Workgroup: dyncast
Internet-Draft: draft-bormann-t2trg-affinity-00
Published: 30 August 2021
Intended Status: Informational
Expires: 3 March 2022
Authors: C. Bormann
Universität Bremen TZI

Providing Instance Affinity in Dyncast

Abstract

Dyncast support in a network provides a client with a fresh optimal path to a service provider instance, where optimality includes both path and service provider characteristics. As a service invocation usually takes more than one packet, dyncast needs to provide instance affinity for each service invocation. Naive implementations of instance affinity require per-application, per service-invocation state in the network.

The present short document defines a way to provide instance affinity that does not require, but also does not rule out per-application state.

It also discusses how the information that an application needs to operate this mechanism can be provided via the discovery mechanisms offered by a CoRE (Constrained RESTful Environments) server, either in /.well-known/core or via the CoRE resource directory.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Dyncast support in a network provides a client with a fresh optimal path to a service provider instance, where optimality includes both path and service provider characteristics. As a service invocation usually takes more than one packet, dyncast needs to provide instance affinity for each service invocation. Naive implementations of instance affinity require per-application, per service-invocation state in the network.

The present short document defines a way to provide instance affinity that does not require, but also does not rule out per-application state.

It also discusses how the information that an application needs to operate this mechanism can be provided via the discovery mechanisms offered by a CoRE (Constrained RESTful Environments) server, either in /.well-known/core or via the CoRE resource directory.

[I-D.liu-dyncast-ps-usecases] lists use cases of dyncast. The present document does not discuss the specifics of how the network provides dyncast, such as the way service instance metrics enter path computations.

2. Terminology

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.

This document uses the terminology of [I-D.liu-dyncast-ps-usecases], in particular Service and Service Instance (the latter often abbreviated to "Instance"). It also defines the following terms:

Client: The system that requests a service.

Service invocation: A single transaction between client and a service instance. The client is interested in talking to the same service instance throughout one service invocation. Subsequent and parallel service invocations can use different service instances without a problem and therefore do not require affinity.

Instance Affinity: The ability of the network to send all the packets of a service invocation to the same service instance. (Note that this doesn't necessarily imply path affinity -- the client does not care about the path, only about getting to the same service instance.)

Service period: The temporal granularity (rhythm) in which the network updates the optimal paths it provides for a service.

Service stretch: The maximum amount of time that the network plans to provide instance affinity for a service invocation.

3. Assumptions

This document makes a number of assumptions, some of which are fundamental to its technical approach, but some of which are only required for the exposition chosen in this document. A future version of this document will clearly separate these two kinds of assumptions.

Due to experience with overly eager load-based updates to routing metrics, we assume that metrics will be updated on the scale of tens of seconds. To simplify exposition we therefore set the service period to 10 seconds (assumptions of this kind are intended to be possible without loss of generality, but should not be wildly off).

We assume the affinity processing for the entire network will be on a rhythm that is consistent with the service period. Updates take effect at the start of a new service period. The entire network is loosely synchronized on this rhythm. The clients are also aware of this rhythm.

We assume the service stretch will be quite limited, on the order of (a generous) five minutes or less. As a result, any service invocation covers less than 32 service periods. Services that do need longer service stretches will need to renew the service invocation regularly (by checking whether the service instance has changed upon such a renewal, any handover effort needed can be minimized).

Service identifiers take the form of IPv6 addresses, or more typically, IPv6 prefixes. The client is able to complete the prefix with application information. (In a pinch, the client can obtain a complete current address via DNS lookup.)

4. Objectives

Dyncast needs to provide instance affinity. The present document outlines how to achieve this without creating per application, or worse, per invocation state in the network.

The network does not provide any signaling to the clients beyond what is expected in an IPv6 environment.

In summary, the objective of this draft is to define a stable client interface to the instance affinity mechanism (and to motivate why this interface is useful). This interface is designed to remain stable even while the network support for this mechanism is evolving.

5. Approach

We number the service periods with a cyclic numbering system that wraps around about every two service stretches. The network and the clients are aware of the current service period number; the synchronization requirement between them is that clients typically aren't ahead of the network.

When starting a new service invocation, the client builds an IPv6 address out of the service identifier and its view of the current service period number (or it obtains this address using a DNS lookup), essentially filling in 6 bits (for the numbers assumed here). Service requests and the resulting communication within the invocation are addressed to this current address. The client stores the current address with the service invocation when initializing it; it is not ever updated for this invocation.

The network keeps its path optimization state relative to (or indexed by) the current period number. Routing updates can be processed at any time but do not lead to an update of the path optimization state for any service period. The result is that the path chosen after a routing update may no longer be optimal, but that instance affinity is kept. For each service, a pointer for the best service instance is kept for the current and the last 32 service periods.

6. Discussion

The approach presented provides instance affinity without requiring per application or per invocation state in the network. It does require up to 32 copies of what are essentially host routes per service instance. The state scales with the number of service instances, and not with the number of clients.

The approach is based on IPv6. It can be made to work in an IPv4 network, if there are plentiful IPv4 addresses available (see also Section 8).

7. Details

The service period number could simply be inserted in the service identifier, or more complex computation could be performed to make the current addresses generated this way stand out in a forwarding engine.

Naïve clients will start a service invocation with a DNS lookup. This allows the insertion of the period number to be performed in a specialized DNS server for the service. Of course, this requires short time to live (TTL) values and clients that do not on their own cache the look up results.

So the preferred variant is for the client to be aware of the current service period number and to do the insertion by itself on each new service invocation.

8. Legacy IP Considerations

To make this work with IPv4 addresses as service identifiers, we would need 6 bits that can be varied over time. This is likely too expensive for many applications. An alternative approach is to use the port number for the 6 bits. This would mean that the network would need to look up paths both on destination IP address and destination port number (48-bit addressing). For IPv4, this should be good enough.

9. CoRE Discovery

For use with IPv6, this document defines target attributes to enable CoAP clients [RFC7252] to discover the availability of affinity addressing and where in the address it is intended to be applied.

The target attributes are:

* affinity-pos: The starting bit position (counting from most significant bit first) of the sequence of bits where the service period number can be inserted into the IPv6 address given.

*affinity-len: The number of bits of the sequence of bits where the service period number can be inserted into the IPv6 address given.

*affinity-period: The number of seconds a service period spans.

affinity-period is used as a divisor of the synchronized time in seconds, yielding an incremented quotient for the next service period, the lower affinity-len bits are then used as the service period number.

Because of general availability of this time scale, the synchronized time is interpreted according to POSIX [TIME_T]. (POSIX time is also known as "UNIX Epoch time".) Note that leap seconds are handled specially by POSIX time and this results in a 1 second discontinuity several times per decade, which should be of rather limited consequence for service affinity.

Using the example at the end of Section 5 of [RFC6690], a server providing a large resource into a dyncast (anycast) pool could include in its /.well-known/core:

```
REQ: GET /.well-known/core?rt=firmware

RES: 2.05 Content
</firmware/v2.1>;rt="firmware";sz=262144;affinity-pos=122;
affinity-len=6;affinity-period=10
```

(Additional line break for exposition. Obviously, more complex services than simple retrieval of a large object could be offered.)

This link could turn up in a resource directory [I-D.ietf-core-resource-directory] entry that looks like:

```
<coap://[2001:db8:3::123]/firmware/v2.1>;rt="firmware";sz=262144;
affinity-pos=122;affinity-len=6;affinity-period=10
```

Note that the address given here has a number of bits set in the section to be overwritten by the service period number to be inserted.

10. Security Considerations

TBD

11. IANA Considerations

No IANA action is required for this concept draft.

Currently, CoRE target attributes are not subject to registration; this draft defines three new target attributes as per Section 9.

12. References

12.1. Normative References

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/info/rfc7252>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <https://www.rfc-editor.org/info/rfc6690>.

[I-D.ietf-core-resource-directory] Amsüss, C., Shelby, Z., Koster, M., Bormann, C., and P. V. D. Stok, "CoRE Resource Directory", Work in Progress, Internet-Draft, draft-ietf-core-resource-directory-28, 7 March 2021, <https://www.ietf.org/archive/id/draft-ietf-core-resource-directory-28.txt>.

[TIME_T] The Open Group Base Specifications, "Open Group Standard: Vol. 1: Base Definitions, Issue 7", Section 4.16 'Seconds Since the Epoch', IEEE Std 1003.1, 2018 Edition, 2018, <http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap04.html#tag_04_16>.

[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/info/rfc2119>.

[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/info/rfc8174>.

12.2. Informative References

[I-D.liu-dyncast-ps-usecases] Liu, P., Willis, P., and D. Trossen, "Dynamic-Anycast (Dyncast) Use Cases & Problem Statement", Work in Progress, Internet-Draft, draft-liu-dyncast-ps-usecases-01, 15 February 2021, <https://www.ietf.org/archive/id/draft-liu-dyncast-ps-usecases-01.txt>.

Author's Address

Carsten Bormann
Universität Bremen TZI
Postfach 330440
D-28359 Bremen
Germany

Phone: +49-421-218-63921

Email: cabo@tzi.org