPv4 and IPv6 will continue to coexist for a long time, connected through integration methods[1], [2]. This situation drives the focus of this thesis: as network operators decide whether to keep dual-stack setups or move to IPv6-only access with translation technologies (like NAT64/DNS64, often paired with CLAT on the client side), it's important to understand the performance trade-offs involved. The next section takes a look at the main transition mechanisms that form the basis of this comparison[1], [2].

2.2 IPv6 transition mechanisms

According to the IETF's classification of transition strategies, there are three main approaches: dual stack, tunneling, and translation**rfc2893**. Because this

thesis focuses on comparing dual stack and translation, tunneling will not be covered. Dual-Stack Architecture Dual stack is considered the main way to allow networks and hosts to gradually migrate, letting IPv4 and IPv6 run side by side**punithavathani2009ipv4**. In this model, a node runs two IP protocol stacks side by side: one for IPv4 and one for IPv6, following the early guidance outlined by the IETF**rfc2893**. This applies to both end systems, clients and servers, and to intermediate devices such as routers and gateways, which can natively handle and forward both IPv4 and IPv6 traffic when they support dual stack **punithavathani2009ipv4**. Traffic selection under dual stack is straightforward. Applications written for IPv4 use the IPv4 stack, and IPv6-capable applications use the IPv6 stack. When a packet is received, the node identifies the protocol using the Version field in the IP header. When sending a packet, the destination address decides which stack to use**rfc4213**. In practice, this works through DNS: A records point to IPv4 addresses, while AAAA records point to IPv6 addresses. The host's resolver and socket APIs then automatically choose the appropriate protocol stack**punithavathani2009ipv4**. From a deployment standpoint, dual stack is practical because most modern operating systems come with mature support for both IPv4 and IPv6**rfc7381**. The wide support for both protocols, combined with the fact that most applications run over their native IP without any modifications, helps explain why dual-stack has become the approach for running IPv4 and IPv6 together in real-world networks**punithavathani2009ipv4**. Its main goal is to ensure compatibility and enable a gradual transition: dual-stack lets operators roll out IPv6 while staying connected to the still-dominant IPv4 Internet, but it isn't meant to solve the issue of address shortages on its own [2]. Dual stack also has clear limits. It only allows direct communication between the same protocol: IPv6 to IPv6 and IPv4 to IPv4, so it doesn't bridge the two by itself. When communication across the protocols is needed during the transition, extra mechanisms like tunneling or translation have to be used**punithavathani2009ipv4**. Moreover, dual-stack doesn't actually save IPv4 addresses. Seeing it as a way to conserve them is misleading, if anything, it maintains high demand for IPv4, since every dual-stacked device still requires IPv4 connectivity**rfc4241** . Global IPv4 exhaustion has continued even with widespread dual-stack deployments, highlighting that dual-stack alone cannot solve the shortage [2]. Within this thesis, dual stack serves as the performance and behavior baseline when no translation is involved. As a result, any differences we observe compared to NAT64/DNS64 with CLAT can be seen as the extra cost of using translation-based approaches. At the same time, dual stack can't reduce reliance on IPv4 without additional tools, which is why the translation-based mechanisms in the next section are important **punithavathani2009ipv4**, [2].

NAT64/DNS64 Principles NAT64 with DNS64 has emerged as a practical response to the limits of dual stack in the current Internet**rfc7269**. Although

dual stack was initially seen as the main strategy for transitioning to IPv6, the ongoing shortage of IPv4 addresses and the uneven adoption of IPv6 make it increasingly difficult to keep every endpoint and network fully dual-stacked**rfc7269**. Translation technologies help bridge the gap, letting IPv6-only devices talk to IPv4-only systems without needing upgrades on every device or huge pools of public IPv4 addresses. In many networks NAT64 and DNS64 can either support or even take the place of dual stack, especially when IPv4 addresses are scarce or updating software everywhere isn't practical**6231295**. NAT64 was designed as the next step in the evolution of IP translation technologies. Earlier approaches like Stateless IP/ICMP Translation (SIIT) handled translation on a per-packet basis but needed a strict one-to-one mapping between IPv4 and IPv6 addresses, which limited scalability and flexibility**rfc6146**. NAT-PT tried to improve on this with stateful translation and a DNS application-layer gateway, but it ran into problems with complexity, fragile DNS interception, and synchronization issues, and was eventually deprecated**rfc4966**. NAT64 and DNS64 were created to overcome these limitations, offering clearer design, well-defined behavior, and explicit support for DNSSEC, while keeping the effective header translation techniques from SIIT**6231295**. Stateful NAT64 enables two-way translation between IPv6 and IPv4, allowing IPv6-only clients to connect to IPv4-only servers using TCP, UDP, or ICMP. It's designed for the long transition period where new networks and devices are primarily IPv6, while many services are still IPv4-only. When used with DNS64, neither the IPv6 client nor the IPv4 server needs any changes to communicate across the translator. Typically, a NAT64 device sits at the boundary between an IPv6-only network and the IPv4 Internet, with one interface facing each side. Packets from an IPv6 host going to an IPv4 server are routed to the NAT64, translated into IPv4, and sent onward. Replies from the server are translated back into IPv6, using the translation state established for that session**rfc6146**. In practice, NAT64 works through three main components. First, it translates IP and ICMP headers while making sure transport-layer checksums and communication rules are preserved across the IPv6-to-IPv4 boundary. Second, it uses algorithmic address embedding: an operator-assigned IPv6 prefix (Pref64::/n) is combined with the IPv4 address to create "IPv4-converted" IPv6 addresses that the translator can use. Third, its NAT behavior follows standard practices for TCP, UDP, and ICMP—like consistent endpoint mappings and typical filtering modes—so applications and network traversal techniques continue to work as expected. **rfc6146**, **6231295**. Address representation is a key part of how NAT64 works. It uses two address pools: an IPv6 pool, made up of a dedicated IPv6 prefix (Pref64::/n) that embeds IPv4 addresses, and an IPv4 pool—usually a small shared prefix—that represents IPv6 clients on the IPv4 Internet. The IPv6 addresses are formed by combining the Pref64::/n with the 32-bit IPv4 address, adding zeros if the prefix

is shorter than 96 bits. The Well-Known Prefix 64:ff9b::/96 provides a globally recognizable format, but operators can also use local prefixes. This Well-Known Prefix is especially useful when DNS64 and NAT64 are managed by different parties, as it separates the resolver from the translator while still ensuring packets reach the NAT64 device correctly. On the IPv4 side, the NAT64 translator usually maintains a small pool of public IPv4 addresses that are shared among many IPv6 clients. To make the most of this limited resource, NAT64 often uses address-and-port translation (A+P), allowing multiple IPv6 flows to share a single IPv4 address by assigning each flow a distinct range of ports**6231295**, **rfc6146**. NAT64 is stateful. Every time an IPv6 client starts a new connection to an IPv4 server, the translator creates a binding that links the IPv6 address, port, and protocol to an IPv4 address and port from its pool. Inside the translator, the Binding Information Base (BIB) keeps track of these mappings, connecting each internal IPv6 transport address to its assigned IPv4 address. These mappings can be reused across different destinations, enabling consistent endpoint-independent connections. A separate session table keeps track of each individual flow to allow more detailed filtering and maintain per-flow state when needed. Connection lifetimes follow BEHAVE guidelines: UDP bindings typically last a few minutes (around two minutes), while TCP bindings can last hours for established connections. The translator also detects TCP connection closures so it can quickly free up resources. Without an existing state in the translator, only IPv6 clients can start new connections to IPv4 servers. For connections initiated from IPv4 to IPv6, the translator needs recent outbound traffic, explicit static mappings, or support from the application itself. Since NAT64 uses endpoint-independent mapping, standard NAT traversal techniques like STUN, TURN, or ICE can still work across the translator. **6231295**, **rfc6146**. DNS64 complements NAT64 by synthesizing AAAA records from A records. When an IPv6-only client looks up a domain and the server only has an IPv4 (A) record, DNS64 steps in: it queries the A record, embeds the resulting IPv4 address into an IPv6 address using the operator's Pref64::/n, and returns that synthesized IPv6 address to the client**rfc6147**. DNS64 and NAT64 share no runtime state, they only need to agree on the Pref64::/n, using the Well-Known Prefix by default or a specific local prefix**6231295**. An end-to-end flow is therefore: an IPv6-only client receives a synthesized AAAA embedding the server's IPv4 address, sends IPv6 packets to that address (routed to the NAT64), the translator allocates or looks up the binding, translates headers, and forwards to the IPv4 server. Return traffic is reverse-translated using the session and BIB state until idle timers expire or TCP teardown is observed **6231295**, **rfc6147**. Choosing between the Well-Known Prefix and a local prefix can affect how well networks work together when DNS64 and NAT64 are run by different organizations, but it doesn't significantly impact the cost of translating individual packets**6231295**. From a deployment standpoint,

setting up NAT64 with DNS64 at the network edge is relatively simple. It lets operators run IPv6-only access networks while still reaching IPv4-only services. This is different from running a full dual-stack network, where both IPv4 and IPv6 are enabled throughout. The main trade-offs of NAT64 with DNS64 are that session initiation tends to favor IPv6 to IPv4 connections, it relies on DNS64 for translating names, and some protocols aren't fully supported. In return, it allows for much more efficient sharing of IPv4 addresses**rfc6146**. From both an operational and cost perspective, NAT64/DNS64 allows operators to run IPv6-only networks at the edge, with IPv4 needed only at the gateway. This makes managing IPv4 addresses easier and avoids the complexity of running dual-stack everywhere. According to a 2024 study, large-scale measurements show that public deployment in DNS resolvers is still quite limited. Among public IPv4 resolvers, only about 0.1Finally, NAT64/DNS64 forms the foundation of 464XLAT. In this setup, a client-side stateless translator (CLAT) handles local IPv4-only applications, while the provider's NAT64 performs the stateful IPv6-to-IPv4 translation. Both use the same Pref64::/n and DNS64 principles. This approach makes legacy applications work more seamlessly on IPv6-only networks, as discussed in the next section**6231295**. Client-Side Translation: The role of CLAT and 464XLAT Building on the NAT64/DNS64 principles, 464XLAT is a practical method to provide limited IPv4 connectivity across IPv6-only access networks without tunnels, keeping the network IPv6-first while allowing legacy IPv4-only applications to function. It is not a full IPv4 replacement: it supports outbound, client-to-server IPv4 use cases toward globally reachable IPv4 servers, does not provide inbound IPv4 reachability, and is not aimed at IPv4 peer-to-peer scenarios. The goal is to restore just enough IPv4 to keep legacy software usable while keeping native and modern traffic on IPv6 wherever possible **rfc6877**. The architecture defines two roles and introduces no new control protocols. On the customer side, the CLAT (Customer-side Translator) performs stateless IPv4/IPv6 translation (SIIT) as specified in the existing translation RFCs. It can run on a router in fixed access or directly on an end host (e.g., a smartphone) in mobile networks. Even when it runs on an end device, the CLAT forwards traffic between a private IPv4 interface for local applications and the IPv6 uplink. In many home or office setups, the CLAT also handles basic LAN services like DHCP for private IPv4 addresses and a DNS proxy. On the provider side, the PLAT (Provider-side Translator) is a stateful NAT64 that allows many IPv6-only clients to share a pool of IPv4 addresses. Together, CLAT and PLAT build on existing SIIT and NAT64 technologies—no new protocols are needed**rfc6877**. 464XLAT can operate with or without DNS64. It doesn't need synthesized AAAA records: an IPv4-only application can open IPv4 sockets or use IPv4 addresses directly, and the CLAT will translate those packets into IPv6 and send them to the PLAT, which then translates them back to IPv4. When DNS64 is

used, name-based connections to IPv4-only destinations go through a single stateful translation at the PLAT. This avoids having both a stateless step at the CLAT and a stateful step at the PLAT, reducing per-connection processing and keeping the translation state centralized at the provider**rfc6877**. The resulting packet flows are straightforward. If the destination supports IPv6, traffic goes directly over IPv6 with no translation. If the destination is IPv4-only and DNS64 is used, the client receives a synthesized AAAA record, sends IPv6 packets, and the PLAT performs a single NAT64 translation. For applications that use IPv4 addresses directly or only support IPv4 sockets, the CLAT first translates the private IPv4 packets to IPv6 in a stateless manner, and the PLAT then applies stateful NAT64. This two-step translation ensures compatibility with applications that rely on hardcoded IPv4 addresses. **rfc6877**. This model works for both fixed and mobile networks. In wired networks, the CPE (customer premises equipment) acts as the CLAT: devices on the LAN keep using private IPv4, the CPE routes native IPv6 traffic, and it applies SIIT toward the PLAT for IPv4-only traffic. In 3GPP mobile networks, the user equipment usually runs the CLAT: it provides a private IPv4 stack for local apps, translates IPv4-only traffic into IPv6 for the PLAT, and sends IPv6-native traffic directly without involving the CLAT. When tethering, the user equipment can create a small private IPv4 LAN behind it (using NAT44) and still perform stateless translation before sending traffic over the IPv6 connection, staying compatible with mobile policies**rfc6877**. From an operational perspective, 464XLAT has several advantages. By keeping IPv4 state centralized in the PLAT, providers can efficiently share limited IPv4 resources among many subscribers. Deployments are also faster to roll out, since the access network can stay IPv6-only and existing standards are reused. Running an IPv6-only access network can be simpler than maintaining dual-stack, it involves fewer protocols to manage and troubleshoot. Native IPv6 traffic stays end-to-end, and translation only happens for IPv4 flows. The PLAT can even be provided by a third party, letting an access provider run an IPv6-only network while sending CLAT-translated traffic to an external NAT64 service. This approach works especially well in mobile networks, where maintaining separate PDP contexts for IPv4 and IPv6 adds complexity**rfc6877**. Some practical considerations apply. CLAT uses standard IPv4-embedded IPv6 formats. It usually reserves separate /64 blocks for the uplink, each downlink segment, and stateless translation traffic. If a dedicated translation prefix isn't provided, the CLAT can combine LAN addresses to a single IPv4 address and map that to an IPv6 address it controls. The CLAT also needs to know the PLAT's translation prefix, which can be discovered automatically or set manually. By design, the prefixes for CLAT and PLAT translations are kept separate**rfc6877**.

2.3 Software implementations of NAT64

Tayga TAYGA is a free, stateless NAT64 implementation for Linux distributed under the GPLv2, positioned as a lightweight, production-quality translator for environments where deploying a dedicated hardware or full-blown software NAT64 device would be excessive**Repas_Farnadi_Lencse_2014**. At the time of the cited study, the latest release referenced was 0.9.2, which the authors evaluate as representative for open-source NAT64 on Linux**palrd_tayga_readme**. Its design philosophy is to perform transparent IPv6–to–IPv4 address and header translation while explicitly leaving policy enforcement and state handling to the Linux packet filtering and NAT framework (iptables)**Repas_Farnadi_Lencse_2014**. The authors stress that TAYGA does not aim to replicate the flexibility of Linux's packet filters; instead, it is built to integrate cleanly with them, keeping the translator itself simple and predictable **Repas_Farnadi_Lencse_2014**. Functionally, TAYGA provides one-to-one IPv6↔IPv4 address mapping (stateless translation) and does not offer many-to-one address multiplexing. In a typical deployment, TAYGA is paired with DNS64 and a stateful NAT44 configured in iptables **Repas_Farnadi_Lencse_2014**. For an IPv6 client reaching an IPv4 server, TAYGA maps the client's IPv6 source to a private IPv4 from a configured dynamic pool (1:1), and then NAT44 performs SNAT from that private address to the NAT64 gateway's public IPv4 **Repas_Farnadi_Lencse_2014**. On the return path, NAT44 restores the private IPv4, after which TAYGA reconstructs the corresponding IPv6 destination using its 1:1 mapping and forwards the IPv6 packet back to the client **Repas_Farnadi_Lencse_2014**. This design requires provisioning a sufficiently large private IPv4 pool to support concurrent mappings **Repas_Farnadi_Lencse_2014**. From a deployment perspective, TAYGA's stateless core means there is no built-in session tracking or policy engine; scalability and IPv4 address conservation depend on the NAT44 stage and the size of the private IPv4 pool, making TAYGA a good fit when a simple, transparent translator is needed**Repas_Farnadi_Lencse_2014**. Tundra Tundra is an open-source IPv6-to-IPv4 and IPv4-to-IPv6 translator for Linux that runs entirely in user space. It is written in C, implements SIIT**rfc7915**, and is designed to take advantage of multicore CPUs with a multi-threaded architecture. Packet input and output can be handled through the Linux TUN driver or via inherited file descriptors**labuda_tundra_nat64**. Functionally, Tundra supports several stateless translation modes. In Stateless NAT64 mode it enables a single host to reach IPv4-only destinations and, when combined with NAT66, it can be used to serve multiple IPv6-only hosts behind it **labuda_tundra_nat64**. In Stateless CLAT mode it allows IPv4-only applications on an IPv6-only network with NAT64 to access IPv4-only hosts. Deployed on a router with an IPv6-only uplink and NAT64, it can produce a dual-stack internal network when paired

with NAT44 **labuda_tundra_nat64**. Beyond these, a pure SIIT mode translates addresses that embed IPv4 within an IPv6 translation prefix and vice versa **labuda_tundra_nat64**. The design focuses on providing a minimal feature set for SIIT/NAT64/CLAT. It avoids unnecessary features and doesn't allocate any extra memory after initialization. **labuda_tundra_nat64**. The addressing model doesn't use a dynamic pool, it relies on a single fixed IP from the configuration. This means that on its own, the translator can only serve one host. However, it can scale to multiple hosts or networks when used together with NAT66 or NAT44**labuda_tundra_nat64**. Compared to similar user-space, stateless translators like Tayga, Tundra offers multi-threading, multiple configurable modes and the option to operate on inherited file descriptors, while it deliberately lacks a dynamic address pool**labuda_tundra_nat64**. Jool Jool is another well-known open-source implementation for IPv4/IPv6 translation, specifically designed for Linux systems**jool_introduction**. Unlike Tundra's focus on userspace implementation, Jool operates within the kernel space and provides support for both Stateful NAT64 and SIIT modes. The SIIT functionality, which was introduced starting with version 3.3.0, is particularly relevant for 464XLAT deployments as it enables CLAT-like functionality on Linux systems, while the NAT64 mode can handle the PLAT side of the translation process**jool_introduction**. From an architectural perspective, Jool offers two distinct integration modes for packet interception: Netfilter mode and iptables mode. Both approaches hook into the PREROUTING chain but differ in their operational characteristics**jool_introduction**. The Netfilter mode, which was the sole operation mode until Jool version 3.5, exhibits what can be described as "greedy" behavior. It intercepts all inbound packets within its network namespace without any matching conditions and attempts to translate everything, only leaving packets untouched when translation fails **jool_introduction**. This mode is limited to at most one Netfilter SIIT instance and one Netfilter NAT64 instance per network namespace and begins translating immediately upon creation. In contrast, the iptables mode, available since Jool 4.0.0, implements a more selective approach by functioning as an iptables target that can be used within specific rules**jool_introduction**. This mode leverages iptables matching system, meaning only packets that match a particular rule are handed to Jool for processing, while other traffic proceeds normally through the network stack. This design allows for any number of iptables-based Jool instances and rules per namespace, with instances remaining idle until a matching iptables rule directs packets to them **jool_introduction**. An important characteristic shared by both modes is their handling of successfully translated packets. Rather than following the conventional path through the FORWARD chain, translated packets are sent directly to POSTROUTING, effectively bypassing the FORWARD chain entirely**jool_introduction**. Jool's packet handling logic follows a system-

atic approach for determining which packets can be translated. For packets that cannot be translated, Jool returns them to the kernel under various conditions, such as when an iptables rule references a non-existent instance, when translation succeeds but the resulting packet is unroutable, or when the translator is disabled by configuration**jool_introduction**. In SIIT mode, specific untranslatable conditions include scenarios where addresses cannot be translated due to local interface addresses or absence of applicable translation strategies**jool_introduction**. The fact that Jool supports the essential protocols used in our measurements makes it well-suited for the throughput and RTT evaluations conducted with iperf and ping respectively **jool_introduction**.

Chapter 3

Related Work

Chapter 4

Implementation

4.1 Test environments

4.2 Implementation of Translation Technologies

4.3 Networking Namespace Configuration

4.4 Measurement Methodology

4.5 Challenges and Solutions

Chapter 5

Results

5.1 Throughput

5.2 RTT

5.3 Discussion

Chapter 6

Evaluation

6.1 RTT

6.2 Throughput

6.3 TSC vs HPET Clocksource Difference

Chapter 7

Conclusion

Chapter 8

References

Appendix

Bibliography

[1] J. Beeharry and B. Nowbutsing, "Forecasting ipv4 exhaustion and ipv6 migration," in *2016 IEEE International Conference on Emerging Technologies and Innovative Business Practices for the Transformation of Societies (EmergiTech)*, 2016, pp. 336–340. DOI: 10.1109/EmergiTech.2016.7737362.

[2] S. L. Levin and S. Schmidt, "Ipv4 to ipv6: Challenges, solutions, and lessons," *Telecommunications Policy*, vol. 38, no. 11, pp. 1059–1068, 2014, ISSN: 0308-5961. DOI: https://doi.org/10.1016/j.telpol.2014.06.008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0308596114001128.

[3] *Internet Protocol*, RFC 791, Sep. 1981. DOI: 10.17487/RFC0791. [Online]. Available: https://www.rfc-editor.org/info/rfc791.

[4] R. Moskowitz, D. Karrenberg, Y. Rekhter, E. Lear, and G. J. de Groot, *Address Allocation for Private Internets*, RFC 1918, Feb. 1996. DOI: 10.17487/RFC1918. [Online]. Available: https://www.rfc-editor.org/info/rfc1918.

[5] D. S. E. Deering and B. Hinden, *Internet Protocol, Version 6 (IPv6) Specification*, RFC 1883, Dec. 1995. DOI: 10.17487/RFC1883. [Online]. Available: https://www.rfc-editor.org/info/rfc1883.

[6] M. Holdrege and P. Srisuresh, *IP Network Address Translator (NAT) Terminology and Considerations*, RFC 2663, Aug. 1999. DOI: 10.17487/RFC2663. [Online]. Available: https://www.rfc-editor.org/info/rfc2663.

[7] I. Livadariu, K. Benson, A. Elmokashfi, A. Dhamdhere, and A. Dainotti, "Inferring carrier-grade nat deployment in the wild," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, IEEE, 2018, pp. 2249–2257.

[8] T. L. Hain, *Architectural Implications of NAT*, RFC 2993, Nov. 2000. DOI: 10.17487/RFC2993. [Online]. Available: https://www.rfc-editor.org/info/rfc2993.

List of Figures

List of Listings