v6ops
Internet-Draft
Intended status: Best Current Practice
Expires: 27 November 2026

P. S. Tiesel
SAP SE
J. Linkova
Google
K. Ouellette
B. Patton
UNH-IOL

26 May 2026

Testing Applications' IPv6 Support
draft-ietf-v6ops-ipv6-app-testing-01

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 "IPv6-only-strict" scenarios without any connectivity towards any relevant IPv4 endpoint. 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.

About This Document

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

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-v6ops-ipv6-app-testing/.

Source for this draft and an issue tracker can be found at https://github.com/ietf-wg-v6ops/draft-itef-v6ops-ipv6-app-testing.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 27 November 2026.

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. This target has changed in part due to a series of national regulations mandating state entities to proceed in the migration to IPv6, e.g., in China [CN-CAC-2023], the United States of America [US-OMB-M-21-07], Germany [DE-BIT-2020-14], and the Czech Republic [CZ-ENDv4]. IPv6 support today means 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 [RFC8504] and Customer Edge (CE) routers [I-D.draft-ietf-v6ops-rfc7084bis].

To fill that gap, this document provides guidance for application developers and cloud application providers on how to approach IPv6 testing. It describes 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 macroservices, 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 Connectivity Scenarios

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

IPv4-only: A node or application that has native connectivity towards all endpoints relevant for the test scenario using IPv4 and no connectivity towards any relevant IPv6 endpoints.

Dual-stack: A node or application that has native connectivity towards all endpoints relevant for the test scenario using IPv4 as well as using IPv6.

IPv6-only with NAT64: A node or application that has native connectivity towards all endpoints relevant for the test scenario using IPv6 and connectivity towards IPv4 endpoints using a transition technology like NAT64, e.g., NAT64 in combination with CLAT, DNS64, or local address synthesis.

IPv6-only-strict: A node or application that has native connectivity

towards all endpoints relevant for the test scenario using IPv6 and no connectivity towards any relevant IPv4 endpoints, neither encapsulated nor translated. This definition slightly diverges from the one in [I-D.draft-palet-v6ops-ipv6-only] as it ignores IPv4 connectivity to anywhere outside the testing scope.

2.3. Lifecycle Functions

Orthogonal to the Base Scenarios, we define lifecycle functions, i.e., the phases in which an application is approached during a simplified lifecycle of the application, in accordance to [US-NIST.SP.500-267Ar1] as follows:

* Installation: The installation of the application including any initial configuration required for getting the application in a state where remote services are operational.

* User Interface: All forms of interactive access to the application (e.g., Web UI, API).

* Management: All forms of remote management and monitoring functions.

* Update: All forms of update functions, including both automatic and manual update mechanisms.

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 IPv6-only-strict 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 with "client" representing the initiating or first acting party.

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 IPv6-only-strict server, to test whether the additional address family confused the client. We also include the 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 the 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 [RFC8489] or TURN [RFC8656], skipping these scenarios is strongly discouraged.

Client	Server	Classification
Dual-stack	IPv4-only	base
Dual-stack	IPv6-only-strict	base
IPv4-only	Dual-stack	base
IPv6-only with NAT64	IPv4-only	base
IPv6-only-strict	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	IPv6-only-strict	extended
IPv6-only-strict	IPv6-only-strict	extended

Table 1: Connectivity scenario combinations to consider

3.2. Testing with Intermediaries (e.g., Proxies)

Many application protocols support communicating across intermediates, most commonly HTTP, HTTP-Connect, SOCKS, or MASQUE proxies. Peer-to-peer applications often support TURN [RFC5766] as an intermediary to traverse NAT and provide connectivity between IPv4-only and IPv6-only hosts. When testing connectivity scenarios for an application, additional test cases including a proxy are recommended. As a proxy can convert between address families, all combinations shown in Table 2, consisting of base scenarios towards the proxy and (assuming the same scenarios on both sides of the proxy) the respective base scenarios from the proxy to the server, should be considered for testing.

Client	Proxy	Server
Dual-stack	IPv4-only	Dual-stack
Dual-stack	IPv6-only-strict	Dual-stack
IPv4-only	Dual-stack	IPv4-only
IPv4-only	Dual-stack	IPv6-only-strict
IPv6-only with NAT64	IPv4-only	Dual-stack
IPv6-only-strict	Dual-stack	IPv4-only
IPv6-only-strict	Dual-stack	IPv6-only-strict

Table 2: Base scenario combinations including a proxy to consider for IPv6 testing

3.3. Testing Name Resolution Issues

As most applications use name resolution to bootstrap their connectivity, it is necessary to consider name resolution aspects when testing IPv6 readiness. While some name resolution issues only manifest in certain connectivity scenarios or can be mitigated by using Happy Eyeballs [RFC6555]/[RFC8305], others will just map to different connectivity scenarios. In this section, we list name resolution issues to consider for testing.

3.3.1. Missing DNS Records

While a server endpoint is intended to support dual-stack connectivity, the A or AAAA DNS records for the endpoint may be missing, e.g., due to misconfiguration or broken tooling, or does not reach the client endpoint, e.g., because it got filtered out by a middle box or local resolver. The same can happen for names discovered and resolved through mDNS [RFC6762].

While deployment and integration testing should try to test for this kind of broken connectivity, this scenario is usually indistinguishable from an IPv4-only or an IPv6-only server endpoint, and therefore already addressed by testing the base scenarios above.

3.3.2. Incorrect DNS Records

Independent of the deployed server endpoint, there may be an A and AAAA record either pointing somewhere else, e.g., to an old or planned deployment.

For either IPv4-only or IPv6-only-strict clients, this scenario should always fail.

For Dual-stack clients, it should be tested whether they can use the working IPv4-only or IPv6-only connectivity scenario, either by using Happy Eyeballs [RFC6555]/[RFC8305] or trying the next resolved address candidate after timeout. Especially for the latter, it is advisable to verify whether the connection delay is acceptable of the desired use-case.

IPv6-only clients with NAT64 are only expected to work with broken AAAA records when deployed with CLAT (should behave like Dual-stack as discussed in Section Section 3.1) or local NAT64 address, e.g., as when implementing Happy Eyeballs v2 [RFC8305]. IPv6-only clients with NAT64 that rely on DNS64 only are expected to fail as the presence of AAAA records prevents synthesis of DNS64 records.

Testing with IPv4-Mapped IPv6 Addresses [RFC4291] in AAAA records is also recommended. While this makes zero sense, it has been seen in the wild and should not confuse the client.

3.3.3. DNS delegation issues

Integration testing for Cloud applications should verify that the necessary domain names are resolvable from IPv4-only or IPv6-only-strict DNS resolvers. [I-D.draft-ietf-dnsop-3901bis] describes misconfigurations that may prevent this and should be prevented.

3.3.4. Testing with IP literals

Most name resolution libraries support IP literals, i.e., textual representations of IP addresses. Applications should be tested to determine whether they work as expected with IPv4 literals and all IPv6 address representations described in [RFC4291].

If there is a use-case for link-local communication using IP literals, it should be tested whether the zone identifier can be entered as described in [RFC9844] and work as expected.

3.4. Testing with Partially Broken Connectivity, MTU, and Fragmentation Issues

When multiple address families are available, network packets may traverse different paths depending on the address family. Even when the same path is traversed, the path can exhibit distinct behaviors, e.g., dropping all or particular packets, especially in the presence of middle-boxes. From the communication endpoints that are expected to be reachable using both address families, some may only be reachable by one address family, while others may only be reachable by the other. Testing applications against these scenarios can become a key enabler for users' acceptance of IPv6, especially during a transition phase where partially broken connectivity is expected more frequently.

In some cases, connectivity issues may only become apparent late in the communication process, for example, after a successful TCP handshake but before a TLS handshake succeeds. In such scenarios, clients restricted to a single address family — such as IPv6-only-strict clients — may experience complete loss of connectivity in these scenarios, while dual-stack clients often mask such failures by automatically falling back to another address family.

In addition to partial blackholing, MTU issues may be limited to one address family or behave differently with respect to aspects like MTU available, dropping of fragmented packets, and ICMP messages generated. As only IPv4 supports on-path fragmentation, IPv6 is more dependent on working ICMP _packet too big_ reporting.

It is advisable to test for partial blackholing and MTU issues during deployment and integration testing by testing with IPv4-only and IPv6-only-strict clients to detect such blackholes. In case these issues can occur outside the testers' circle of control, it is advisable to simulate this type of failure and ensure that the application's behavior supports the detection and analysis of these errors.

3.5. Testing without IPv4 Loopback Addresses (127.0.0.0/8)

Some applications and services may assume the existence and reachability of the IPv4 loopback addresses (127.0.0.0/8) when binding to a socket or for communicating with other services on the same host. For example, a web server may explicitly listen on a 127.0.0.0/8 by default. For IPv6-only-strict scenarios, system administrators may choose to disable IPv4, including loopback. In such cases, applications may fail to operate correctly. Applications expecting to bind to a IPv4 loopback address may fail to start when these addresses are unavailable due to a bind failure. Applications expecting these addresses to be available for inter-service communication will result in these services being unable to communicate properly.

Because of this, when testing applications for the IPv6-only-strict scenario, it is recommended to test the application in an environment without IPv4 on the loopback interface.

3.6. Testing Lifecycle Function Considerations

To cover the whole lifecycle of an application including installation, user interface, management, and update, it is recommended to test that the lifecycle functions defined in Section 2.3 are operational within the connectivity scenarios defined in Table 1.

In particular, keep the following considerations in mind:

* Installation: Installation may require communications with remote first-party services (e.g., activation/license server) or remote third-party services (e.g., package repositories). In these scenarios, the installer acts as the client, and the remote service acts as the server. In cases of remote third-party services, testing all server scenarios in Table 1 may not be feasible, and impact the client scenarios that can be supported. For example, if a third-party service is IPv4-only, then supporting a IPv6-only-strict client is not feasible.

* User Interface: User interfaces can be incredibly complex with numerous contexts, views, API endpoints, CLI commands, etc. When testing non-web-based user interfaces, it is recommended to focus on components of the interface that involve communications with remote services, and those that handle network configuration parameters. For example, a network configuration interface may only accept IPv4 address literals for certain parameters. For testing web-based user interfaces, see Section 3.8.

* Management: Depending on the application, management functions may be provided via the user interface. However, the application may have additional management functions (e.g., SNMP, syslog, etc.) that should be tested. As the source triggering application behavior is crucial for logging and auditing functionality, addresses recorded need to be verified to be represented correctly. See Section Section 3.9 about representation of addresses.

* Update: Depending on the application, update functions may be exercised during installation. However, the application may have additional update functions (e.g., automatic updates, manual update mechanisms, etc.) that should be tested.

3.7. Testing Complex Cloud Applications and Applying Test Cases

When testing complex applications, especially cloud applications, they typically involve many data flows. An application or component may be considered as a server for some of these, 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, or 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.

3.8. 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 IPv6-only-strict users, being unable to load some resources due to missing/defective IPv6 support at the respective parties can have effects 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].

3.9. Considerations for Addresses as Data

When applications process IP addresses as data, e.g., as part of logging or management functionality, this functionality needs to work for all possible address families and representations.

One challenge to consider is that the textual representation of IPv4 and IPv6 addresses are not canonical. It allows several valid textual representations for the same address, which makes direct string comparison and arithmetic operations on addresses error-prone and slow. For example, the IPv6 address 2001:db8::1 can be written as 2001:db8:0:0:0:0:0:1, 2001:0db8::0.0.0.1, or in several other forms.

While custom logic to check, parse and process addresses is often error-prone and should be validated thoroughly, modern environments and frameworks usually provide data structures that encapsulate the canonical binary representation and include methods for parsing the various textual representations, comparing addresses, performing subnet operations, and rendering addresses in a consistent format.

For situations where textual representation of IPv6 addresses is needed — such as in user interfaces, logging output, and text-based data formats like JSON, YAML, TOML, and XML — [RFC5952] provides recommendations on which of the valid textual representations should be used. Applications should be tested whether they follow [RFC5952] when rendering IPv6 addresses in textual form, as required by national regulations like [US-NIST.SP.500-267Ar1], while accepting all valid representations defined in [RFC4291].

4. Testing Strategies

Nave 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.

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 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. IPv6-only-strict Clients

This is the most natural way to test whether IPv6-only-strict 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 IPv6-only-strict 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 IPv6-only-strict 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 IPv6-only-strict 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 IPv6-only-strict 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 IPv6-only-strict 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 in a Dual-stack environment uses IPv6 for all of its flows. If this can be verified, failures in IPv6-only-strict environments are unlikely.

While this is the least invasive method of testing IPv6-only-strict 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 [ADDR-SELECT] 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 [ADDR-SELECT] or its predecessors [RFC6724]/[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.

* Some resolvers ignore address families for which no default route exists or where the default-route is pointing to an unsupported/ ignored device. This becomes cumbersome especially in split-VPN use cases, e.g. when trying to contact IPv6-only endpoints via the VPN while having IPv4-only Internet connectivity.

5.3. Input Validation and Output Rendering

While most libraries and application frameworks have decent IPv6 support, there often is still application logic that prevents taking advantage of the IPv6 support by the underlying components. Checking whether user input is a valid IPv4 address or rendering output under the assumption that an address is always an IPv4 address are typical examples for this class of limitations.

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.

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 testing procedures described in this document do not create any new security implications. Some security-related issues that should be considered and ruled out by appropriate testing are discussed in Section 5.

8. IANA Considerations

This document has no IANA actions.

9. References

9.1. Normative References

[ADDR-SELECT] Buraglio, N., Chown, T., and J. Duncan, "Prioritizing known-local IPv6 ULAs through address selection policy", Work in Progress, Internet-Draft, draft-ietf-6man-rfc6724-update-25, 11 August 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-6man-rfc6724-update-25>.

[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>.

[RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, DOI 10.17487/RFC4291, February 2006, <https://www.rfc-editor.org/rfc/rfc4291>.

[RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, DOI 10.17487/RFC5952, August 2010, <https://www.rfc-editor.org/rfc/rfc5952>.

[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] Colitti, L., Linkova, J., and T. Jensen, "464XLAT Customer-side Translator (CLAT): Node Behavior and Recommendations", Work in Progress, Internet-Draft, draft-ietf-v6ops-claton-16, 5 March 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-v6ops-claton-16>.

[CN-CAC-2023] "2023 Work Arrangement for Further Promoting Large-scale IPv6 Deployment and Application", 27 April 2023, <http://www.cac.gov.cn/2023-04/27/c_1684239012351367.htm>.

[CZ-ENDv4] "Czech Republic sets IPv4 end date", 17 January 2024, <https://konecipv4.cz/>.

[DE-BIT-2020-14] "Zukunftsfhige Netzinfrastrukturen auf Basis von funktionsfhigem IPv6", IT-Steuerung Bund Beschluss 2020/14, 11 November 2020, <https://www.cio.bund.de/SharedDocs/downloads/Webs/CIO/DE/it-beirat/beschluesse/2020_14_Beschluss_Konferenz_IT-Beauftragte.pdf>.

[I-D.draft-ietf-dnsop-3901bis] Yamamoto, M. and T. Fiebig, "Operational Guidelines for DNS Transport in Mixed IPv4/IPv6 Environments", Work in Progress, Internet-Draft, draft-ietf-dnsop-3901bis-17, 12 February 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-dnsop-3901bis-17>.

[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-07, 2 March 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-v6ops-6mops-07>.

[I-D.draft-ietf-v6ops-rfc7084bis]
Lencse, G., Martinez, J. P., Patton, B., and T. Winters, "Basic Requirements for IPv6 Customer Edge Routers", Work in Progress, Internet-Draft, draft-ietf-v6ops-rfc7084bis-05, 2 March 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-v6ops-rfc7084bis-05>.

[I-D.draft-palet-v6ops-ipv6-only]
Martinez, J. P., "IPv6-only and IPv6-Mostly Terminology Definitions", Work in Progress, Internet-Draft, draft-palet-v6ops-ipv6-only-14, 7 May 2026, <https://datatracker.ietf.org/doc/html/draft-palet-v6ops-ipv6-only-14>.

[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>.

[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>.

[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>.

[RFC6555] Wing, D. and A. Yourtchenko, "Happy Eyeballs: Success with Dual-Stack Hosts", RFC 6555, DOI 10.17487/RFC6555, April 2012, <https://www.rfc-editor.org/rfc/rfc6555>.

[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>.

[RFC6762] Cheshire, S. and M. Krochmal, "Multicast DNS", RFC 6762, DOI 10.17487/RFC6762, February 2013, <https://www.rfc-editor.org/rfc/rfc6762>.

[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>.

[RFC8489] Petit-Huguenin, M., Salgueiro, G., Rosenberg, J., Wing, D., Mahy, R., and P. Matthews, "Session Traversal Utilities for NAT (STUN)", RFC 8489, DOI 10.17487/RFC8489, February 2020, <https://www.rfc-editor.org/rfc/rfc8489>.

[RFC8504] Chown, T., Loughney, J., and T. Winters, "IPv6 Node Requirements", BCP 220, RFC 8504, DOI 10.17487/RFC8504, January 2019, <https://www.rfc-editor.org/rfc/rfc8504>.

[RFC8656] Reddy, T., Ed., Johnston, A., Ed., Matthews, P., and J. Rosenberg, "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)", RFC 8656, DOI 10.17487/RFC8656, February 2020, <https://www.rfc-editor.org/rfc/rfc8656>.

[RFC9844] Carpenter, B. and R. Hinden, "Entering IPv6 Zone Identifiers in User Interfaces", RFC 9844, DOI 10.17487/RFC9844, August 2025, <https://www.rfc-editor.org/rfc/rfc9844>.

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

[US-NIST.SP.500-267Ar1]
"NIST Special Publication 500-267 Revision 1 - NIST IPv6 Profile", 23 November 2020, <https://nvlpubs.nist.gov/nistpubs/specialpublications/NIST.SP.500-267Ar1.pdf>.

[US-OMB-M-21-07]
"M-21-07 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>.

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

Acknowledgments

Thanks to Holger Fler, Michael Richardson, Tommy Jensen, Nathan Sherrard, Jeremy Duncan, Brian E Carpenter, for the discussions, the input, and all contribution.

Authors' Addresses

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

Jen Linkova
Google
Email: furry13@gmail.com, furry@google.com

Kyle Ouellette
University of New Hampshire Interoperability Labs
Email: kouellette@iol.unh.edu

Ben Patton
University of New Hampshire Interoperability Labs
Email: bpatton@iol.unh.edu