Routing in Fat Trees (RIFT)

Update

IETF 105

Tony Przygienda, Editor, Juniper

Published -06, Repo -07 Changes vs. -05

All RIFT routers MUST support IPv4 forwarding and MAY support IPv6 forwarding. A three way adjacency over IPv6 addresses implies support for IPv4 forwarding.

All RIFT routers MUST support IPv4 forwarding and MAY support IPv6 forwarding. A three way adjacency over IPv6 addresses implies support for IPv4 forwarding. A node that does not process received IPv6 LIEs MUST NOT originate IPv6 LIEs.

5.4.3. Security Envelope

RIFT MUST be carried in a mandatory secure envelope illustrated in Figure 31. Local configuration can allow to skip the checking of the envelope's integrity.

5.4.3. Security Envelope

RIFT MUST be carried in a mandatory secure envelope illustrated in Figure 31. Any value in the packet following a security fingerprint MUST be used only after the according fingerprint has been validated.

Any implementation including RIFT security MUST generate and wrap around local nonces properly. All implementation MUST reflect the neighbor's nonces. An implementation SHOULD increment a chosen nonce on every LIE FSM transition that ends up in a different state from

Any implementation including RIFT security MUST generate and wrap around local nonces properly. When a nonce increment leads to `undefined_nonce` value the value SHOULD be incremented again immediately. All implementation MUST reflect the neighbor's nonces.

As optional optimization, an implemenation MAY send one LIE with previously negotiated neighbor's nonce to try to speed up a neighbor's transition from 3-way to 1-way and MUST revert to sending `undefined_nonce` after that.

Published -06, Repo -07 Changes vs. -05

```
5.4.5.  Lifetime

   Protecting lifetime on flooding can lead to excessive number of
   security fingerprint computation and hence an application generating
   such fingerprints on TIEs SHOULD round the value down to the next
   `rounddown_lifetime_interval` defined in the schema when sending
   TIEs.
```

```
5.4.5.  Lifetime

   Protecting lifetime on flooding may lead to excessive number of
   security fingerprint computation and hence an application generating
   such fingerprints on TIEs MAY round the value down to the next
   `rounddown_lifetime_interval` defined in the schema when sending TIEs
   albeit such optimization in presence of security hashes over
   advancing weak nonces may not be feasible.
```

```
struct LinkIDPair {
    /** node-wide unique value for the local link */
    1: required common.LinkIDType      local_id;
    /** received remote link ID for this link */
    2: required common.LinkIDType      remote_id;

    /** optionally describes the local interface index of the link */
   10: optional common.PlatformInterfaceIndex       platform_interface_index;
   /** optionally describes the local interface name */
   11: optional string                              platform_interface_name;
```

```
struct LinkIDPair {
    /** node-wide unique value for the local link */
    1: required common.LinkIDType      local_id;
    /** received remote link ID for this link */
    2: required common.LinkIDType      remote_id;

    /** optionally describes the local interface index of the link */
   10: optional common.PlatformInterfaceIndex       platform_interface_index;
   /** optionally describes the local interface name */
   11: optional string                              platform_interface_name;
   /** optional indication whether the link is secured, i.e. protected by outer key, absence
       of this element means no indication, undefined outer key means not secured */
   12: optional common.OuterSecurityKeyID           trusted_outer_security_key;
```

```
/** Header of a TIE as described in TIRE/TIDE.
*/
struct TIEHeaderWithLifeTime {
    1: required     TIEHeader                       header;
    /** remaining lifetime that expires down to 0 just like in ISIS.
        TIEs with lifetimes differing by less than `lifetime_diff2ignore` MUST
        be considered EQUAL. */
    2: required     common.LifeTimeInSecType        remaining_lifetime;
}
```

```
struct PrefixAttributes {
    2: required common.MetricType           metric = common.default_distance;
    /** generic unordered set of route tags, can be redistributed to other protocols or use
        within the context of real time analytics */
    3: optional set<common.RouteTagType>     tags;
    /** optional monotonic clock for mobile addresses */
    4: optional common.PrefixSequenceType    monotonic_clock;
    /** optionally indicates the interface is a node loopback */
    6: optional bool                         loopback = false;
    /** indicates that the prefix is directly attached, i.e. should be routed to even if
        the node is in overload. **/
    7: optional bool                         directly_attached = true;
```

Rounds of Juniper vs. Python-RIFT Interop with Security Envelope

- Python-RIFT has full single plane implementation now
- 0.11 Juniper about to be released (-07 draft implementation)
- 0.11 security envelope interop concluded
- 0.11 Juniper will be released to public
  - Will include specification of
    - Configuration API
    - Operational state API
    - Real-time analytics API

Early Directorate Review & Discussions

- Discussion on router requirements for RIFT
  - Phrasing requiring v4 forwarding support will be removed
  - Needs addition to LIE in schema to indicate whether v4 forwarding is supported
- Review came in
  - Mostly editorial/better wording
- Review concluded, results posted, will be published on -07 after IETF

THANK YOU FOR YOUR ATTENTION