Stream: Internet Engineering Task Force (IETF)
RFC: 9992
Category: Standards Track
Published: June 2026
ISSN: 2070-1721
Authors: J. Head, Ed. *HPE*
T. Przygienda *HPE*

RFC 9992
Routing in Fat Trees (RIFT) Key/Value Topology Information Elements Structure and Processing

Abstract

The Routing in Fat Trees (RIFT) protocol allows for key/value pairs to be advertised within Key-Value Topology Information Elements (KV TIEs). The data contained within these KV TIEs can be used for any imaginable purpose.

This document specifies behavior for the various Key Types (i.e., Well-Known, Organizationally Unique Identifier (OUI), and Experimental) and a method to structure corresponding values. It also defines a Well-Known Key Sub-Type used for testing tie-breaking behavior.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9992.

Copyright Notice

Table of Contents

1. Introduction

The Routing in Fat Trees (RIFT) [RFC9692] protocol allows for key/value pairs to be advertised within Key-Value Topology Information Elements (KV TIEs). There are no restrictions placed on the data that is contained in KV TIEs nor what the data is used for.

For example, it might be beneficial to advertise overlay protocol state from leaf nodes to the Top-of-Fabric (ToF) nodes. This would make it possible to view the critical state of a fabric-wide service from a single ToF node rather than retrieving and reconciling the same state from multiple leaf nodes.

1.1. 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. Key-Value Structure

This section describes the generic key structure and semantics; Figure 1 further illustrates these components.

Section 6.1 of [RFC9692] specifies the use of Thrift [THRIFT] to define the protocol's packet structure. While no explicit restrictions are placed on Key-Value data or what it is used for, it is **RECOMMENDED** that a serialized Thrift model also be used to define a KV TIE structure for simpler interoperability. For example, [RIFT-AUTO-EVPN] describes this type of implementation.

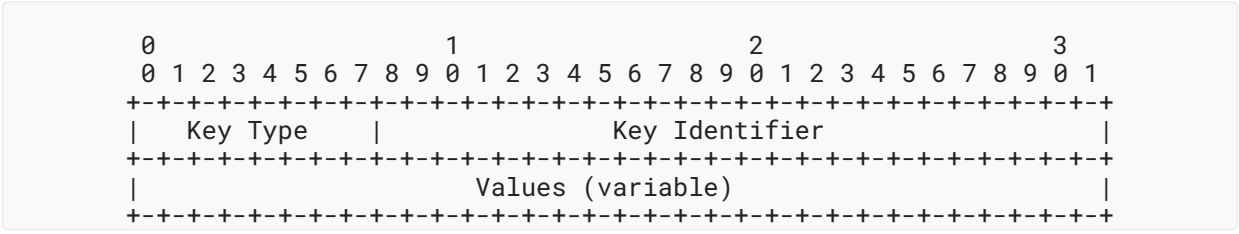

Figure 1: Generic Key-Value Structure

where:

Key Type:
: A 1-byte value that identifies the Key Type. Key Type values are taken from the "RIFTCommonKVTypes" registry defined in [RFC9692].

  The range of valid values is 1 - 255 (2^8-1).

0 is an illegal value and **MUST NOT** be allocated to or used by any implementation. KV TIEs received with this value **MUST** be discarded and logged at least once.

Key Identifier:
A 3-byte value that identifies the specific key and describes the semantics of any contained values. It **SHOULD** be unique within the context of the given Key Type.

The range of valid values is 1 - 16777215 (2^{24}-1).

0 is an illegal value and **MUST NOT** be allocated to or used by any implementation. KV TIEs received with this value **MUST** be discarded and logged at least once.

Values:
A variable length value that contains data associated with the Key Identifier. It **SHOULD** contain 1 or more elements. The semantics (i.e., existence, order, duplication, etc.) of any contained values is governed by the particular key's specification.

2.1. Key Sub-Type

The Key Sub-Type is a mechanism to further describe the key's semantics. This is illustrated by Figure 2. The Key Sub-Type **MUST** be used when the Key Type is either Well-Known or Experimental in order to avoid interoperability issues but is **OPTIONAL** for other Key Types.

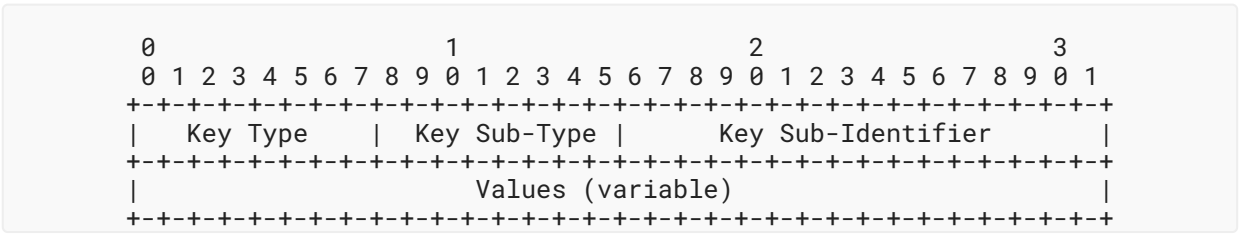

Figure 2: Generic Key-Value Structure with Key Sub-Type

where:

Key Sub-Type:
A 1-byte value that identifies the Key Sub-Type that describes the key and its semantics.

The range of valid values is 1 - 255 (2^{8}-1).

0 is an illegal value and **MUST NOT** be allocated to or used by any implementation. KV TIEs received with this value **MUST** be discarded and logged at least once.

Key Sub-Identifier:
A 2-byte value that identifies the specific key and describes the semantics of any contained values. It **SHOULD** be unique within the context of the given Key Sub-Type.

The range of valid values is 1 - 65535 (2^{16}-1).

0 is an illegal value and **MUST NOT** be allocated to or used by any implementation. KV TIEs received with this value **MUST** be discarded and logged at least once.

2.2. Experimental Key Type

This section describes the Experimental Key Type.

As shown in Figure 3, the Key Type is set to 1, which identifies the Key Type as Experimental. The Experimental Key Type **MUST** support the use of a Key Sub-Type. The Key Sub-Identifier will be used to identify the specific key and the semantics of any contained values.

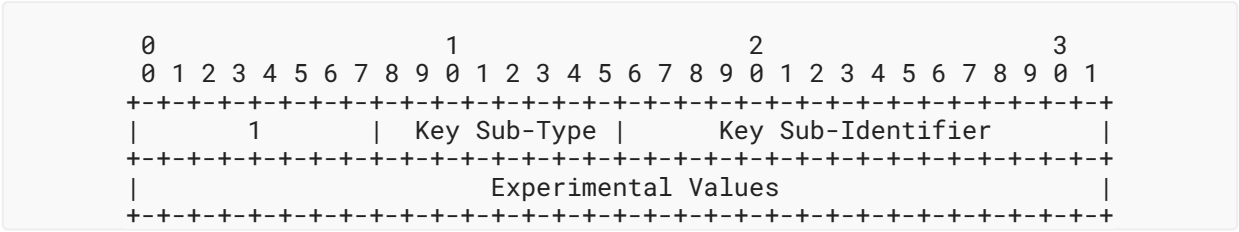

Figure 3: Experimental Key Type

2.3. Well-Known Key Type

This section describes the Well-Known Key Type.

As shown in Figure 4, the Key Type is set to 2, which identifies the Key Type as Well-Known. The Well-Known Key Type **MUST** support the use of a Key Sub-Type. The Key Sub-Identifier will be used to identify the specific key and the semantics of any contained values.

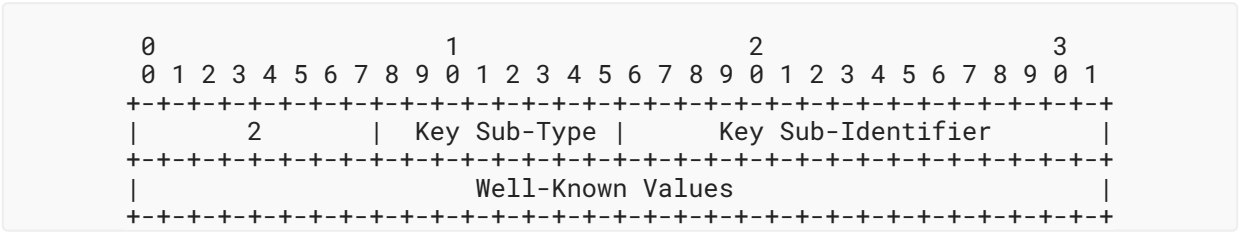

Figure 4: Well-Known Key Type

2.4. OUI Key Type

This section describes the OUI (vendor-specific) Key Type that an implementation **MAY** support.

As shown in Figure 5, the Key Type is set to 3, which identifies the Key Type as OUI. The Key Identifier **MUST** use the implementing organization's reserved OUI [OUI] space to indicate the key and the semantics of any contained values.

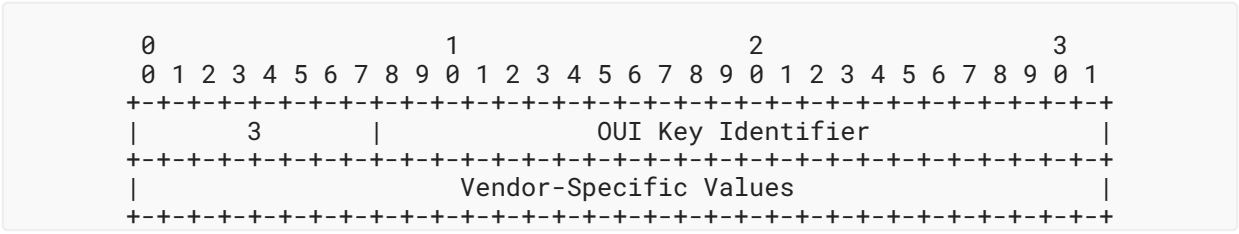

Figure 5: OUI Key Type

3. Design Considerations

> **NOTE:** Like [RFC9692], this section uses terms to denote directionality, specifically, "northbound" meaning "toward the top of the fabric" and "southbound" meaning "toward the bottom of the fabric".

While no explicit restrictions are placed on how Key-Value elements are used, it is of critical importance to consider these details. For example, they should not be used to carry topology information used by RIFT itself to perform distributed computations as it would likely lead to race conditions in convergence, oscillations, and/or other suboptimal behaviors.

It is possible that deployments may have nodes that support a given KV TIE and others that do not. In this scenario, nodes that receive KV TIEs that they don't recognize (e.g., an unknown Key Type) will flood them normally as specified in Section 6.3.4 of [RFC9692].

New Key Types offer 3 bytes of key identification space, and new Well-Known Key Sub-Types offer 2 bytes. When defining how key identification space is used, it is important to consider how much space is actually necessary in order to help ensure efficient use of available registry values.

3.1. Tie-Breaking Considerations

In cases where KV TIEs are flooded southbound, mechanisms **SHOULD** be implemented in order to avoid network-wide flooding where possible. Key Targets (defined in Section 3.2) are one such mechanism.

Section 6.8.5.1 of [RFC9692] specifies that only one KV TIE is selected when identical keys are received from multiple northbound neighbors. Therefore, it is **RECOMMENDED** that implementations ensure that nodes determine Values within KV TIEs independently in a consistent fashion in order to prevent scenarios where multiple ToFs advertise KV TIEs with identical keys but differing Values. In such scenarios, node(s) will select the KV TIE with the highest System ID, which may lead to unintended effects. Even with a robust implementation, operators should also consider that this may still happen under failure conditions, for example, multiple ToFs becoming split-brained.

3.1.1. Southbound KV TIE Tie-Break Sub-Type

This section reserves a Key Sub-Type from the "RIFT Well-Known Key Sub-Types" registry.

This Key-Value pair contains information that allows implementations to test and verify proper tie-breaking behavior for the Southbound Keystore. All implementations **MUST** support this Sub-Type.

All implementations **SHOULD** use the Thrift model defined in Section 3.1.1.1.

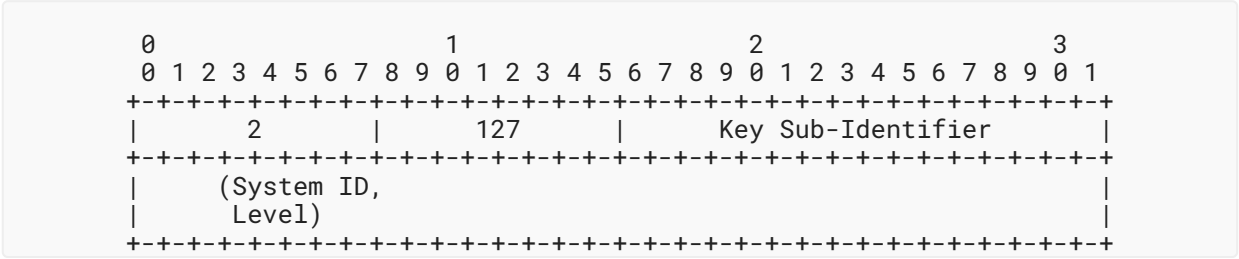

Figure 6: Southbound Tie-Break Sub-Type

where:

System ID:
A **REQUIRED** value indicating the node's unique System ID.

Level:
A **RECOMMENDED** value indicating the node's level.

3.1.1.1. Thrift Models

This section contains the normative Thrift model to support testing southbound Key-Value tie-breaking based on System ID. Per Section 7 of [RFC9692], all signed values **MUST** be interpreted as unsigned values.

```
include "common.thrift"

namespace py southbound_kv
namespace rs models

const i8            GlobalSystemIdentifierKV  = 127

/** simple type to test correct tie-breaking based on system ID */
struct SystemIdentifierKV {
    1:  required   common.SystemIDType         system_id,
    2:  optional   common.LevelType            level,
}
```

Figure 7: RIFT Common Schema for Southbound Key-Value Tie-Break Key Sub-Type

3.2. Key Target

The Key Target is an **OPTIONAL** 64-bit value that identifies group(s) of node(s) that are intended to receive a given KV TIE. Key Targets have a valid range of 0 - 18446744073709551615 (2^{64}-1).

The Thrift model defined in Section 7.2 of [RFC9692] **SHOULD** be used for Key Target implementation.

Figure 8 illustrates the format.

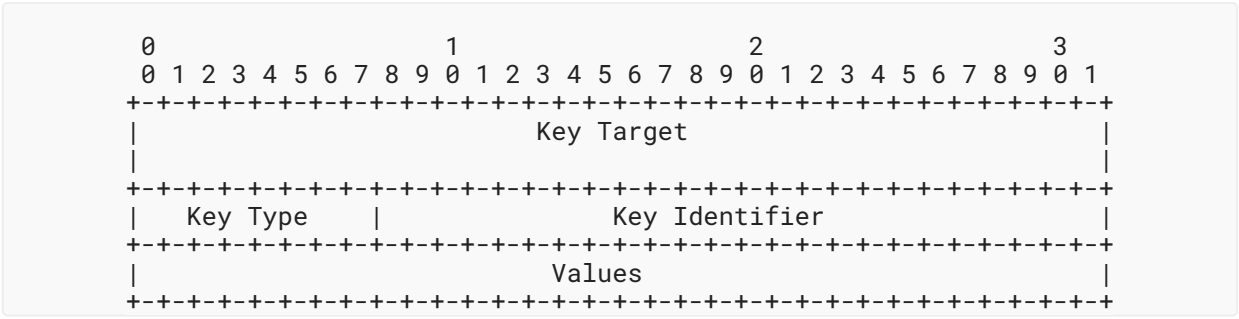

Figure 8: Key Target Format

A value of all 0s indicates that every node is intended to receive this KV TIE and **MUST NOT** be used for any other reason.

A value of all 1s indicates that all leaf nodes are intended to receive this KV TIE and **MUST NOT** be used for any other reason.

Any other value **MUST** be derived from the following normative algorithm. Note that while the algorithm is shown using example code written in [Rust], this document does not mandate the use of any particular language for implementation.

```
<CODE BEGINS>
/// random seeds used in algorithms to increase entropy
pub const RANDOMSEEDS: [UnsignedSystemID; 3] = [
    67438371571u64,
    37087353685,
    88675895388,
];

/// given a system ID delivers the bits set by the according
/// Bloom Filter in the southbound key value target.
pub (crate) fn target2bits(target: UnsignedSystemID) ->
KeyValueTargetType {
    (0 as usize .. 3)
        .map(|s| {
            let rot = (target ^ RANDOMSEEDS[s]).rotate_left(s as _);
            rot.to_ne_bytes().iter().fold(0,
                            |v: u8, nv| v.rotate_right(4) ^ *nv) % 64
        })
        .fold(0, |v, nv| v | (1 << nv))
}

<CODE ENDS>
```

Figure 9: Key Target Standard Algorithm

3.2.1. Key Target Processing

Nodes that support the processing of Key Targets **MUST** only do so on KV TIEs in the southbound direction. Key Targets **MUST NOT** be present on KV TIEs in the northbound direction and are ignored and logged at least once.

Nodes that do not support the processing of Key Targets **MUST** continue to send KV TIEs to all nodes in the appropriate direction. Additionally, Key Targets **MUST** be preserved when KV TIEs are re-originated in the southbound direction.

3.2.1.1. Purging/Rollover

There are several reasons a node may select a different KV TIE. For example, the KV TIE is considered newer due to the sequence number incrementing, a change in the original tie-breaking result between multiple KV TIEs, or a loss of northbound connectivity to the node that advertised the previously selected KV TIE.

Consider a case where Leaf-1, Leaf-2, and Leaf-3 are members of a group of nodes represented by Key Target KT1. If Leaf-2 is removed from that group and a newer instance of the KV TIE needs to be flooded, Leaf-2 will have to maintain the older KV TIE in the Link State Database (LSDB) until the lifetime expires. This could lead to suboptimal behavior in the fabric.

If the new KV TIE being flooded does not include the previous Key Target value, then implementations **SHOULD** flood the newer instance of the KV TIE with a very short lifetime to nodes that belonged to the previous Key Target but not the new Key Target.

4. IANA Considerations

Per [RFC8126], IANA has created the "RIFT Well-Known Key Sub-Types" registry in the "Routing in Fat Trees (RIFT)" registry group at <https://www.iana.org/assignments/rift>.

IANA has updated the "RIFTCommonKVTypes" registry based on values defined in Section 2 of this document, and this document has been added as a reference.

Experts reviewing requests for new values to either the "RIFTCommonKVTypes" registry or the "RIFT Well-Known Key Sub-Types" registry **MUST** consider the items in "Expert Review Guidance" Section 4.2.

4.1. RIFT Well-Known Key Sub-Types

IANA has created the following registry:

Registry Name:
RIFT Well-Known Key Sub-Types

Registration Procedures:
Expert Review

Description:
Well-Known Key Sub-Types registry for the RIFT protocol.

Reference:
RFC 9992

4.1.1. RIFT Well-Known Key Sub-Types Entries

IANA has registered the following values in the "RIFT Well-Known Key Sub-Types" registry.

Value	Name	Description	Reference
0	Illegal	Not allowed.	RFC 9992
1-126	Unassigned		
127	Southbound Tie-Break Sub-Type	Used for testing/verifying Southbound Keystore tie-breaking behavior.	RFC 9992
128-255	Unassigned		

Table 1: RIFT Well-Known Key Sub-Types Entries

4.2. Expert Review Guidance

Experts reviewing requests for values from the "RIFTCommonKVTypes" registry or the "RIFT Well-Known Key Sub-Types" registry are responsible for the following:

1. Ensuring that the supporting documentation accompanying the request properly defines how Key Identifiers and/or Key Sub-Identifiers are used (e.g., as a boolean, an explicit value, an auto-derived value, etc.).
2. Ensuring that the supporting documentation provides normative Thrift model(s) (if applicable).
3. Ensuring that any work originating outside the IETF does not conflict with any work that is already published or in active pursuit of being published.

5. Security Considerations

This document introduces no new security concerns to RIFT or other specifications referenced in this document given that the KV TIEs are already extensively secured by the RIFT [RFC9692] protocol specification itself.

6. References

6.1. Normative References

[OUI] IEEE, "Guidelines for Use of Extended Unique Identifier (EUI), Organizationally Unique Identifier (OUI), and Company ID (CID)", <https://standards-support.ieee.org/hc/en-us/articles/4888705676564-Guidelines-for-Use-of-Extended-Unique-Identifier-EUI-Organizationally-Unique-Identifier-OUI-and-Company-ID-CID>.

[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>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[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>.

[RFC9692] Przygienda, T., Ed., Head, J., Ed., Sharma, A., Thubert, P., Rijsman, B., and D. Afanasiev, "RIFT: Routing in Fat Trees", RFC 9692, DOI 10.17487/RFC9692, April 2025, <https://www.rfc-editor.org/info/rfc9692>.

[Rust] Rust Foundation, "The Rust Reference", <https://doc.rust-lang.org/reference/>.

[THRIFT] Apache Software Foundation, "Thrift Language Implementation and Documentation", commit 66d8976, <https://github.com/apache/thrift/tree/0.15.0/doc>.

6.2. Informative References

[RIFT-AUTO-EVPN] Head, J., Przygienda, T., and W. Lin, "RIFT Auto-EVPN", Work in Progress, Internet-Draft, draft-ietf-rift-auto-evpn-06, 8 July 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-rift-auto-evpn-06>.

Acknowledgements

Thanks to Italo Busi for his very thoughtful review that yielded an improved spec.

Authors' Addresses

Jordan Head (EDITOR)
HPE
1701 East Mossy Oaks Road
Spring, TX 77389
United States of America
Email: jordan.head@hpe.com

Tony Przygienda
HPE
1701 East Mossy Oaks Road
Spring, TX 77389
United States of America
Email: antoni.przygienda@hpe.com