Routing in Fat Trees (RIFT) Interim

Status

6/6/19

Tony Przygienda, Juniper

Text changes in ’about to be published’ -06 Draft

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.

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.

Schema changes 1 in 'about to be published ' -06 Draft

```
/** Maximum delta (negative or positive) that a mirrored nonce can
    deviate from local value to be considered valid. If nonces are
    changed every minute on both sides this opens statistically
    a `maximum_valid_nonce_delta` minutes window of identical LIEs,
    TIE, TI(x)E replays.
    The interval cannot be too small since LIE FSM may change
    states fairly quickly during ZTP without sending LIEs*/
const i16              maximum_valid_nonce_delta   = 5;
```

```
/** Maximum delta (negative or positive) that a mirrored nonce can
    deviate from local value to be considered valid. If nonces are
    changed every minute on both sides this opens statistically
    a 3 minutes window of identical LIEs, TIE, TI(x)E replays **/
const i16              maximum_valid_nonce_delta   = 3;
```

```
struct TIEHeader {
    2: required TIEID                           tieid;
    3: required common.SeqNrType                seq_nr;

    /** optional absolute timestamp when the TIE
        was generated. This can be used on fabrics with
        synchronized clock to prevent lifetime modification attacks. */
   10: optional common.IEEE802_1ASTimeStampType   origination_time;
   /** optional original lifetime when the TIE
       was generated. This can be used on fabrics with
       synchronized clock to prevent lifetime modification attacks. */
   12: optional common.LifeTimeInSecType          origination_lifetime;
}
```

```
struct TIEHeader {
    2: required TIEID                           tieid;
    3: required common.SeqNrType                seq_nr;
    /** remaining lifetime that expires down to 0 just like in ISIS.
        TIEs with lifetimes differing by less than `lifetime_diff2ignore` MUST
        be considered EQUAL.

        When using security envelope,
        this is just a model placeholder for convienence
        that is never being modified during flooding. The real remaining lifetime
        is contained on the security envelope. */
    4: required common.LifeTimeInSecType          remaining_lifetime;
    /** optional absolute timestamp when the TIE
        was generated. This can be used on fabrics with
        synchronized clock to prevent lifetime modification attacks. */
   10: optional common.IEEE802_1ASTimeStampType   origination_time;
   /** optional original lifetime when the TIE
       was generated. This can be used on fabrics with
       synchronized clock to prevent lifetime modification attacks. */
   12: optional common.LifeTimeInSecType          origination_lifetime;
}
```

Schema changes 2 in ’about to be published’ -06 Draft

```
/** 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;
```

```
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;
```

'About to be published' Applicability Draft

- In the usual repo
  - https://bitbucket.org/riftrfc/rift_draft/src/master/draft-rift-applicability-00.xml

‘About to happen’ Juniper vs. Python-RIFT Interop with Security Envelope

- Python-RIFT has full single plane implementation now
- 0.11 Juniper about to be released (-06 draft implementation)
- Python-RIFT will need minimal update to new schema changes
- Interop planned off-line before IETF
- That would be then two full RIFT implementations for single plane
- After interop 0.11 Juniper will be released to public
  - Will include specification of
    - Configuration API
    - Operational state API
    - Real-time analytics API

THANK YOU FOR YOUR ATTENTION