v6ops Working Group
Internet-Draft
Intended status: Best Current Practice
Expires: 26 January 2026

P. S. Tiesel
SAP SE

25 July 2025

Testing Applications' IPv6 Support
draft-tiesel-v6ops-ipv6-app-testing-00

Abstract

This document provides guidance for application developers and software as a service providers on how to approach IPv6 testing in Dual-Stack (IPv4+IPv6), and IPv6-only scenarios, including "true IPv6-only" scenarios. It discusses common misconceptions about the degree to which operating systems and libraries can abstract IPv6 issues away and explains common regressions to avoid when deploying IPv6 support.

Discussion Venues

This note is to be removed before publishing as an RFC.

Discussion of this document takes place on the IPv6 Operations Working Group mailing list (v6ops@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/v6ops/.

Source for this draft and an issue tracker can be found at https://github.com/philsbln/ipv6-app-testing.

Status of This Memo

Table of Contents

1. Introduction

For the last 20 years, enabling applications for IPv6 has focused on coexistence with IPv4 and allowing traffic to shift towards IPv6 without breaking IPv4 operation. With the US mandate to move all governmental agencies to "IPv6-only" [M-21-07], this target for IPv6 support changed to being fully functional in the absence of IPv4 and transition technologies providing connectivity to the IPv4 Internet. Therefore, today's applications are expected to function regardless of whether they are used in an IPv4-only environment, a Dual-Stack environment, or an IPv6-only environment, with or without connectivity to the IPv4 Internet. To achieve this, applications need to be verified against all these scenarios.

While the availability of IPv6 support in applications has a considerable impact on the success of IPv6, there exists no documented best current practices how to do so. Testing IPv6 compliance of network gear and operating systems has been documented extensively. While the IETF does not define compliance tests, best current practice exists for the behavior of general IPv6 nodes [RFC8405] and Customer Edge (CE) routers [I-D.draft-winters-v6ops-rfc7084bis].

To fill that gap, this document provides guidance for application developers and cloud application providers on how to approach IPv6 testing. It described which scenarios they should consider validating against, and which common regressions to avoid when adding IPv6 support. While many application developers assume that the network abstractions of the operating system (OS), communication libraries, and application frameworks will handle the transition towards IPv6 transparently, leaky abstractions within these frameworks will make it difficult for an application developer to write address family-independent code for features such as allow/deny lists and logging. In addition to that challenge, modern cloud applications are typically composed of hundreds to thousands of micro- and macro-services, forming a complex distributed system that requires intricate communication and orchestration infrastructure to operate. Enabling these applications to communicate over IPv6 requires careful analysis of data flows within all services and proper IPv6 support in all components that may require IPv6 traffic, as well as IPv6 addresses as metadata.

2. Conventions and Definitions

2.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2.2. Base Scenarios

Within this document, we define the following "base scenarios" in which applications ought to be verified for availability and functional correctness:

IPv4-only: A node or application that has native connectivity towards the IPv4 Internet and no connectivity towards the IPv6-only Internet.

Dual-Stack: A node or application that has native connectivity towards the IPv4 as well as the IPv6 Internet.

IPv6-only with NAT64: A node or application that has native connectivity towards the IPv6 Internet and connectivity towards the IPv4 Internet using a transition technology like NAT64.

True IPv6-only: A node or application that has native connectivity towards the IPv6 Internet and no connectivity towards the IPv4-only Internet.

3. Testing Objectives

As a basic principle, IPv6 application testing should always be derived from functional and integration testing. Therefore, the goal is to verify that the expected behavior is consistent across all connectivity scenarios, i.e., the application functions correctly in IPv4-only, Dual-Stack, IPv6-only with NAT64 and True IPv6-only settings. The following sections provide guidance on which connectivity scenarios to include in a testing campaign and how to approach testing complex cloud applications.

3.1. Connectivity Scenarios

Table 1 lists the combinations of connectivity scenarios that application testing should generally consider. Note, while the involved parties are listed here as "client" and "server" to reflect the most common case, the combinations can be used the same way when considering peer-to-peer applications, while NAT64 becomes replaceable with other network functions like TURN offering translation capabilities.

The first five scenarios marked as _base_ should cover all major code paths and fallback conditions. These include Dual-Stack clients combined with IPv4-only and a True IPv6-only server, to test wither the additional address family confused the client. We also include then cases with Dual-Stack Server and Single-Stack clients, to test whether a single address family at client side works as anticipated and look at the transition case using NAT64. We have no special scenarios for 464XLAT [RFC6877] and IPv6-Mostly [I-D.draft-ietf-v6ops-6mops], as these architectures are from then client side indistinguishable from the Dual-Stack (464XLAT or IPv6-Mostly with CLAT) or IPv6-only with NAT64 (IPv6-Mostly without CLAT).

For the IPv6-only datacenter case, where servers may be exposed to the IPv4-only Internet using NAT64, it is also advisable to consider the case marked as IPv6-only-DC in Table 1.

The other combinations are unlikely to exhibit additional problems for client-server-based applications and therefore are marked as extended in Table 1. For peer-to-peer applications and applications with complex connection handling like using STUN [RFC5389] or TURN [RFC5766], skipping these scenarios is strongly discouraged. In case of TURN, it is also recommended to test with and without TURN relay in the path, essentially doubling the number or scenarios.

Client	Server	Verdict
Dual-Stack	IPv4-only	base
Dual-Stack	True IPv6-only	base
IPv4-only	Dual-Stack	base
IPv6-only with NAT64	IPv4-only	base
True IPv6-only	Dual-Stack	base
IPv4-only	IPv6-only with NAT64	IPv6-only-DC
Dual-Stack	Dual-Stack	extended
IPv4-only	IPv4-only	extended
IPv6-only with NAT64	True IPv6-only	extended
True IPv6-only	True IPv6-only	extended

Table 1: Scenario combinations to consider for IPv6 testing

3.2. Testing Complex Cloud Applications and Applying Test Cases

When testing complex applications, especially cloud applications, they typically involve countless data flows. For some of these, the application may be considered as server, while being a client in others. Therefore, test cases need to cover each data flow in all relevant scenarios.

As functional and integration tests are often defined as end-to-end test cases, they often involve several components, e.g., micro-services, load-balancers, application gateways, logging, authentication, and authorization services, which use IP-based protocols between the components. Therefore, an end-to-end test case breaks down to a series of flows between components, and for each of these flows, we need to determine whether we need to apply the connectivity scenarios from Table 1 to it, of whether the connectivity scenarios are only controlled by the deployment of the application.

For external flows, i.e., flows outside the developers’ control, usually all base scenarios from Section 3.1 need to be accounted for. If one side of the flow is under administrative control, the number

of scenario combinations can still be limited: For example, a cloud software provider choosing to deploy Dual-Stack endpoints can skip all non-Dual-Stack cases on the respective side of the communication. For internal flows, the relevant scenarios only depend on the applications' architecture, and only scenarios planned in the deployment need to be considered. From a networking perspective, flows between components are typically independent. There is no need to run the Cartesian product of scenarios x communications as long as all relevant scenarios for a given flow are tested.

In addition to the data flows, an implementation may include metadata about the data flow when communicating with backend systems, e.g., for logging or authorization purposes. While the flows towards these backend systems themselves may be safe to ignore as outlined above, the functional correctness of the backend systems for all kinds of IP address need to be verified as part of the test series. Ignoring IP addresses as data in the testing may result in malfunctions, like always denying access over IPv6, or security issues, like not logging access from IPv6 clients.

3.3. Special considerations for Web-based Applications

Web-based applications usually load resources from multiple parties, including CDNs and analytic tools, involving data flows to all these parties. When facing the requirement to support True IPv6-only users, being unable to load some resources due to missing/defective IPv6 support at the respective parties can have any effect from missing analytics insights or ad revenue to severe functional defects rendering the application unusable. When testing such applications, it is not sufficient to only focus on the initial/main interactions, but it is necessary to consider all resources and parties providing them. As Web browsers load these resources dynamically and third-party resources may themselves may request resources from more parties, this kind of testing usually requires an instrumented Web browser, e.g., using [Selenium].

4. Testing Strategies

Na誰ve IPv6 testing, based on end-to-end functional tests as outlined in Section 3, would require running a set of functional tests in various connectivity scenarios. In certain environments, setting up test cases for all scenarios can become forbiddingly expensive, especially for complex cloud applications, application platforms, or when dealing with corporate IT environments. While in today's environment getting Dual-Stack connectivity is possible in most cases,

In this section, we give recommendations how to set up scenarios defined in Section 3.1 and present strategies to meet the relevant testing objective by modifying the client or using Dual-Stack clients and servers to conclude the results for other scenario combinations, e.g., by tracing whether the right address family is used.

4.1. True IPv6-only Clients

This is the most natural way to test whether True IPv6-only clients behave correctly. The client device is either placed in a network without IPv4 connectivity or the IPv4 stack is disabled on the device while it is in a Dual-Stack network. While most desktop operating systems allow disabling IPv4, mobile operating systems, such as Android and iOS, do not. For mobiles operating systems, a True IPv6-only environment is needed.

In both cases, it has to be ensured that there is no way to access IPv4-only resources. In particular, fallback to NAT64 must be prevented by disabling CLAT [CLAT], making sure DNS resolution does not perform DNS64 address synthesis [RFC6147] and blocking the well-known NAT64 prefix [RFC6052] for these clients. In addition, VPN services including privacy services like [iCloud-Private-Relay] need to be disabled as they can provide connectivity towards the IPv4 Internet.

A note on the applicability of disabling IPv4: Before disabling IPv4 make sure the environment supports IPv6-only operation. Many desktop virtualization environments become unusable because IPv4 is needed to access and manage the virtual machines. Some corporate environments may render the machines unusable as they require IPv4 connectivity for sign-on.

4.2. IPv6-only Servers

IPv6-only servers are a good option when setting up a True IPv6-only client environment is infeasible and clients are know to only contact a single server or a small number of servers under the testers' control. Even if setting up a True IPv6-only server environment is infeasible, most testing is also achievable by setting up a dedicated DNS name only containing an AAAA record pointing to the IPv6 addresses of an otherwise Dual-Stack server.

4.3. Client-based tracing

If we can't limit the available address families, we can still trace and verify whether the address family desired for the scenario is used.

Client-based tracing is especially useful when Dual-Stack servers and clients are available and a conclusion for the True IPv6-only case is desired. By using the clients' logging/tracing/debugging functionality, the tester can verify that the actual data flows happen over IPv6, which is preferred by most network abstractions. If the client allows changing the preference between IPv6 and IPv4, IPv4-only testing is also possible.

The most relevant case for this strategy is testing Web applications. By examining the Web browsers' performance log or using a plugin like [IPvFoo] that visualizes connectivity information, the tester can determine whether all resources are available using IPv6.

4.4. Server-based tracing

Analogue to tracing on the client side, it is also possible to look at the protocols used on the server side. While this is functionally equivalent for protocols where clients only communicate to a single server, this approach is not feasible for Web-based applications where a client usually needs flows towards many servers, where client or network based tracing are the only feasible alternatives to testing with an True IPv6-only client.

4.5. Network-based tracing

If the communication pattern of an application is known well enough, a packet tracer as [Wireshark] allows to verify that an application uses IPv6 for all flows in a Dual-Stack environment. If this can be verified, failures in True IPv6-only environments are unlikely.

While this is the least invasive method of testing True IPv6 scenarios in a Dual-Stack setup, it is the most error-prone as it requires the tester to fully understand the network flows of the application and requires the skills to interpret the output of a packet tracer.

5. Common Sources of IPv6 Related Failures and Misbehavior

In this section, we discuss special failure modes that can cause unexpected application behavior that is hard to debug. While some of these cases can be automatically mitigated, especially through generalizing the concept of Happy Eyeballs [RFC8305], others may not. In cases that developers choose not to mitigate erroneous application behavior, users and operators should be supported in the resolution by exposing specific and detailed error or debug messages.

5.1. Enable IPv6 Feature Gates

Some applications completely ignore IPv6 unless explicitly configured to enable IPv6. This adds another class of user or configuration errors, like deploying an application without enabled IPv6 support in an IPv6-only environment. As these feature gates are often buried deeply in the documentation and are often vendor, product, or component specific, every component needs to be checked to determine whether IPv6 support needs extra configuration.

5.2. Destination Address Selection Preference and Address Filtering

The destination address selection algorithm in [RFC6724] filters unavailable address families (Rule 1) and de-prioritizes non-matching address families (Rule 2) and clearly prioritizes IPv6 GUA addresses over IPv4 addresses. While most operating systems and some alternative resolver libraries, such as [C-ARES], implement [RFC6724] or its predecessor [RFC3484] correctly, there are a number of notable and widely used implementations that implement something else, causing anything from unexpected behavior to hard-to-debug errors.

* Most JAVA runtimes do the opposite and prefer IPv4 destinations over IPv6. To prefer IPv6 addresses over IPv4, one needs to set the system property java.net.preferIPv6Addresses=true.

* Some applications only use the first address candidate from the getaddrinfo() and fail if the connection attempt to that one fails.

* Applications composed of services built on different programming languages or runtimes may behave inconsistently with regard to choosing destination addresses.

* NGINX has its own user DNS resolver without address filtering; thus, adding a _AAAA_ record to a backend can render that backend unusable. After having resolved a _AAAA_ record, it is trying to open an IPv6 socket, even when the IPv6 stack is disabled. As socket creation failure is not expected, an internal server error is sent back to the client.

5.3. Input Validation and Output Rendering

While most libraries and application frameworks have decent IPv6 support, there often is still application logic checking whether user input is a valid IPv4 address or rendering output under the assumption that an address is always an IPv4 address, preventing to take advantage of the IPv6 support by the underlying components.

5.4. Misbehaving Middle-Boxes

In practice, many IPv6-related regressions uncovered during testing turn out to be caused by hidden components outside of the application developers' control. Middle-Boxes, e.g., firewalls, virus scanners, and intrusion detection systems, can break end-to-end tests in surprising ways, like terminating TLS sessions over IPv6 with certain extensions in the _TLS client hello_ while correctly passing the same flow over IPv4.

6. Deployment Considerations

Lab testing of applications for IPv6 compliance should always have the next step in mind: Deploying the application and providing the users with decent IPv6 support. Therefore, end-to-end tests, especially of cloud applications, should also keep deployment steps, prerequisites, and risks in mind. This section discusses some issues to keep in mind when planning and executing IPv6 testing.

6.1. Operational Scope & Software Lifecycle

Depending on the application and deployment model, the timing of deploying IPv6 support may be in control of the users' organization, the developers' organization, or neither of them. Based on this setup, certain combinations of IPv6-enabled clients, servers, and infrastructure in between may or may not be excluded from consideration. Therefore, it may be necessary to add test cases for old software versions with known and already fixed bugs against newly IPv6-enabled servers. If regressions and service disruptions cannot be ruled out by the tests, a per-user or per-customer tenant opt-in/ opt-out/roll-back scheme for the IPv6 enablement should be considered.

6.2. Allow & Deny Lists

Application-level IP allow and deny lists pose a special challenge for deploying IPv6 in a cloud application. As users may already have IPv6 connectivity, adding IPv6 support to the server may cause clients to use IPv6 immediately. Having no allow list entry for the users' IPv6 addresses results in service disruptions. Happy Eyeballs as defined in [RFC8305] does not solve the problem as allow list checks usually take place after the transport connection has already been established.

To mitigate allow or deny lists causing service disruptions when enabling IPv6, support to include IPv6 addresses in allow and deny lists needs to be enabled way before rolling out IPv6 on the transport and communicated towards the users. To further limit the probability of service disruptions, generalizing Happy Eyeballs to re-try using IPv4 after certain error conditions should be evaluated.

6.3. Component and Service Reuse

If components or cloud services can be reused in other products, special care needs to be taken when planning IPv6 deployment. The interaction contracts between the reusing parties and the service need to be checked whether IPv6 enablement of the services also affects the flows of these. Additional end-to-end tests, including the reusing parties, are recommended. This is often a recursive process宴ヲ

6.4. Ownership of Software Components

Sometimes IPv6 enablement requires touching components that are not actively maintained anymore. Be prepared for this and plan extra time or budget for updating or replacing these components.

7. Security Considerations

The document itself has no specific security implications; thus, some of the issues discussed in Section 5 have.

8. IANA Considerations

This document has no IANA actions.

9. References

9.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC3484] Draves, R., "Default Address Selection for Internet Protocol version 6 (IPv6)", RFC 3484, DOI 10.17487/RFC3484, February 2003, <https://www.rfc-editor.org/rfc/rfc3484>.

[RFC6724] Thaler, D., Ed., Draves, R., Matsumoto, A., and T. Chown, "Default Address Selection for Internet Protocol Version 6 (IPv6)", RFC 6724, DOI 10.17487/RFC6724, September 2012, <https://www.rfc-editor.org/rfc/rfc6724>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

9.2. Informative References

[C-ARES] "C-ARES - a modern DNS (stub) resolver library, written in C", n.d., <https://c-ares.org/>.

[CLAT] Linkova, J. and T. Jensen, "464XLAT Customer-side Translator (CLAT): Node Recommendations", Work in Progress, Internet-Draft, draft-ietf-v6ops-claton-05, 6 July 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-v6ops-claton-05>.

[I-D.draft-ietf-v6ops-6mops] Buraglio, N., Caletka, O., and J. Linkova, "IPv6-Mostly Networks: Deployment and Operations Considerations", Work in Progress, Internet-Draft, draft-ietf-v6ops-6mops-01, 3 March 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-v6ops-6mops-01>.

[I-D.draft-winters-v6ops-rfc7084bis] Lencse, G., Martinez, J. P., Patton, and T. Winters, "Basic Requirements for IPv6 Customer Edge Routers", Work in Progress, Internet-Draft, draft-winters-v6ops-rfc7084bis-03, 8 July 2024, <https://datatracker.ietf.org/doc/html/draft-winters-v6ops-rfc7084bis-03>.

[iCloud-Private-Relay] "Apple iCLoud Private Relay (WWDC2021)", n.d., <https://developer.apple.com/videos/play/wwdc2021/10096/>.

[IPvFoo] Marks, P., "IPvFoo - a Chrome/Firefox extension that adds an icon to indicate whether the current page was fetched using IPv4 or IPv6.", n.d., <https://github.com/pmarks-net/ipvfoo>.

[M-21-07] "M-21-07 寰Completing the Transition to Internet Protocol Version 6 (IPv6)", United States of America Office of Management and Budget Memorandum for Heads of Executive Departments and Agencies, 19 November 2020, <https://www.whitehouse.gov/wp-content/uploads/2020/11/M-21-07.pdf>.

[RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, DOI 10.17487/RFC5389, October 2008, <https://www.rfc-editor.org/rfc/rfc5389>.

[RFC5766] Mahy, R., Matthews, P., and J. Rosenberg, "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)", RFC 5766, DOI 10.17487/RFC5766, April 2010, <https://www.rfc-editor.org/rfc/rfc5766>.

[RFC6052] Bao, C., Huitema, C., Bagnulo, M., Boucadair, M., and X. Li, "IPv6 Addressing of IPv4/IPv6 Translators", RFC 6052, DOI 10.17487/RFC6052, October 2010, <https://www.rfc-editor.org/rfc/rfc6052>.

[RFC6147] Bagnulo, M., Sullivan, A., Matthews, P., and I. van Beijnum, "DNS64: DNS Extensions for Network Address Translation from IPv6 Clients to IPv4 Servers", RFC 6147, DOI 10.17487/RFC6147, April 2011, <https://www.rfc-editor.org/rfc/rfc6147>.

[RFC6877] Mawatari, M., Kawashima, M., and C. Byrne, "464XLAT: Combination of Stateful and Stateless Translation", RFC 6877, DOI 10.17487/RFC6877, April 2013, <https://www.rfc-editor.org/rfc/rfc6877>.

[RFC8305] Schinazi, D. and T. Pauly, "Happy Eyeballs Version 2: Better Connectivity Using Concurrency", RFC 8305, DOI 10.17487/RFC8305, December 2017, <https://www.rfc-editor.org/rfc/rfc8305>.

[RFC8405] Decraene, B., Litkowski, S., Gredler, H., Lindem, A., Francois, P., and C. Bowers, "Shortest Path First (SPF) Back-Off Delay Algorithm for Link-State IGPs", RFC 8405, DOI 10.17487/RFC8405, June 2018, <https://www.rfc-editor.org/rfc/rfc8405>.

[Selenium] "Selenium WebDriver", n.d., <https://www.selenium.dev/>.

[Wireshark]
"Wireshark packet tracer", n.d.,
<https://www.wireshark.org/>.

Acknowledgments

TODO acknowledge.

Author’s Address

Philipp S. Tiesel
SAP SE
Email: philipp@tiesel.net