

LISP Transition Mechanisms

Oct 2007

Darrel Lewis, Dino Farinacci, David Meyer, Vince Fuller, Scott Brim, Noel Chiappa

Introduction

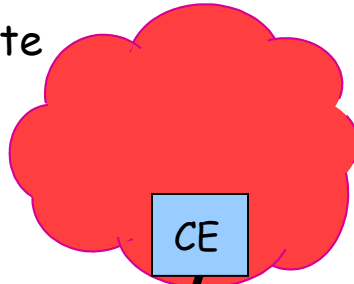
- General Thoughts
 - Share the goals of LISP
 - No magic bullet
- Core requirements
 - Incremental deployment
 - Minimize pain/cost
 - Independent of which mapping system chosen (?)

Three Transition Mechanisms

- #1 Routable EIDs
 - Not much time spent on this
 - Having everything in both mapping systems seems non optimal
- #2 Proxy Tunnel Routers (PTRs)
 - Can work nicely if we use a separate sub namespace for the new 'PI EIDs'
- #3 Source NAT
 - Has all the classic problems of NAT

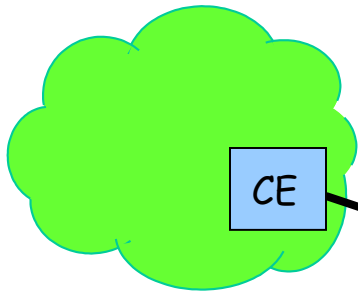
Reference Transition Topology

Non LISP Site



CE

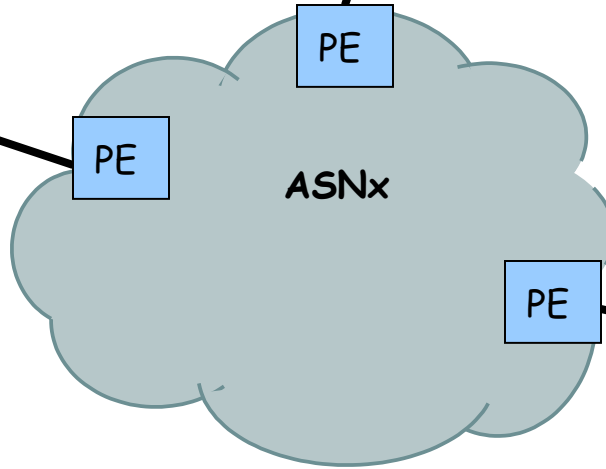
PE



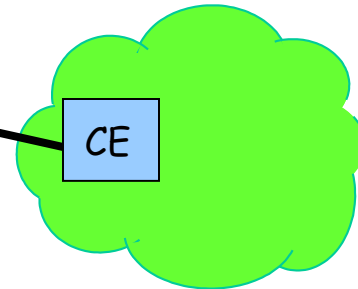
CE

PE

ASNx



PE



CE

LISP Site: LISP-R
(addressed from routable space)

LISP Site: LISP-NR
(addressed from non-routable space)

Routable EIDs

- EIDs published in both the existing BGP DFZ *and* the LISP mapping database
 - Essentially there are no sites that are 'LISP-NR'
- EIDs can only be withdrawn from a table after transition is 'completed'
- This mechanism may provide a good way to get started and gather data

Proxy Tunnel Routers (PTRs)

- PTRs Originate the new EID sub-namespace
 - Sub-namespace: A chunk of PA locator space set aside for EID transition
 - Some advantages if this space aggregated
 - Something like 240/4 for example ☺
 - <Insert your own address aggregate here>
- Packets from non LISP sites trying to reach LISP-NR sites are routed to these PTRs
 - The PTR has the mapping information of the destination ETR
 - The return path does NOT go back through the PTR - the default is asymmetric

Scaling PTRs

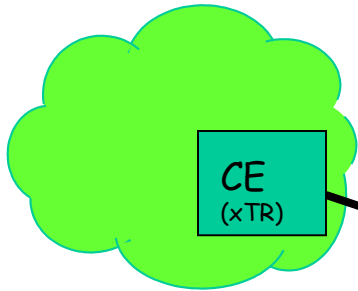
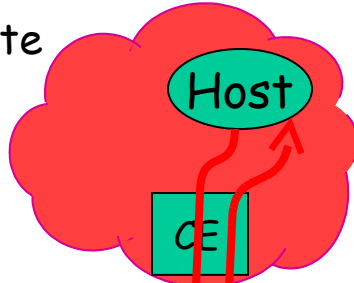
- PTRs sink traffic to them by announcing EID namespace
 - can announce the entire EID sub-namespace or more specific pieces of the sub-namespace
 - PTRs need to be robust and scale well
 - Puts onus on a SP to manage/pay for transition
- Performance considerations
 - Stretch
 - State
 - Asymmetry

Packet Flow with PTRs

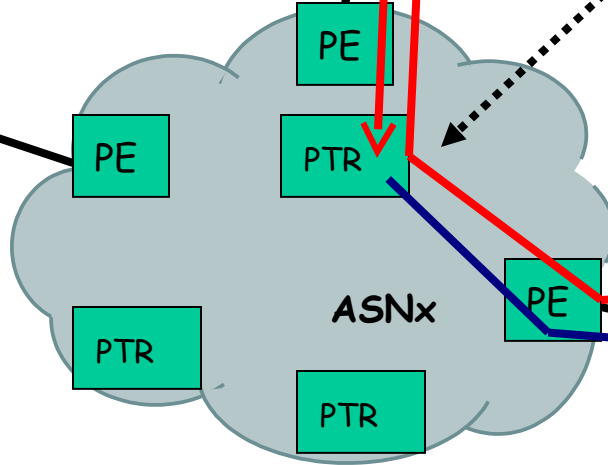
- A packet flow from non LISP site to LISP-NR site:
 - Host looks up EID for dest - gets 240.1.1.1
 - CE default routes to its PE (240/4 not in table)
 - PE has route to 240/4 next hop is the PTR
 - PTR has mapping information and LISP encaps
 - Return path is asymmetrical
- Packet Flow from non lisp site to lisp-r site
 - Since destination is routable PTR not used

Transition Topology: PTR

Non LISP Site

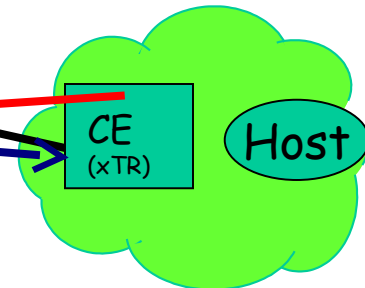


LISP Site: LISP-R
(addressed from routable space)



Proxy TR (PTR)

Not used in the case of LISP-NR site talking to another LISP-NR site.



LISP Site: LISP-NR
(addressed from non-routable space)

Further Thoughts on PTRs

- Need to figure out how forwarding features like uRPF work in this model
- How best to control route announcements
- Better understand security
-

Source NAT

- Source NAT hides the NR space from non-lisp sites
- Each xTR can have the ability to NAT before encap
- Should support both 1:1 NAT, and P-NAT
 - 1:1 NAT will require more than one /32 of PA space assigned to that site
 - Multi-homing should still work
 - Ingress will have to be statically mapped (just like NAT today)

Source NAT

- Source NAT hides the NR space from non-lisp sites
- Each xTR can have the ability to NAT before encap
- Should support both 1:1 NAT, and P-NAT
 - 1:1 NAT will require more than one /32 of PA space assigned to that site
 - Multi-homing should still work
 - Ingress will have to be statically mapped (just like NAT today)

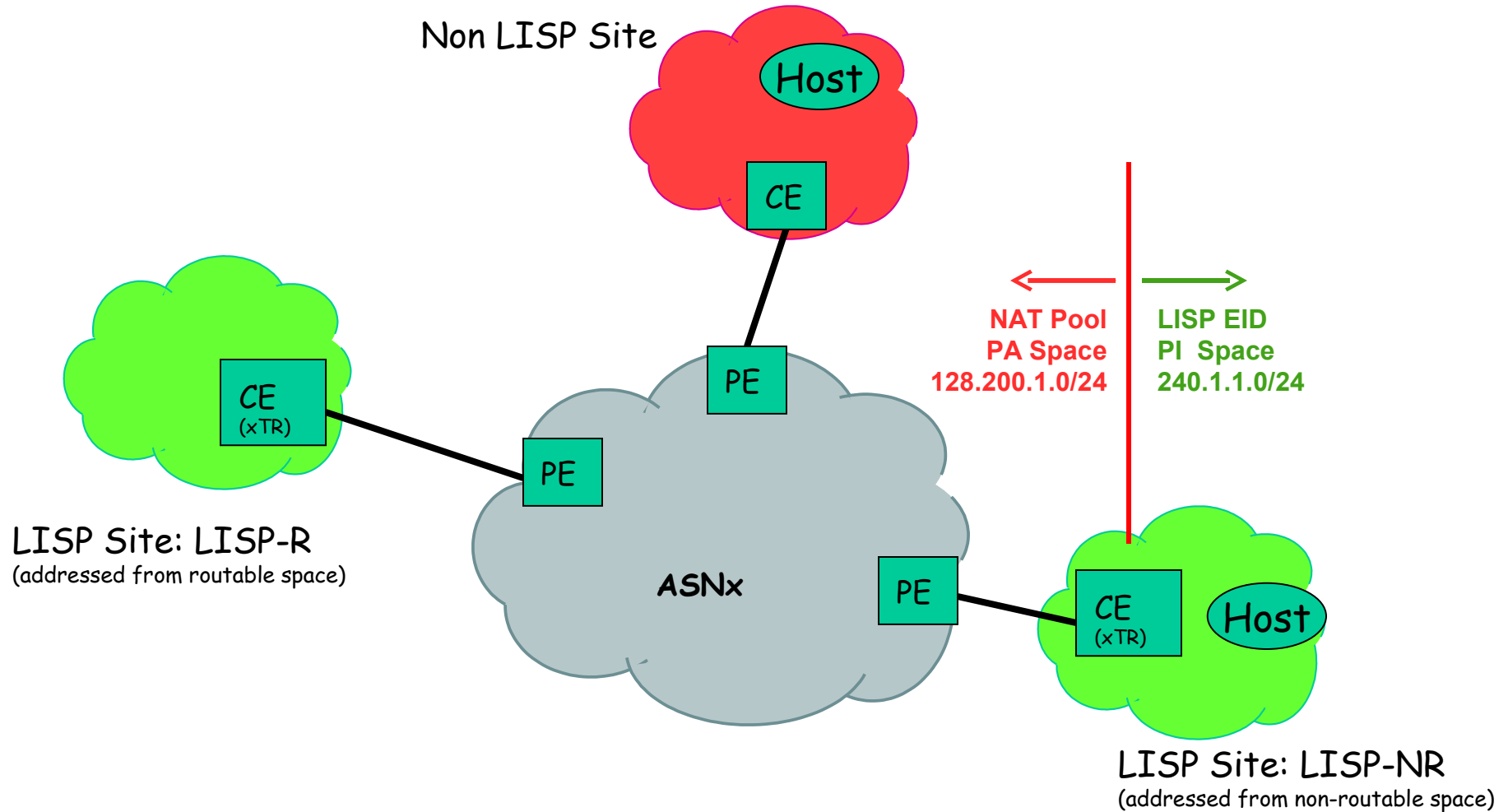
Source NAT Steps

- Site has PA 128.200.1.0/24 assigned to it
 - 128.200.1.1 will be used as the R-Loc site
 - The rest of the /24 will be used for 1:1 NATs
- The PI EID space for the site is 240.1.1.0/24
 - Just an example, can be any non-routed address

Source NAT Steps (cont)

- The ITR then performs a NAT function on the outgoing packet of say 240.1.1.2 to 128.200.1.2 when talking to a non-lisp site...
 - Ingress connections must use 128.200.1.2 as the public EID for this host (classic NAT ugliness)
- When 240.1.1.2 wants to talk to a LISP site, the EID is maintained, and the source rloc of 128.200.1.1 is used
 - Ingress connections to 128.200.1.1 are always LISP packets

Transition Topology: Source NAT



Further Thoughts on NATs

- One big open question is how you deal with inbound connections with two possible EIDs
- Need to gather a lot more data on how this would work in the real world
- Better understand security
-

Some Open Questions

- Did we miss anything?
- Which is uglier - PTRs or Source-NAT

References (incomplete)

- Background Information
 - Route Scalability work (vaf, jason, et al)
 - RAWs Report
- RRG List
- LISP
- CONS

(defun changer-one (failures database) ; If we have an exact match between source/destination ; ; we've reached the conclusion of CBR - return the path
(cond ((null (cdr failures)) (cdr failures) second-part))
(t (changer-one (cdr failures) second-part)))
(defun convert-output-helper-b (path) ; (cond ((cbr-exact-match (list source destination) plan
 (append path (cbr-get-path (list source destination)
 (cbr-exact-match (list source destination) prev
 (append path (cbr-get-path (list source destination
 ; If we have no chosen-path, we can't recurse - we return
 ((null chosen-path) nil)
(defun process-one (pair (cdr database)) ; If either the two elements of the chosen-path match, ; ; the second element of chosen-path as
(cond ((null chosen-path) (remove-pair (cadr pair) (cdar database))) ; ; a sub-element of the chosen-path matches the destination
 ; ; we have an invalid recursive call - we return nil.
(t (changer-one (cdr pair) (cdar database)) (cadr chosen-path) (cadr chosen-path))
 ((equal (cadr chosen-path) destination) nil)
(execute-plan-helper (source path (cbr-super source destination plan*)) (cadr chosen-path) (cadr chosen-path))
 ((equal (cadr chosen-path) destination) nil)
(check-failure (first path) *failure-points (cdr database))
(append (list (append (list (first path) (list 'UNKNOWN)) (remove-pair (cadr pair) (cdar database))) (cadr chosen-path)
 (append (list (append (list (first path) (list 'FAILURE)))
 (execute-plan-helper source (rest path) (cbr-super source destination plan*)) (cadr chosen-path) (cadr chosen-path))
 ((equal (cadr chosen-path) destination) nil)
(t (append (list (car database)) (process-one pair (cdr database))
(execute-plan-helper source (rest path) (cbr-super source destination plan*)) (cadr chosen-path) (cadr chosen-path))
 ((equal (cadr chosen-path) destination) nil)
(t (append (list (first path) (list 'FAILURE))
(execute-plan-helper source (rest path) (cbr-super source destination plan*)) (cadr chosen-path) (cadr chosen-path))
 ((equal (cadr chosen-path) destination) nil)
(defun process-two (pair (cdr database)) ; If either the two elements of the chosen-path match, ; ; the second element of chosen-path as
(cond ((null database) nil) ; ; a sub-element of the chosen-path matches the destination
 ; ; we have an invalid recursive call - we return nil.
(t (or (cbr-top (cadr chosen-path) destination) (append path (cbr-get-path chosen-path
 plan)
 (cbr-top (car chosen-path) destination) (append path (cbr-get-path chosen-path
 plan)
 (cbr-top source destination (cbr-new-plan
 (cbr-evaluate source destination
 (cbr-retrieve source destination plan)
 plan)))
(defvar *database*
'(((10th Tech-Parkway) (10th Curran) (Tech-Parkway North-Avenue))
 ((10th Curran) (10th hemphill) (hemphill 8th-1))
 (Curran 8th-1))
 ((10th McMillan) (10th Curran) (10th hemphill) (mcmillan 8th-1))
 ((10th hemphill) (10th mcmillan) (10th dalney) (hemphill 8th-1))

LISP

RULES