#### **LISP: Intro and Update**

RIPE Berlin - May, 2008

Vince Fuller (for Dino, Dave, Darrel, et al)

http://www.vaf.net/prezos/lisp-ripe-short.pdf

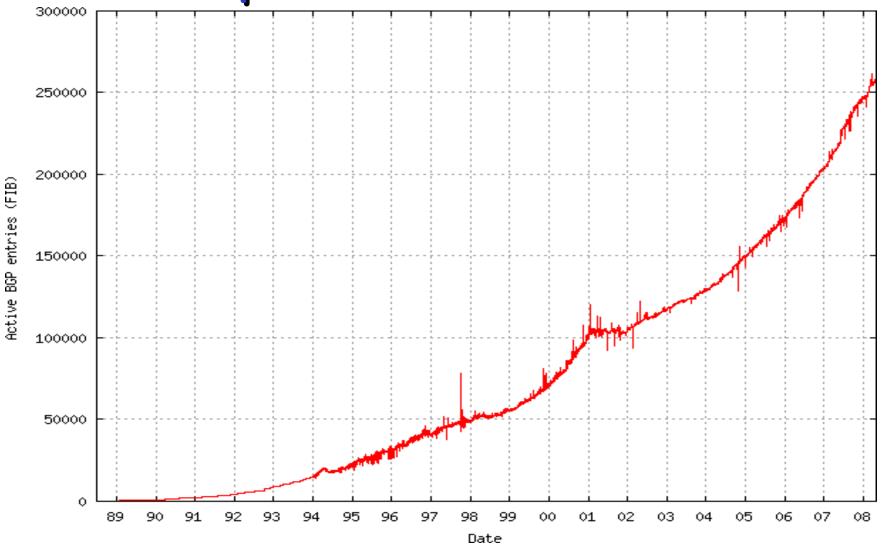


- What is LISP?
- What problem is LISP solving?
  - www.vaf.net/prezos/rrg-prague.pdf
- Why Locator/ID Separation?
- Data Plane Operation
- Finding Mappings LISP+ALT
- Incremental Deployability
- Implementation and Testing status

### What is LISP?

- Locator/ID Separation Protocol
- Ground rules for LISP
  - Network-based solution
  - No changes to hosts whatsoever
  - No new addressing changes to site devices
  - Very few configuration file changes
  - Imperative to be incrementally deployable
  - Address family agnostic

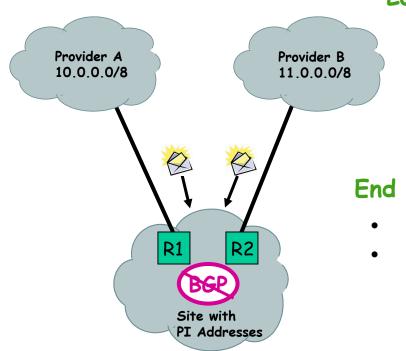
#### A picture is worth...



## What is ID/Loc separation?

- Instead of IP addresses, two numbering spaces:
- Endpoint Identifiers (EIDs): hierarchically assigned to sites along administrative lines (like DNS hostnames)
  - do not change on devices that remain associated with the site; think "PI" but not routable
- Routing Locators (RLOCs): assigned according to network topology, like "PA" address assignments
  - Locators are aggregated/abstracted at topological boundaries to keep routing state scalable
  - When site's connection to network topology changes, so do the locators – aggregation is preserved

## What Features do I get?



#### Lower OpEx for Sites and Providers

- Improve site multi-homing
- Improve provider traffic engineering
- Reduce size of core routing tables

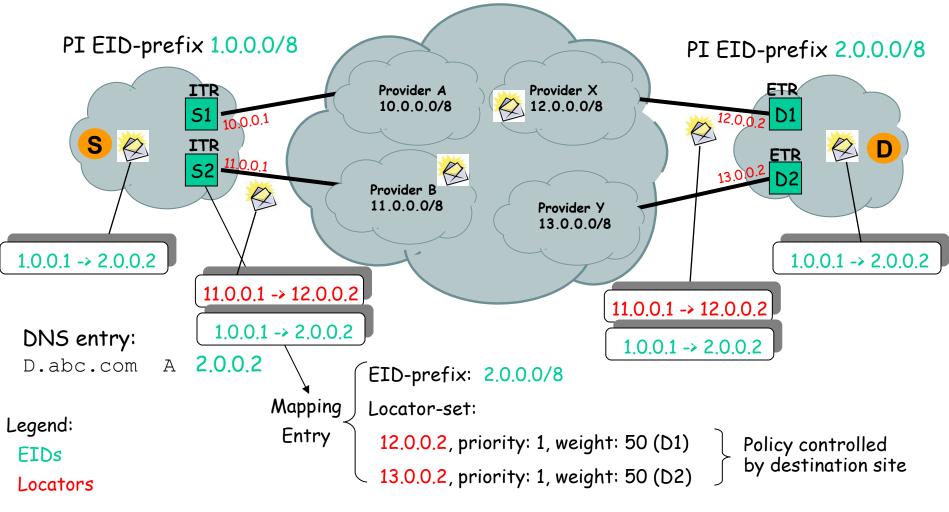
#### End Site Benefit

- Easier Transition to ipv6
- Change provider without address change

### New Network Elements

- Ingress Tunnel Router (ITR)
  - Finds EID to RLOC mapping
  - Encapsulates to Locators at source site
- Egress Tunnel Router (ETR)
  - Owns EID to RLOC mapping
  - Decapsulates at destination site

## Packet Forwarding



LISP: What and Why

RIPE Berlin, May, 2008

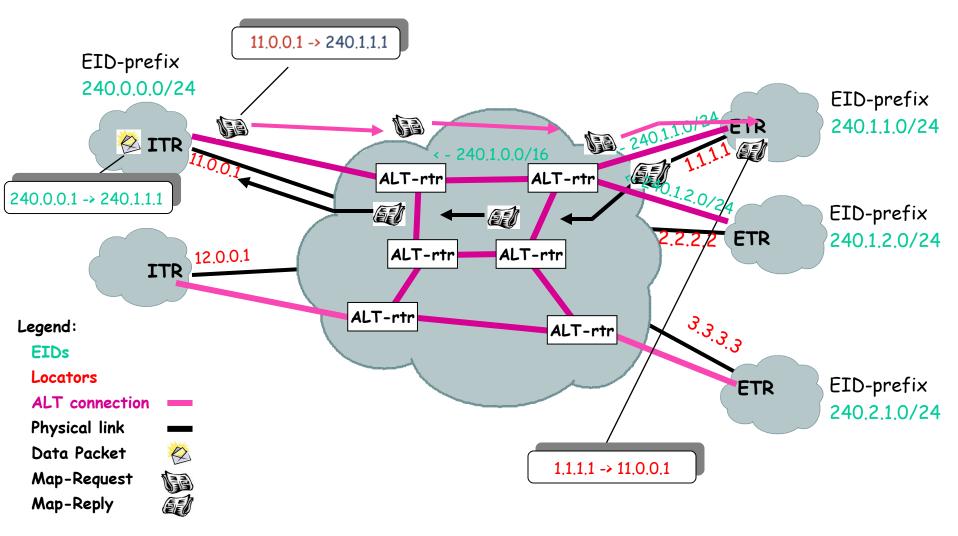
# When the ITR has no Mapping

- ITR needs to obtain from ETR
- ITR sends Map Request (or Data Probe)
- ETR returns Map Reply
- But how do the ITR and ETR hook up?

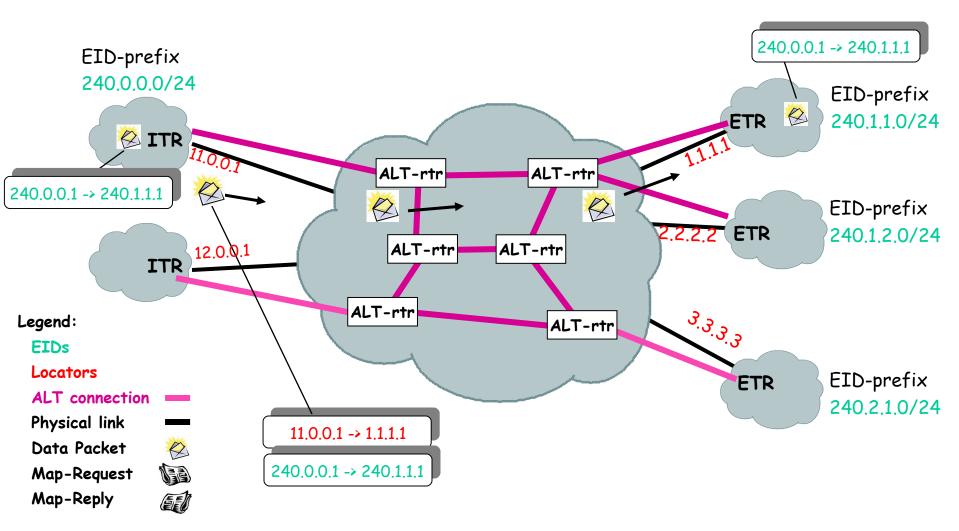
# Finding an ETR: LISP+ALT

- Hybrid push/pull approach
  - ALT pushes aggregates, LISP pulls specifics
- Hierarchical EID prefix assignment
- Aggregation of EID prefixes
- GRE-based overlay network
- BGP used to advertise EIDs on overlay
- Option for data-triggered Map-Replies

### LISP+ALT in action



### LISP+ALT in action



# Interworking Deployability

- These combinations must be supported
  - Non-LISP site to non-LISP site
    - Today's Internet
  - LISP site to LISP site
    - Encapsulation over IPv4 makes this work
    - IPv4-over-IPv4 or ipv6-over-IPv4
  - LISP-R site to non-LISP site
    - When LISP site has PI or PA routable addresses
  - LISP-NR site to non-LISP site
    - When LISP site has PI or PA non-routable addresses

# Interworking Deployability

- LISP-R site to non-LISP site
  - ITR at LISP site detects non-LISP site when no mapping exists
    - Does not encapsulate packets
  - Return packets to LISP site come back natively since EIDs are routable
  - Same behavior as the non-LISP to non-LISP case
    - LISP site acts as a non-LISP site

# Interworking Deployability

- LISP-NR site to a non-LISP site
  - ITR at LISP site detects non-LISP site when no mapping exists
    - Does not encapsulate packets
  - For return packets to LISP site
    - ITR translates to a source routable address so packets symmetrically sent natively
    - PTR advertises NR prefixes close to non-LISP sites so return packets are encapsulated to ETR at LISP site

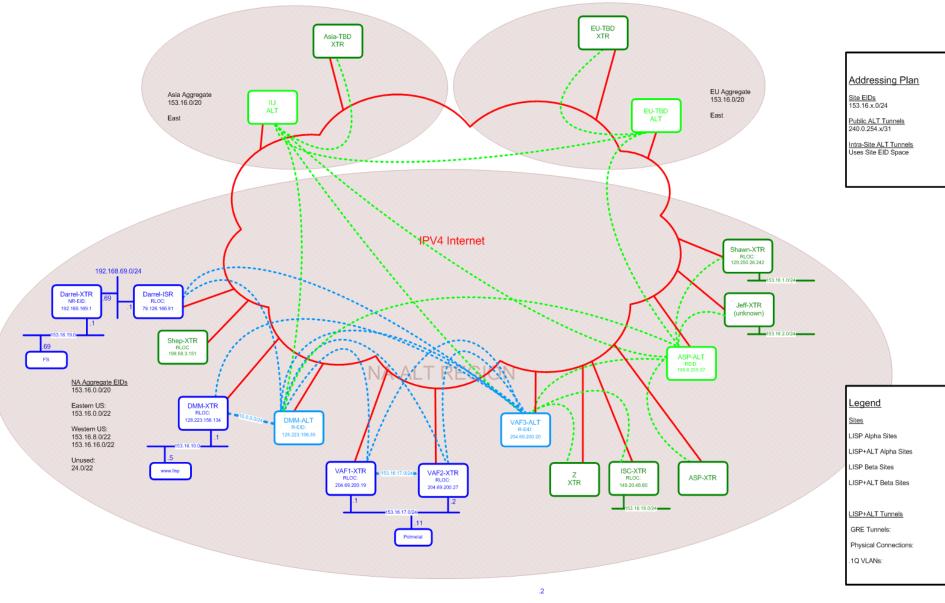
## **Implementation Status**

- cisco has a LISP prototype implementation
  - Started the week of IETF Prague (March 2007)
- OS platform is NX-OS (on top of Linux)
- Hardware platform is Titanium
  - 1 RU dual-core off-the-shelf PC with 7 GEs
- **Based on** draft-farinacci-lisp-07.txt
- Software switching only
- Supports both IPv4 and ipv6
- Includes ALT and Interworking

## **Implementation Status**

- IOS 12.4T prototype is in the works
- OpenLISP implementation draft-iannone-openlisp-implementation-00.txt
- Would really like to see more

#### LISP and LISP+ALT Test Topology



# Wanna play with us?

- Looking for more external test sites
  - Particular need in European region
  - Must be able to dedicate minimum of 1 day a week
- Goals:
  - Test multiple implementations
  - Experience with operational practices
  - Learn about revenue making opportunities
- Contact us: lispers@cisco.com
- See also: lisp-interest@lists.civil-tongue.net

#### **Internet Drafts**

- draft-farinacci-lisp-07.txt
- draft-fuller-lisp-alt-02.txt
- draft-lewis-lisp-interworking-00.txt

draft-farinacci-lisp-multicast-00.txt

```
(defun changer-one (failures database)
                                                                       (defun cbr-top (source destination plan path prev-plan)
                                                                          (setf chosen-path (cbr-evaluate source destination
-f-p-helper (condin((nulltefailures) database))
                                                                                                  (cbr-retrieve source destina
failure-poi(Set priist partoi (process-one (car failures) database)) (cbr-new-plan chosen-path plan))
    (append' (list (list index
          (setf (second part 2) (process-two (car failures), first-part))
(t t))
          (cond ((null (cdr failures)) second-part)
                                                                        ;; we've reached the conclusion of CBR - return the path
                (de(from concertante in the last be (cor failures) second-part)) (core (cond) ((cbr-exact-match (list source destination) plan
                    (cond ((nu
                                                                               (append path (cbr-get-path (list source destinat
                        ((eq
                                                     263.53
                                                                               ((cbr-exact-match (list source destination) prev
                                  st (caasr pa
                                               (first
                                                    dadar path)))
                                                                               (append path (cbr-get-path (list source destinat
                                    nvert-outp
                                              helper-
                                                      rest path))))
                        ((eq
                                     ath) (ca
                                              Redad
                                                     ath)))
                               (cal
                                                    (cdadar path)))
                                                                        ;; If we have no chosen-path, we can't recurse - we retu
   (defun pr
                ess-one
                             (pallay
                                              haledr-b (rest path))))
                  (null'eg
                                              (cdadar path)))
                             tapaser
        (cond
                                                                               ((null chosen-path) nil)
                                               (first (cdadar path)))
                                             (cdadar path))
                                                                        ;; If either the two elements of the chosen-path match,
secute-plan-helper (source pach 144 uparanidast-
                                      rth)*(ca
d ((or (null source) (null path) (aqual (14 reverseth) (car pair) +h) (caar database he)t of the chosen-path matches the destina
    (append (list (append path)) st (append (list (caar database)))
 ((equal fail-param 'F)
                     (list 'UNADACANON (remove-pair (cadr pair) (cdar datadaase)) ds) chosen-path) (cadr chosen-path))
     (execute-plan-helper source (rest path) 'F)))
                                                                               ((equal (cadr chosen-path) destination) nil)
 ((check-failure (first path) *failure-poipadr database)))
    (append (list (append (list (first par))) st (car d
                                                       se))
                                                                                  sequenti
                                                                                                               wavs
                                                                                              second e
                                                                                                        hent of chosen-path as
                                                                                 urse with
     (execute-plan-helper source (rest paterocess-one
                                                  pai:
                                                                databas
                                                         (cd:
                                                                                 usting the
                                                                                             w-plan, p
                                                                                                        , and prev-plan accor
 (t (append (list (append (list (first path))
                                                                                  as apple
                    (list 'FAILURE)))
                                                                                                          te-matteh of der
     (execute-plan-helper source(deron path)n-BbUh-b (source
                                                                                  as the s
                                                       oation i
                                                                hod)
                                                                        11
                                                                                             and aking be don achterent
                                                                            m) P
                                                                                 a source a
                                                                 e destina
                             (cond ((eql method ::
                                                   rch)
                                                         fs so
                                                                                 bra.defaul
                                                                                             ase which arays
                                                   ) (cb)
                                                                                                             Ma Ithe groupers
                                   ((eql method :
                                                          uper
                                                                 rce desi
   (defun process-two (paifedatabase
                                                   ose)
                                                                                                            gual
                                                                                                                   (rever
                                       (or (cbr-super source destination "plan")
        (cond ((null database) night source destination)))
                                                                               (t (or (cbr-top (cadr phose bisthe xate mate
                                                                                        (append path (cbr-get-path chosen-path
                 ((or (equal (Cadr) pair) (caar database))
                                                                                       plas)
 (defun convert-output-helper (path) (reverse (cadr pair)) (caar database))) or-top (car chosen-path) destination ne
                                                                                        (append path (cbr-get-path chosen-path
   (cond ((null path) mil) (append (list (append (list (caar database))
                                                                                       plan)
        (t (append (list (cons (first path) (list (list (first path)
                                (removerpairpatigan pair) (cdar database) do + top source destination (cbr-new-plan
                 (convert-output-helper (rest path)))
(cdr database)))
                                                                                        (cbr-evaluate source destination
                                                                                        (cbr-retrieve source destination plan))
         (defvar /database
                                               database)
                                                                                           plan)))
           · (((10th Then P)
                                                               av North-Avenue))
            ((10th Curran) (10th process two pair M(conadatabase))))
                (Curran 8th-1))
            ((10th McMillan) (10th Curran) (10th hemphill) (mcmillan 8th-1))
            ((10th hemphill) (10th mcmillan) (10th dalney) (hemphill 8th-1))
```