Eliminating Duplicate Checks in ICE: Alternate Proposal

Philip Matthews
Eric Cooper

Alternate Proposal

- Combines best ideas from both Jonathan’s proposal and Philip/Eric’s proposal.
- Has a unified state machine (rather than separate Rx and Tx state machines).
- Takes advantage of “associated transport address” information signaled in SDP.
- Eliminates all duplicate checks.
- Is significantly simpler than the two earlier proposals.

Alternate Proposal

Each endpoint maintains two lists:

- List of Transport Address Pairs, each with two associated state variables:
  - IN: pair works in inbound direction
  - OUT: pair works in outbound direction
- List of checks to perform, each of the form:
  - From native **base** transport address (where "base" = "not server-reflexive")
  - To remote transport address
  - One check for each possible combination

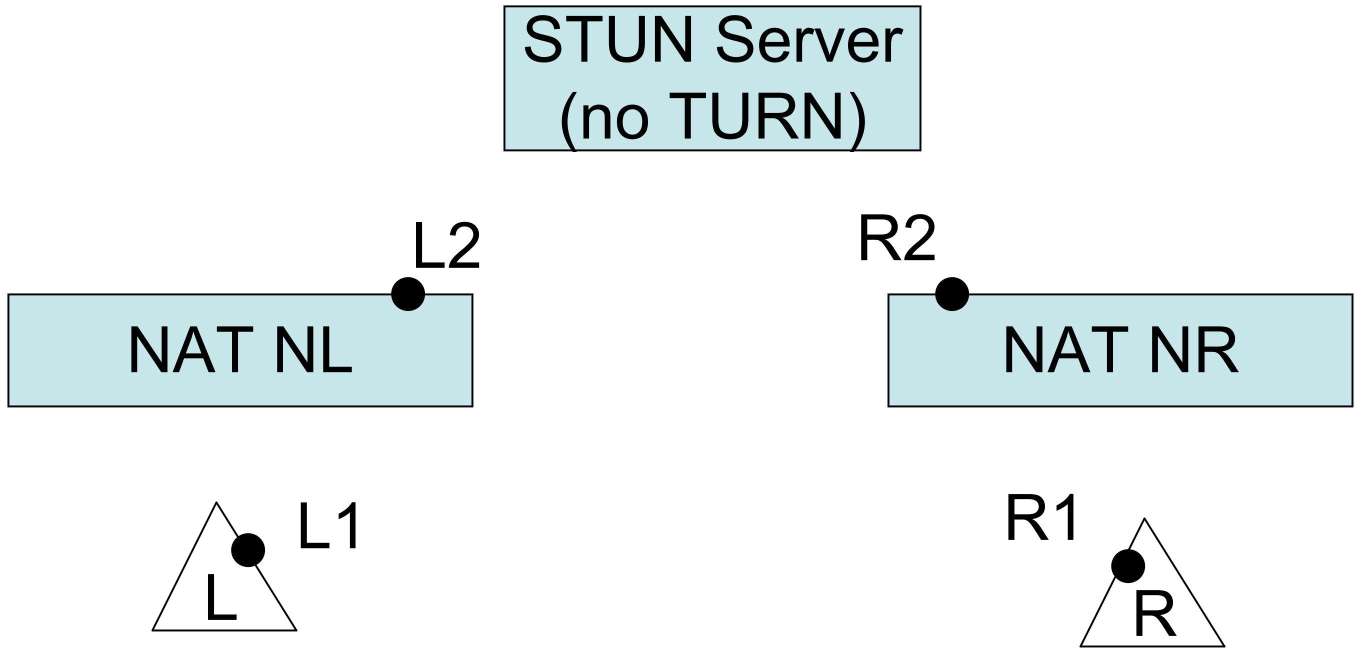
Alternate Proposal

- When a Binding Request arrives, receiving endpoint knows that the transport address pair given in the username works inbound.
- Also, receiving endpoint knows that any associated transport address pair also works.
  - For example, on L, receiving L1:1:R1:1 means that both L1:1:R1:1 **and** L1:1:R2:1 work inbound, if R2:1 is a server-reflexive tid derived from R1:1.

Alternate Proposal

- Similarly, when a Binding Response arrives, the endpoint knows that, not only does that specific transport address pair work outbound, but so does any associated transport address pairs
  - For example, on R, receiving a response for L1:1:R1:1 means that both L1:1:R1:1 **and** L1:1:R2:1 work outbound, if R2:1 is a server-reflexive tid derived from R1:1.

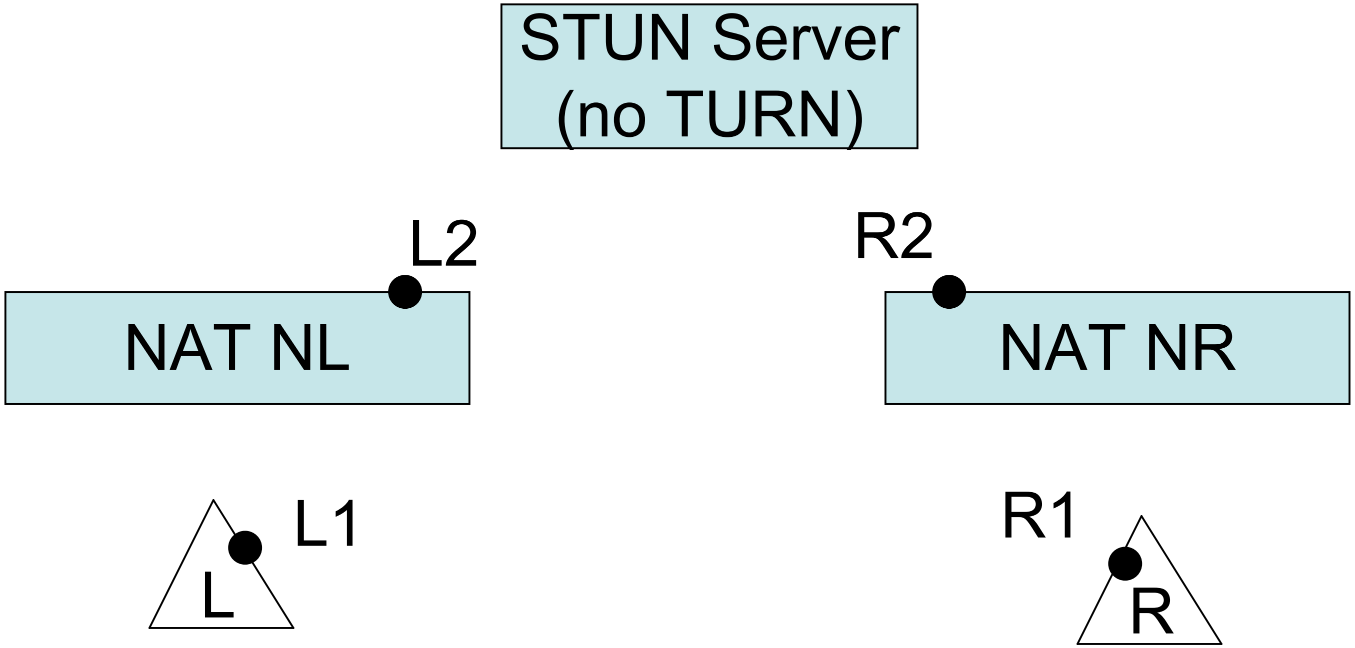
Example



Both NATs are BEHAVE compliant. For simplicity, we assume they have the endpoint-independent filtering property.

L is the Offerer, R is the Answerer. This means that R starts its checks slightly before L.

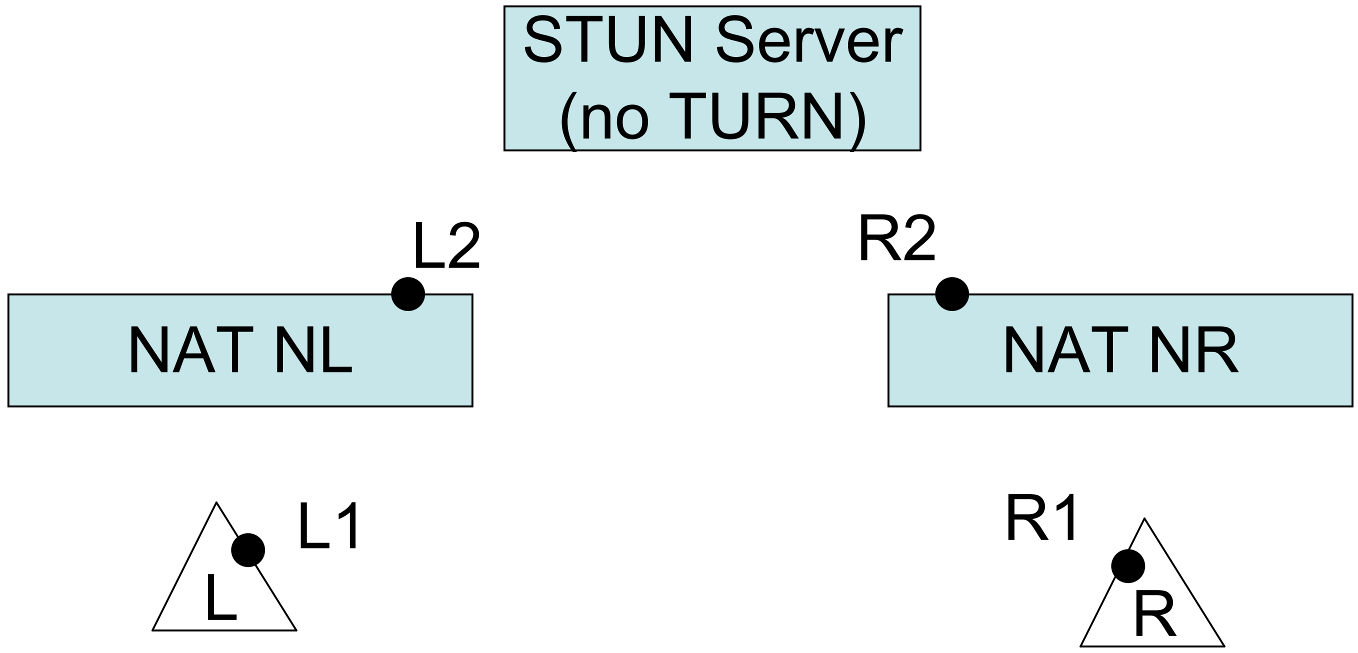
Example



Candidates are:

L1, q = 1	R1, q = 1
L2, q = .7	R1, q = .7

Example



In this example, the m/c line is empty (= a-inactive). Thus the transport address check ordering is:

L1:1:R1:1	1st
L1:1:R2:1	2nd
L2:1:R1:1	3rd
L2:1:R2:1	4th

Example (Step 0)

Check List -- List of checks to perform (different for each end)
"In" (resp. "Out") - Can receive (resp. transmit) on that pair.

On L				On R			
Check List	**Pair**	**In**	**Out**	**Pair**	**In**	**Out**	**Check List**
L1:1→R1:1	L1:1:R1:1			L1:1:R1:1			L1:1←R1:1
L1:1→R2:1	L1:1:R2:1			L1:1:R2:1			L2:1←R1:1
	L2:1:R1:1			L2:1:R1:1			
	L2:1:R2:1			L2:1:R2:1			

Example (Step 1)

Check List -- List of checks to perform (different for each end)
"In" (resp. "Out") - Can receive (resp. transmit) on that pair.

On L				On R			
Check List	**Pair**	**In**	**Out**	**Pair**	**In**	**Out**	**Check List**
L1:1→R1:1	L1:1:R1:1			L1:1:R1:1			L1:1←R1:1
L1:1→R2:1	L1:1:R2:1			L1:1:R2:1			L2:1←R1:1
	L2:1:R1:1			L2:1:R1:1			
	L2:1:R2:1			L2:1:R2:1			

L1:1→R1:1 (=R1:1:L1:1)

L1:1←R1:1 (=L1:1:R1:1)

Step 1: R tries check L1:1←R1:1, and L tries L1:1→R1:1; both fail.

Example (Step 2)

Check List -- List of checks to perform (different for each end)
“In” (resp. “Out”) - Can receive (resp. transmit) on that pair.

On L				On R			
Check List	**Pair**	**In**	**Out**	**Pair**	**In**	**Out**	**Check List**
L1:1→R1:1	L1:1:R1:1			L1:1:R1:1			L1:1←R1:1
L1:1→R2:1	L1:1:R2:1			L1:1:R2:1			L2:1←R1:1
	L2:1:R1:1	√		L2:1:R1:1			
	L2:1:R2:1	√		L2:1:R2:1			

L1:1←R1:1 (=L1:1:R1:1)

L1:1→R1:1 (=R1:1:L1:1)

L2:1←R1:1 (=L2:1:R1:1)

Step 2: R tries L2:1←R1:1, which reaches L. Thus L knows L2:1:R1:1 works inbound. In addition, L2:1:R2:1 also works inbound, since R2:1 is server-reflexive version of R1:1.

Example (Step 3)

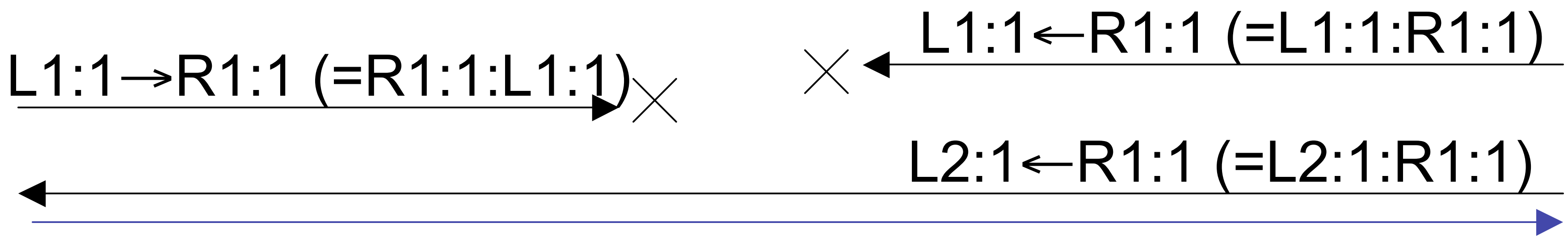
Check List -- List of checks to perform (different for each end)
"In" (resp. "Out") - Can receive (resp. transmit) on that pair.

On L

Check List	**Pair**	**In**	**Out**
L1:1→R1:1	L1:1:R1:1		
L1:1→R2:1	L1:1:R2:1		
	L2:1:R1:1	√	
	L2:1:R2:1	√	

On R

Pair	**In**	**Out**	**Check List**
L1:1:R1:1			L1:1←R1:1
L1:1:R2:1			L2:1←R1:1
L2:1:R1:1		√	
L2:1:R2:1		√	



Step 3: L sends the response back to R. Now R knows that L2:1:R1:1 and L2:1:R2:1 work outbound.

Example (Step 4)

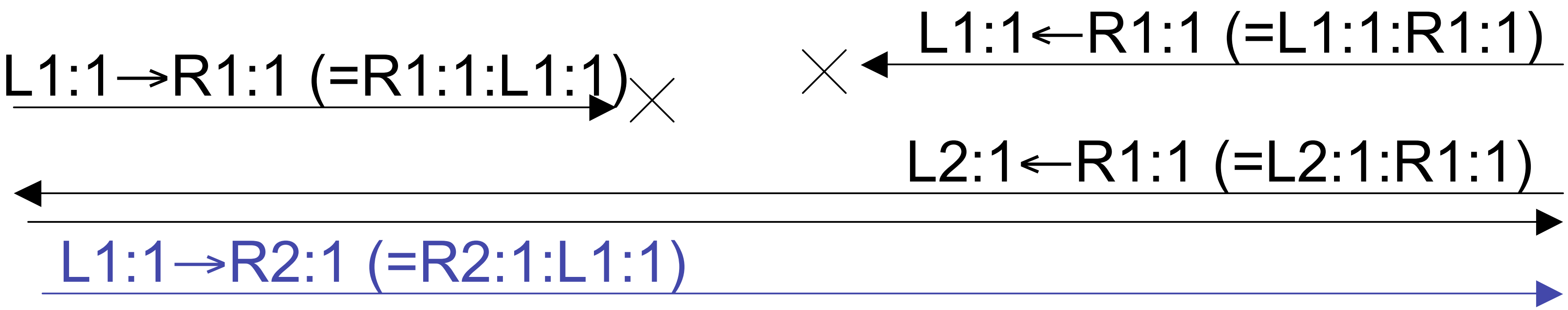
Check List -- List of checks to perform (different for each end)
"In" (resp. "Out") - Can receive (resp. transmit) on that pair.

On L

Check List	**Pair**	**In**	**Out**
L1:1→R1:1	L1:1:R1:1		
L1:1→R2:1	L1:1:R2:1		
	L2:1:R1:1	√	
	L2:1:R2:1	√	

On R

Pair	**In**	**Out**	**Check List**
L1:1:R1:1			L1:1←R1:1
L1:1:R2:1	√		L2:1←R1:1
L2:1:R1:1		√	
L2:1:R2:1	√	√	



Step 4: L tries L1:1→R2:1, which reach R. Thus R knows that both L1:1:R2:1 and L2:1:R2:1 work inbound.

Example (Step 5)

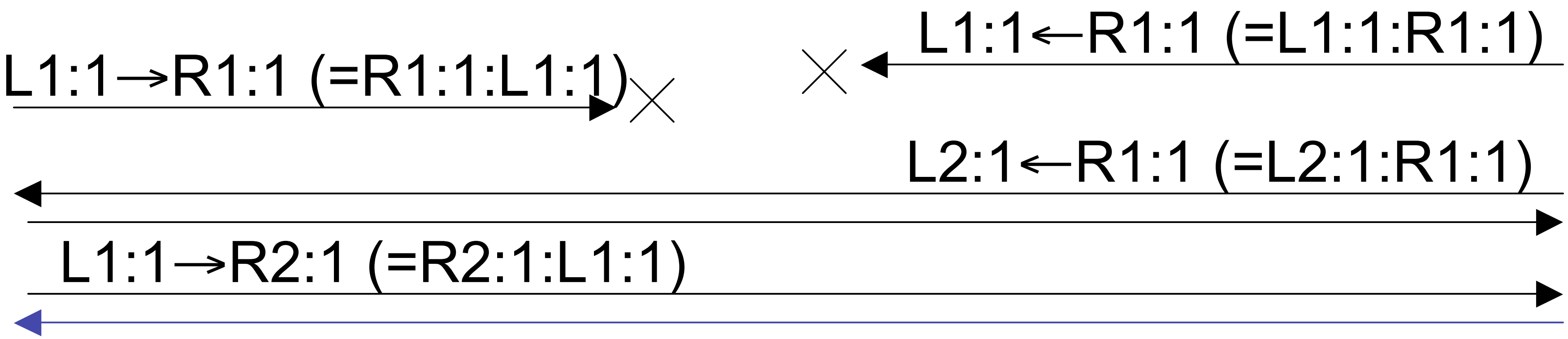
Check List -- List of checks to perform (different for each end)
"In" (resp. "Out") - Can receive (resp. transmit) on that pair.

On L

Check List	Pair	In	Out
L1:1→R1:1	L1:1:R1:1		
L1:1→R2:1	L1:1:R2:1		√
	L2:1:R1:1	√	
	L2:1:R2:1	√	√

On R

Pair	In	Out	Check List
L1:1:R1:1			L1:1←R1:1
L1:1:R2:1	√		L2:1←R1:1
L2:1:R1:1		√	
L2:1:R2:1	√	√	

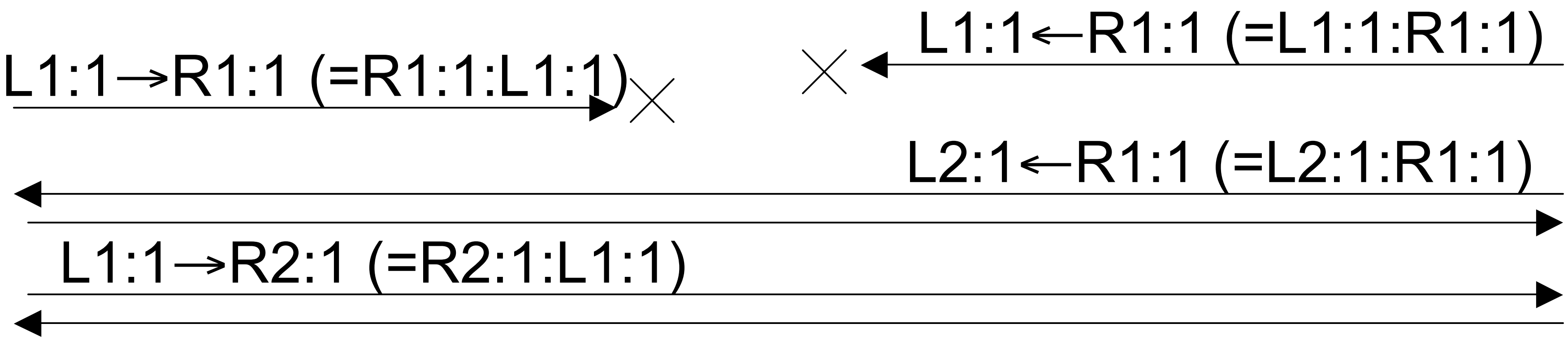


Step 5: R replies, and thus L knows that both L1:1:R2:1 and L2:1:R2:1 work outbound.

Example (Step 6)

Check List -- List of checks to perform (different for each end)
“In” (resp. “Out”) - Can receive (resp. transmit) on that pair.

On L				On R			
Check List	**Pair**	**In**	**Out**	**Pair**	**In**	**Out**	**Check List**
L1:1→R1:1	L1:1:R1:1			L1:1:R1:1			L1:1←R1:1
L1:1→R2:1	L1:1:R2:1		√	L1:1:R2:1	√		L2:1←R1:1
	L2:1:R1:1	√		L2:1:R1:1		√	
	L2:1:R2:1	√	√	L2:1:R2:1	√	√	



Step 6: At this point, both L and R know that pair L2:1:R2:1 works in both directions, and can be promoted.