Internet Draft

Puneet Agarwal
Pluris
Bora A. Akyol
Cisco Systems

Document: draft-ietf-mpls-ttl-02.txt
Category: Informational
Expires: November 2002

May 2002

Time to Live (TTL) Processing in MPLS Networks

Status of this Memo

Abstract

This document describes TTL processing in hierarchical MPLS networks. TTL processing in both pipe and uniform model hierarchical tunnels are specified with examples for both "push" and "pop" cases. The document also complements rfc-3270 "MPLS Support of Differentiated Services" and ties together the terminology introduced in that document with TTL processing in hierarchical MPLS networks.

Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC-2119].

1. Introduction and Motivation

This document describes Time to Live (TTL) processing in hierarchical MPLS networks. We believe that this document adds details that have not been addressed in [MPLS-ARCH, MPLS-ENCAPS], and that the methods presented in this document complement [MPLS-DS].

2. TTL Processing in MPLS Networks

2.1. Changes to RFC 3032 [MPLS-ENCAPS]

a) [MPLS-ENCAPS] only covers the Uniform Model and does NOT address the Pipe Model or the Short Pipe Model. This draft will address these 2 models and for completeness will also address the Uniform Model.
b) [MPLS-ENCAPS] does not cover hierarchical LSPs. This draft will address this issue.
c) [MPLS-ENCAPS] does not define TTL processing in the presence of Penultimate Hop Popping (PHP). This draft will address this issue.

2.2. Terminology and Background

As defined in [MPLS-ENCAPS], MPLS packets use a MPLS shim header that indicates the following information about a packet:

a. MPLS Label (20 bits)
b. TTL (8 bits)
c. Bottom of stack (1 bit)
d. Experimental bits (3 bits)

The experimental bits were later redefined in [MPLS-DS] to indicate the scheduling and shaping behavior that could be associated with a MPLS packet.

[MPLS-DS] also defined two models for MPLS tunnel operation: Pipe and Uniform models. In the Pipe model, a MPLS network acts like a circuit when MPLS packets traverse the network such that only the LSP ingress and egress points are visible to nodes that are outside the tunnel. A Short variation of the Pipe Model is also defined in [MPLS-DS] to differentiate between different egress forwarding and QoS treatments. On the other hand, the Uniform model makes all the

nodes that a LSP traverses visible to nodes outside the tunnel. We will extend the Pipe and Uniform models to include TTL processing in the following sections. Furthermore, TTL processing when performing PHP is also described in this document. For a detailed description of Pipe and Uniform models, please see [MPLS-DS].

TTL processing in MPLS networks can be broken down into two logical blocks: (i) the incoming TTL determination to take into account any

tunnel egress due to MPLS Pop operations; (ii) packet processing of (possibly) exposed packet & outgoing TTL.

We also note here that signaling the LSP type (pipe, short pipe or uniform model) is out of the scope of this document, and that is also not addressed in the current versions of the label distribution protocols, e.g. LDP [MPLS-LDP] and RSVP-TE [MPLS-RSVP].

2.3. New Terminology

iTTL: The TTL value to use as the incoming TTL. No checks are performed on the iTTL.

oTTL: This is the TTL value used as the outgoing TTL value (see section 3.5 for exception). It is always (iTTL - 1) unless otherwise stated.

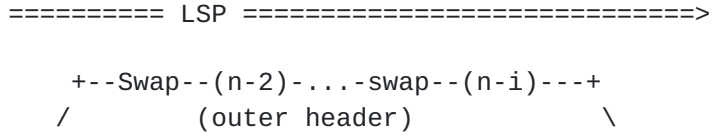
oTTL Check: Check if oTTL is greater than 0. If the oTTL Check is false, then the packet is not forwarded. Note that the oTTL check is performed only if any outgoing TTL (either IP or MPLS) is set to oTTL (see section 3.5 for exception).

3. TTL Processing in different Models

This section describes the TTL processing for LSPs conforming to each of the 3 models (Uniform, Short Pipe and Pipe) in the presence/absence of PHP (where applicable).

3.1. TTL Processing for Uniform Model LSPs (with or without PHP)

(consistent with [MPLS-ENCAPS]):



```
            (n-1)                                   (n-i)
            /                                          \
   >--(n)--Push..............(x).....................Pop--(n-i-1)->
           (I)           (inner header)             (E or P)
```

(n) represents the TTL value in the corresponding header
(x) represents non-meaningful TTL information
(I) represents the LSP ingress node
(P) represents the LSP penultimate node
(E) represents the LSP Egress node

This picture shows TTL processing for a uniform model MPLS LSP. Note that the inner and outer TTLs of the packets are synchronized at tunnel ingress and egress.

3.2. TTL Processing for Short Pipe Model LSPs

3.2.1. TTL Processing for Short Pipe Model LSPs without PHP

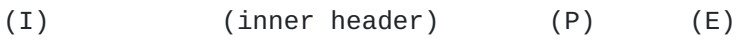
```
            ========== LSP ============================>

                +--Swap--(N-1)-...-swap--(N-i)-----+
               /        (outer header)              \
             (N)                                   (N-i)
             /                                        \
   >--(n)--Push..............(n-1).....................Pop--(n-2)->
           (I)           (inner header)                (E)
```

(N) represents the TTL value (may have no relationship to n)
(n) represents the tunneled TTL value in the encapsulated header
(I) represents the LSP ingress node
(E) represents the LSP Egress node

Short Pipe Model was introduced in [MPLS-DS]. In the short pipe model, the forwarding treatment at the egress LSR is based on the tunneled packet as opposed to the encapsulating packet.

3.2.2. TTL Processing for Short Pipe Model with PHP:

```
            ========== LSP ====================================>
                +-Swap-(N-1)-...-swap-(N-i)-+
               /       (outer header)        \
             (N)                            (N-i)
             /                                \
   >--(n)--Push............(n-1)............Pop-(n-1)-Decr.-(n-2)->
```

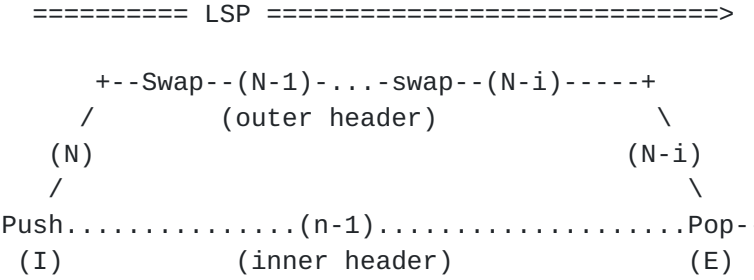


(N) represents the TTL value (may have no relationship to n)
(n) represents the tunneled TTL value in the encapsulated header
(I) represents the LSP ingress node
(P) represents the LSP penultimate node
(E) represents the LSP egress node.

Since the label has already been popped by the LSPÆs penultimate node, the LSP egress node just decrements the header TTL.

Also note that at the end of short pipe model LSP, the TTL of the tunneled packet has been decremented by two either with or without PHP.

3.3. TTL Processing for Pipe Model LSPs (without PHP only):



(N) represents the TTL value (may have no relationship to n)
(n) represents the tunneled TTL value in the encapsulated header
(I) represents the LSP ingress node
(E) represents the LSP Egress node

From the TTL perspective, the treatment for a Pipe Model LSP is identical to the Short Pipe Model without PHP.

3.4. Incoming TTL (iTTL) determination

If the incoming packet is an IP packet, then the iTTL is the TTL value of the incoming IP packet.

If the incoming packet is a MPLS packet and we are performing a

Push/Swap/PHP, then the iTTL is the TTL of the topmost incoming label.

If the incoming packet is a MPLS packet and we are performing a Pop (tunnel termination), the iTTL is based on the tunnel type (Pipe or Uniform) of the LSP that was popped. If the popped label belonged to a Pipe model LSP, then the iTTL is the value of the TTL field of the header exposed after the label was popped (note that for the purpose of this draft, the exposed header may be either an IP header or an MPLS label). If the popped label belonged to a Uniform model LSP, then the iTTL is equal to the TTL of the popped label. If multiple Pop operations are performed sequentially, then the procedure given above is repeated with one exception: the iTTL computed during the previous Pop is used as the TTL of subsequent label being popped; i.e. the TTL contained in the subsequent label is essentially ignored and replaced with the iTTL computed during the previous pop.

3.5. Outgoing TTL Determination and Packet Processing

After the iTTL computation is performed, the oTTL check is performed. If the oTTL check succeeds, then the outgoing TTL of the (labeled/unlabeled) packet is calculated and packet headers are updated as defined below.

If the packet was routed as an IP packet, the TTL value of the IP packet is set to oTTL (iTTL - 1). The TTL value(s) for any pushed label(s) are determined as described in section 3.6.

For packets that are routed as MPLS, we have four cases:

1) Swap-only: The routed label is swapped with another label and the TTL field of the outgoing label is set to oTTL.

2) Swap followed by a Push: The swapped operation is performed as described in (1). The TTL value(s) of any pushed label(s) are determined as described in section 3.6.

3) Penultimate Hop Pop (PHP): The routed label is popped. The oTTL check should be performed irrespective of whether the oTTL is used to update the TTL field of the outgoing header. If the PHPed label belonged to a short Pipe model LSP, then the TTL field of the PHP exposed header is neither checked nor updated. If the PHPed label was a Uniform model LSP, then the TTL field of the PHP exposed header is set to the oTTL. The TTL value(s) of additional labels are determined

as described in section 3.6

4) Pop: The pop operation happens before routing and hence it is not considered here.

3.6. Tunnel Ingress Processing (Push)

For each pushed Uniform model label, the TTL is copied from the label/IP-packet immediately underneath it.

For each pushed Pipe model or Short Pipe model label, the TTL field is set to a value configured by the network operator. In most implementations, this value is set to 255 by default.

3.7. Implementation Remarks

1) Although iTTL can be decremented by a value larger than 1 while it is being updated or oTTL is being determined, this feature should be only used for compensating for network nodes that are not capable of decrementing TTL values.

2) Whenever iTTL is decremented, the implementer must make sure that the value does not go negative.

3) In the short pipe model with PHP enabled, the TTL of the tunneled packet is unchanged after the PHP operation.

4. Conclusion

This Internet Draft describes how TTL field can be processed in a MPLS network. We clarified the various methods that are applied in the presence of hierarchical tunnels and completed the integration of Pipe and Uniform models with TTL processing.

5. Security Considerations

This document does not add any new security issues other than the ones defined in [MPLS-ENCAPS, MPLS-DS]. In particular, the document does not define a new protocol or expand an existing one and does not introduce security problems into the existing protocols. The authors believe that clarification of TTL handling in MPLS networks benefits service providers and their customers since troubleshooting

is simplified.

6. References

[MPLS-ARCH] E. Rosen, A. Viswanathan, R. Callon, "Multiprotocol Label Switching Architecture", RFC 3031.

[MPLS-ENCAPS] E. Rosen, D. Tappan, G. Fedorkow, Y. Rekhter, D. Farinacci, T. Li, A. Conta, "MPLS Label Stack Encoding", RFC3032.

[MPLS-DS] F. Le Faucheur, L. Wu, B. Davie, S. Davari, P. Vaananen, R. Krishnan, P. Cheval, J. Heinanen, "MPLS Support of Differentiated Services", RFC3270.

[MPLS-LDP] L. Andersson, P. Doolan, N. Feldman, A. Fredette, B. Thomas, "LDP Specification", RFC 3036.

[MPLS-RSVP] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, G. Swallow, "RSVP-TE: Extensions to RSVP for LSP Tunnels", RFC 3209.

7. Acknowledgements

The authors would like to thank the members of the MPLS working group for their feedback. We would especially like to thank Shahram Davari and Loa Andersson for their careful review of the document and their comments.

8. Author's Addresses

Puneet Agarwal
Pluris
10455 Bandley Drive
Cupertino, CA 95014
Email: puneet@pluris.com

Bora Akyol
Cisco Systems
170 W. Tasman Drive

San Jose, CA 95134
Email: bora@cisco.com