# Name overlay (NOL) service for scalable Internet routing

Yangyang Wang, Wei Zhang, Jun Bi

## Scaling Issues from the Internet edge

- Routing scalability:
  - routing table size and updates are growing rapidly
  - Multi-homing, traffic engineering, non-aggregate provider-independent (PI) addressing, mobility.
- Proposed solutions
  - 1) Separate edges from transit core
  - 2) ID/Locator split, and provider-allocated (PA) addressing on host.
  - Proposals: LISP, APT, Ivip, TRRP, VA, Six/one, HIP, Shim6, name-oriented stack, ILNP,…

## Less incentives for deployment

- One deploys, others gain.
- Can not be deployed unilaterally
- There is no flag day for clean-slate designs
- It is not clear how much incentives for ISPs to upgrade their routers.

# From transit core to edge

- Most of solutions are mainly to increase benefits of the transit networks.
  - How to do routing and addressing to reduce the routing table entries of the upstream networks
- We should also pay attention to the benefits of the Internet edge
  - End applications and users are the engine of Internet growth from historical view
  - Changes at the edge are easier than transit networks

How to enable the edge to obtain more benefits from multi-homing, TE, Mobility in a rapid scaling Internet? Maybe such solutions also facilitate the changes for scalability

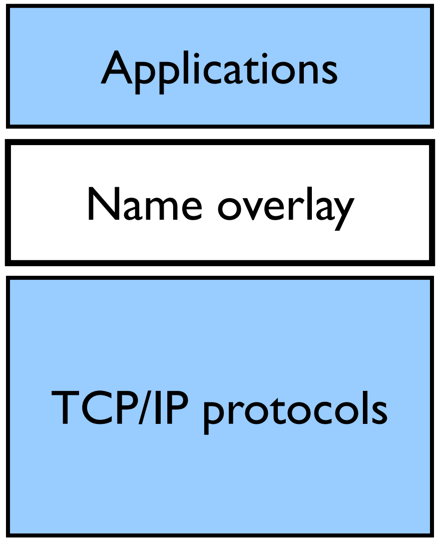
## Lessons from NAT

- Pros
  - Can be deployed unilaterally and benefits deployed networks
  - Easy site renumbering
  - Can support PA address multi-homing
  - No change to protocol stack
  - Most hosts behind NAT don't complain being disable to receive incoming connections
- Cons
  - Break the e2e sessions (also firewalls, proxies)
  - Prevent initiating session from outside
  - NAT is already used to IP scaling

## Short-term objective

- Benefits both ISPs and end users
  - From multi-homing, TE, mobility
- Control the routing tables growth
- Do not change the operation of today's Internet
- Can be integrated with legacy systems
- Unilaterally deployable

# NOL Architecture



- Overlay on host protocol stack
- Host name configuration, register and authentication
- Initiate and manage transport connection channels by name

## Comparison with host-based solutions

- Existing host based solutions need to change protocol stack and/or socket
  - change socket implementations(OS kernel level reprogramming) more or less
  - HIP, SCTP, Shim6, name-oriented socket
- NOL don't change protocol and socket
  - Overlay on existing stack and socket
  - NOL applications can interact with legacy applications

# What benefits NOL can provide ?

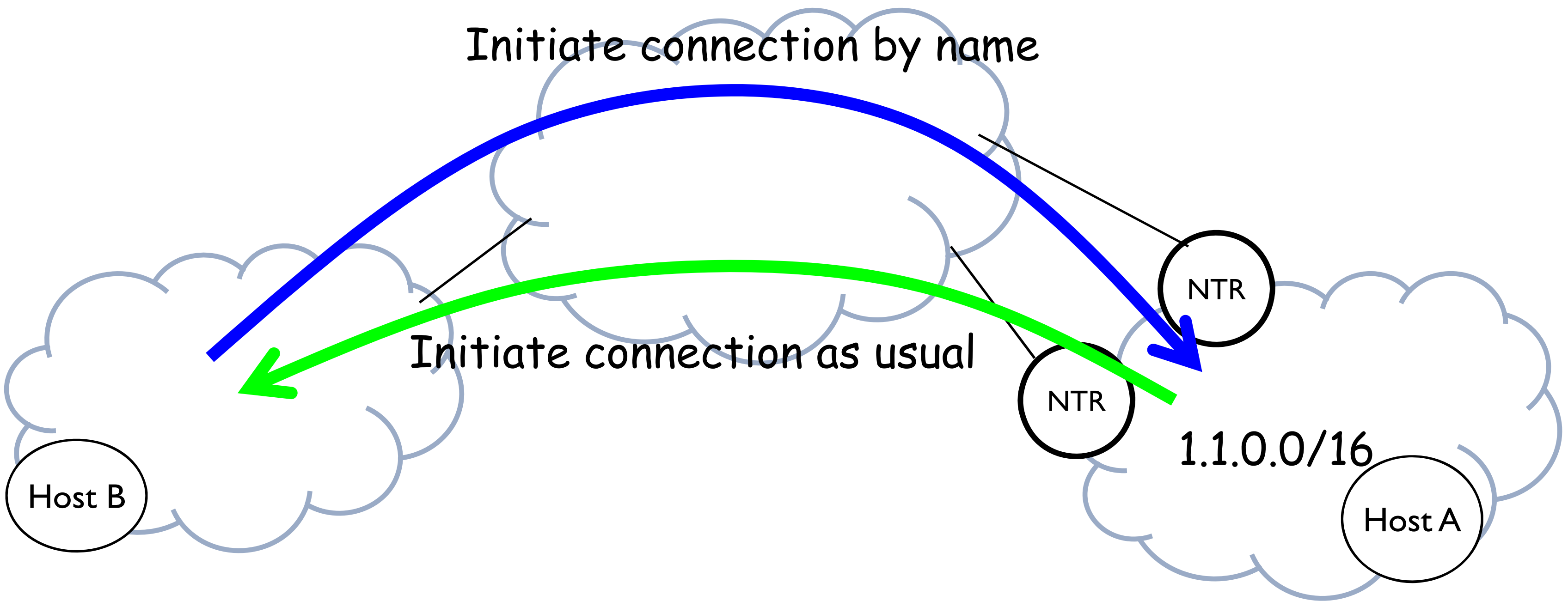
- benefits end users from multi-homing, TE and mobility
  - Manage transport connections by names
  - Provide user controlled multi-path data transport
- Reduce routing table size
  - control PI address by NTR *(to be introduced in the following)*
- Evolutionary deployment
  - Reside as an overlay on TCP/IP
  - Don’t change TCP/IP stack and DNS infrastructure
  - NTR can be deployed unilaterally
- Can coexist with core-edge separation solutions (e.g., APT, LISP, Ivip, etc.)

## NTR: Name Transfer Relay

- Cooperate with NOL applications
  - create address mapping based on name
  - To help host behind NTR to receive incoming connection from outside
- NAT based extension
- Deployed at access point of edge networks
- Can be used to control the edges PI addresses announcement

# Control PI addresses using NTR

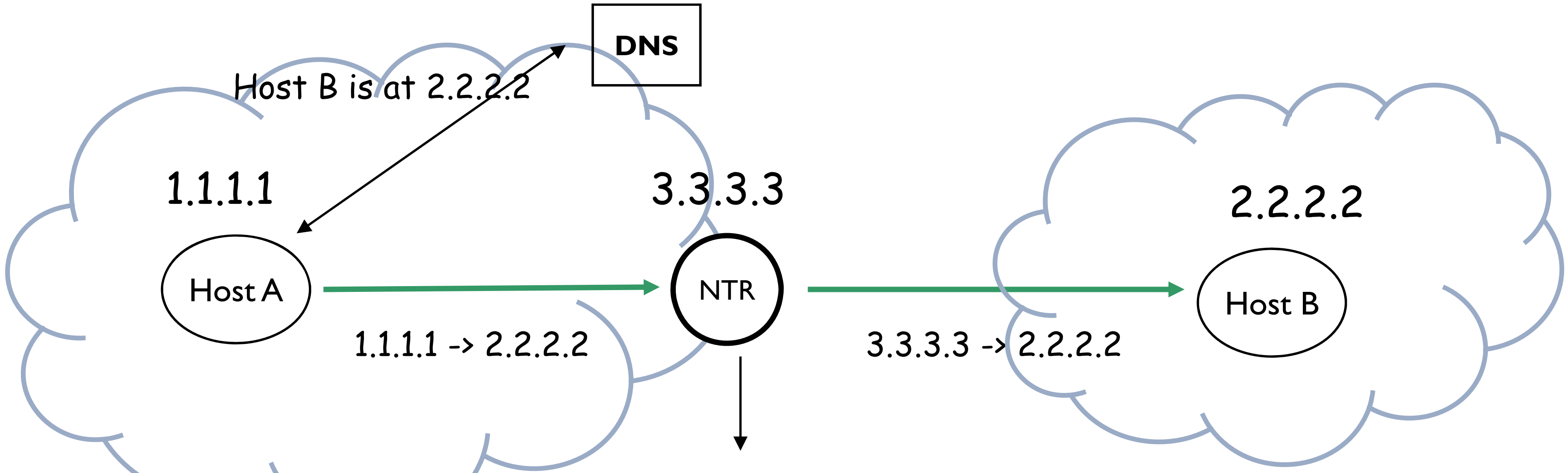
NTR can selectively prevent edge PI addresses from entering DFZ
(function similar to NATs )



# Initiate access to non-NTR hosts

Non-NTR hosts are the hosts within the networks that doesn't deploy NTR Gateway (i.e., today's Internet)

In this scenario, NTR works like NAT/NAPT



Address and/or Port translation

2.2.2.2 to 3.3.3.3

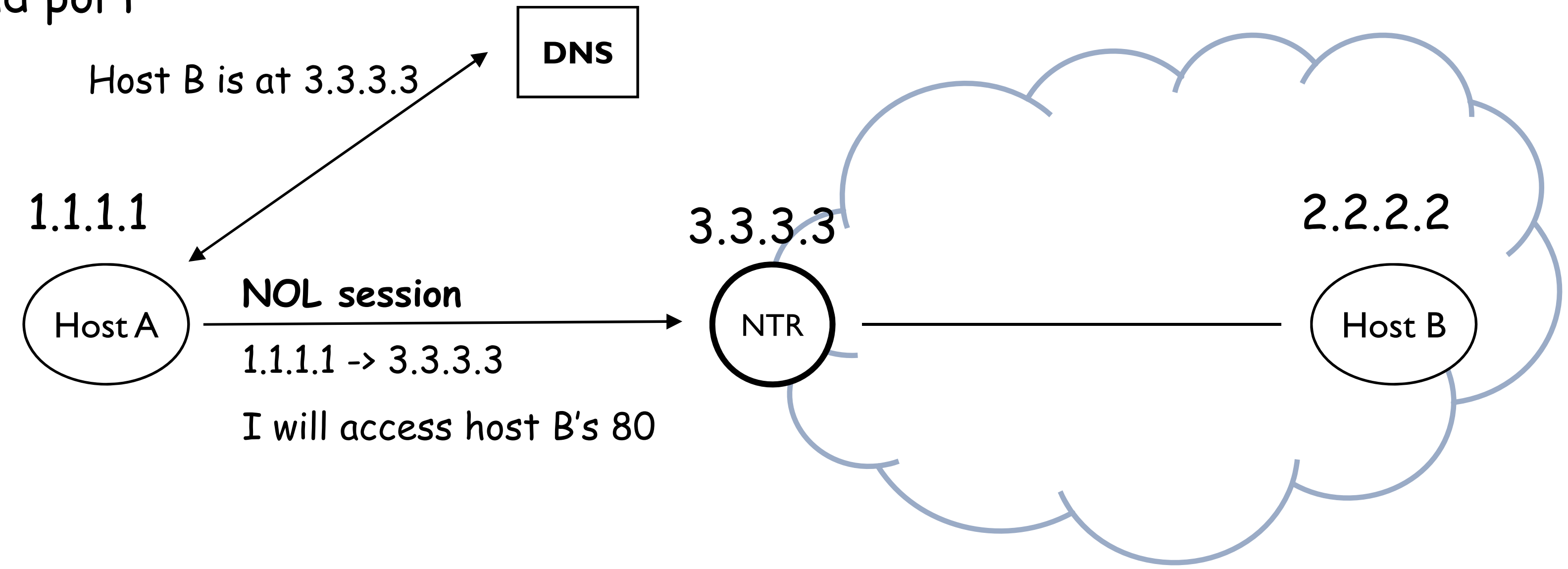
# Initiate access to NTR hosts (by name)

First, Host B obtained a name from NTR (e.g., B@network.com) or a FQDN (e.g., B.com) which is also known by NTR

Name of NTR hosts should be limited in a domain hierarchy, fox example, email address "host@domain" or "host.domain", We just query DNS for the "domain" that will not increase the load of DNS.

Second, host A query DNS for host B, and the returned IP is NTR 3.3.3.3, (configured in DNS)

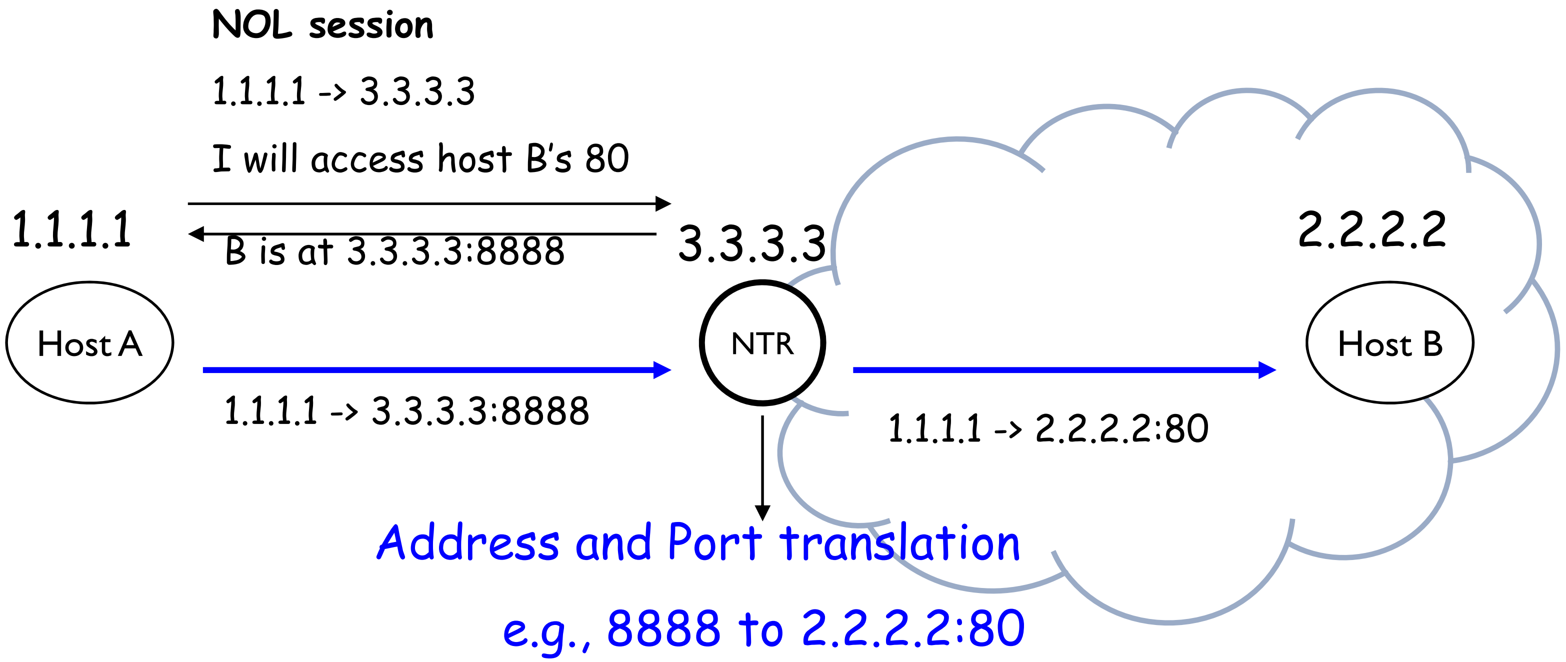
And then, host A's NOL will initiate NOL session to NTR, with B's name and port



Then, NTR look up B's name, and knows that B's IP is 2.2.2.2

We assume NTR has only one IP address, (here, 3.3.3.3), it will create a port to 2.2.2.2 mapping entry (e.g., port 8080 -> 2.2.2.2:80), and return the port (e.g., 8080) to A, NOL session is successful. (If NTR has many addresses, create address-to-address translating entry, or other type of translation)

Finally, host A initiate packet to 3.3.3.3:8888, and NTR translate it to 2.2.2.2:80. Initiating access from A to B succeed :)

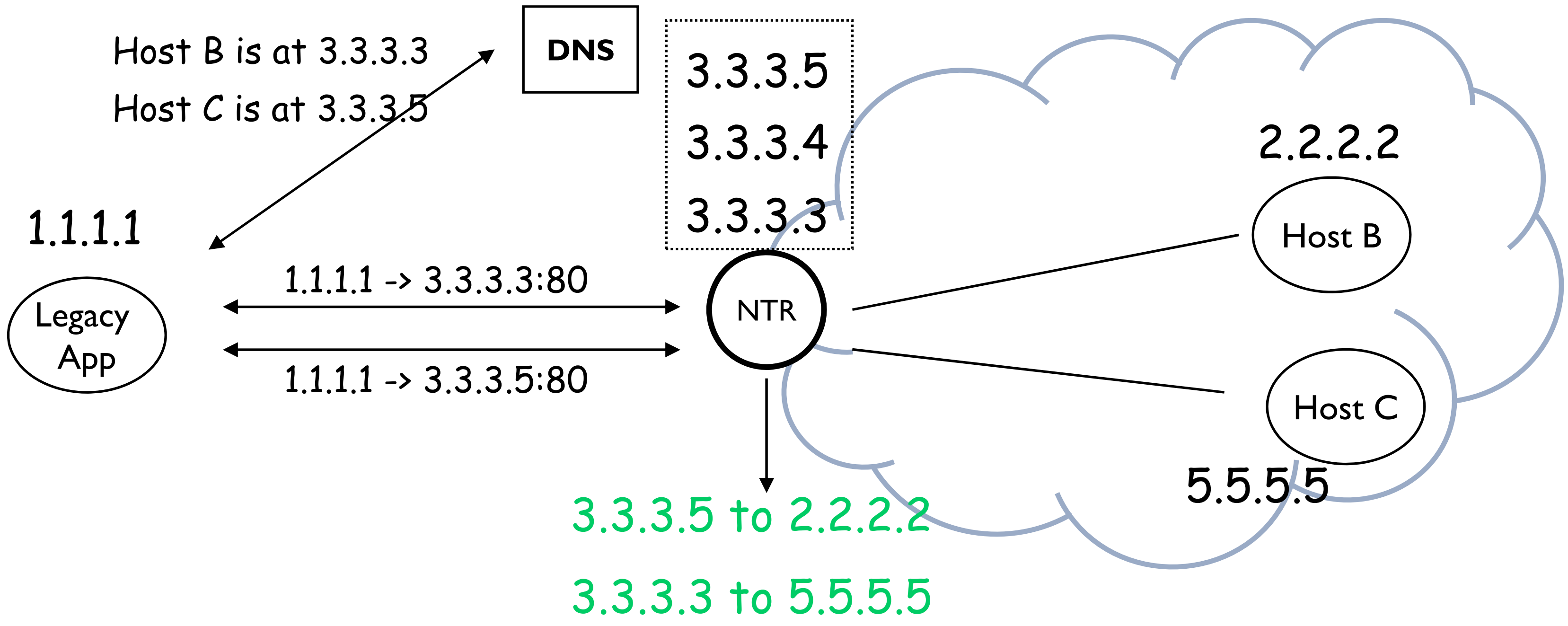


Address multiplexing by name and Port translation

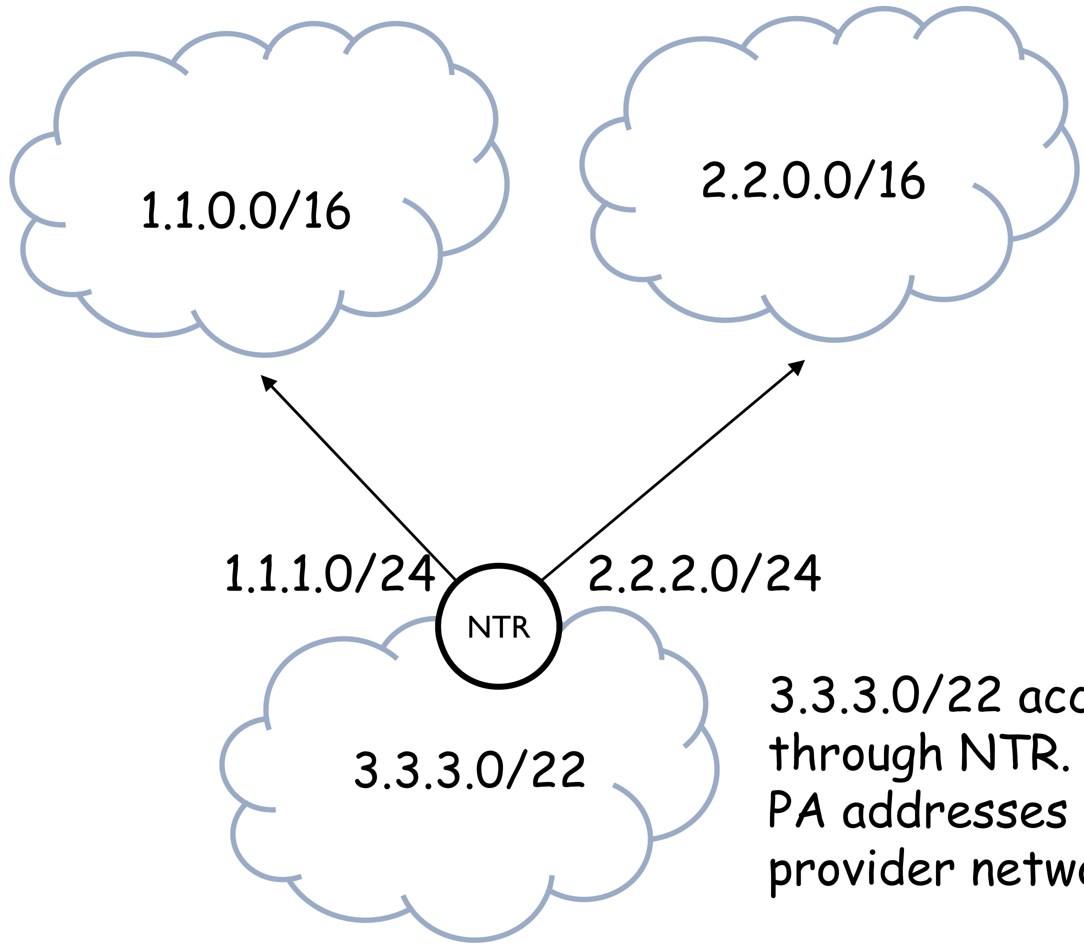
# How does legacy applications work (1)

NTR can hold multiple IP addresses (e.g., PA addresses from multi-homing) in its address pool.

delegate the IP addresses in NTR's address pool to the servers behind NTR.



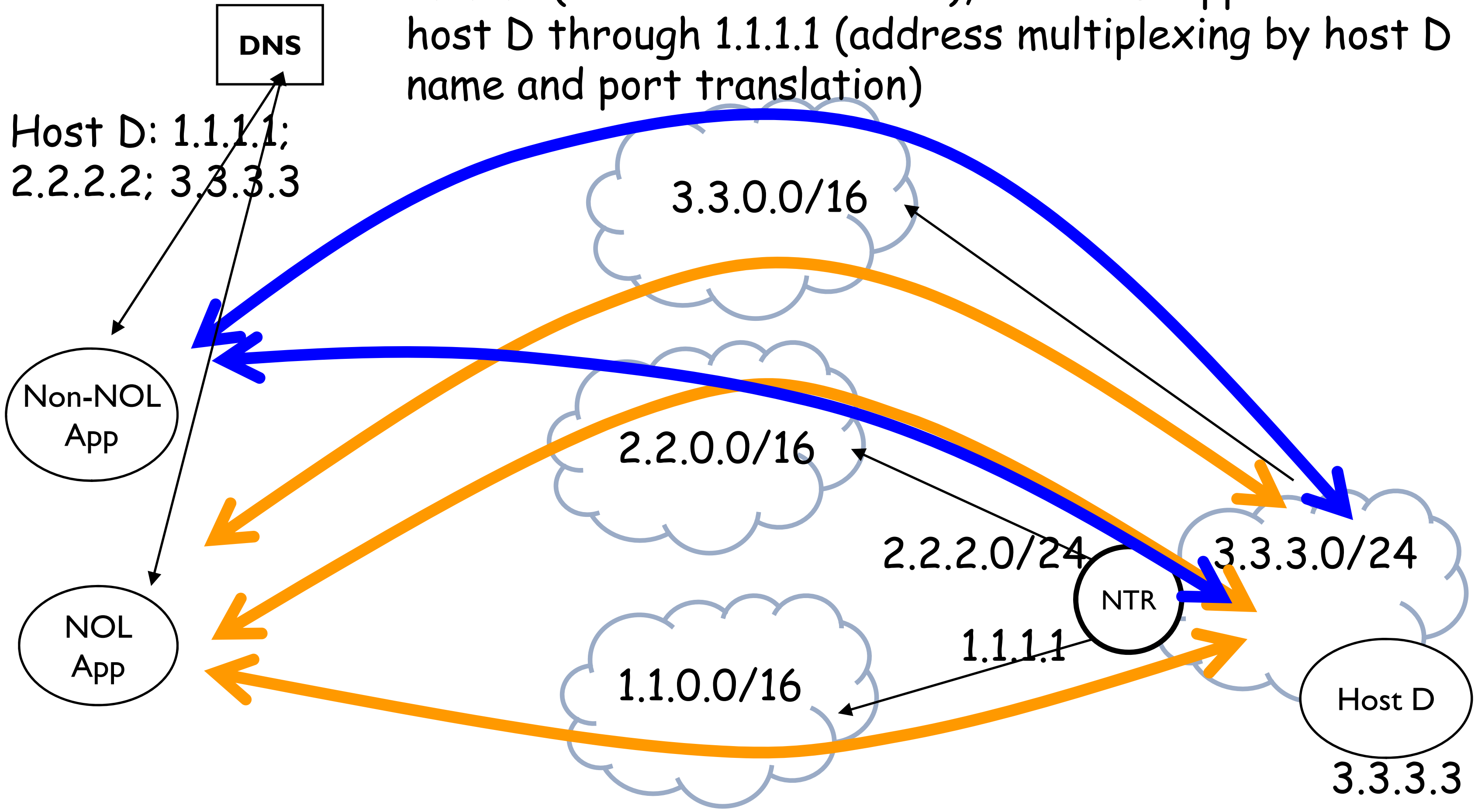
# Multi-homing



3.3.3.0/22 access Internet through NTR. And NTR get PA addresses from two provider networks

# Multi-homing

Non-NOL App can access host D through 3.3.3.3 and 2.2.2.2 (address translation), But NOL App can access host D through 1.1.1.1 (address multiplexing by host D name and port translation)

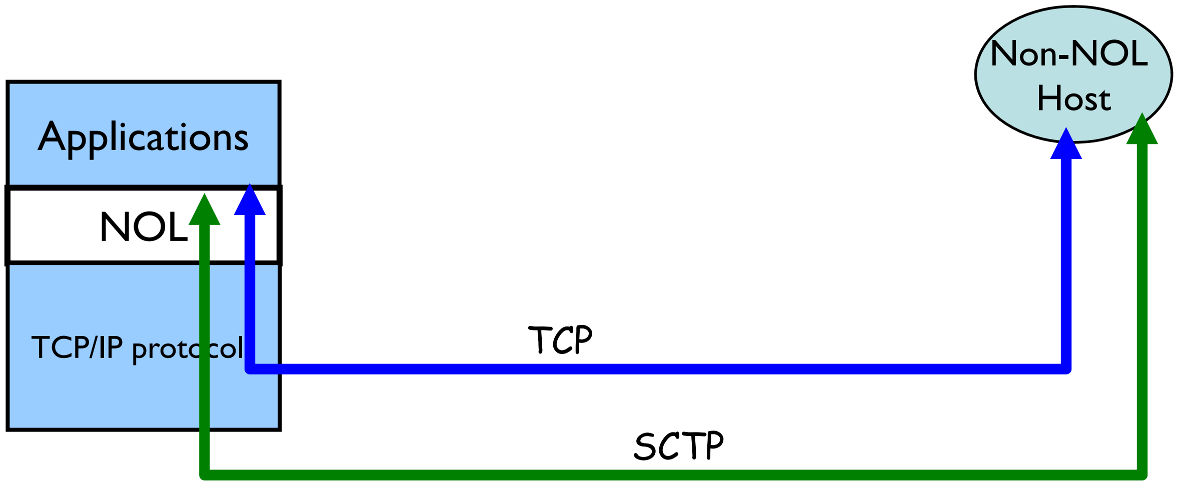


## Some considerations of the address mapping and translating on NTR

- One-to-one stateless address mapping
  - between PI addresses and multi-homing PA addresses
  - can be applied to IPv6 (e.g., NAT66)
- one-to-many address mapping
  - Need IP address multiplexing in IPv4
  - some possible ways
    - multiplexing on port number (e.g., NAPT)
    - based on the name or ID in NOL layer
    - multiplexing on obsolete IP option, such as Stream ID
    - IP-in-IP tunneling (e.g., ITR/ETRs of LISP)

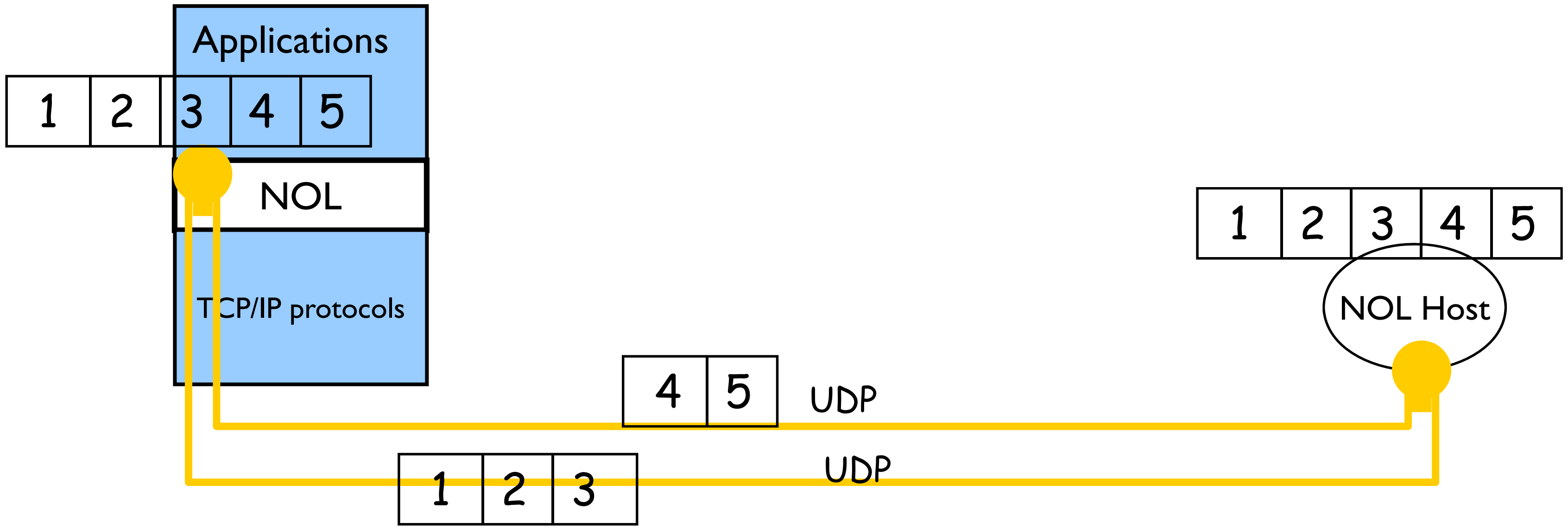
# NOL is not ID/locator split

- NOL create and manage transport connection channels by names.
- IP addresses or ports change, new connection channels are created
- Applications use these channels to transport data in their own ways



## Bundling the channels

- Applications can delegate their channels to NOL, which manage these channels like an e2e pipe (here, NOL seem to be similar to SCTP)

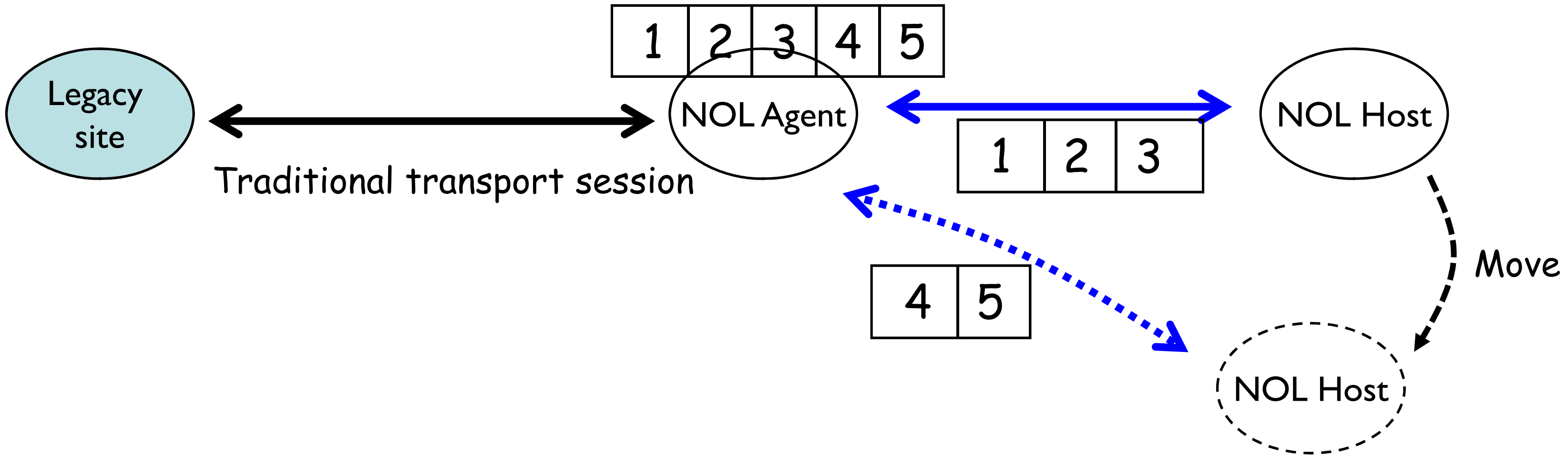


# API example

- Channels = Connect (name, port)
- Possible returned result: Channels[1], Channels[2], Channels[3]… multiple channels with multiple IP addresses
- Remove (Channels[3])
- Pipe = Bundle (channels[1], channels[2])

# Mobility

- Only NOL applications support mobility
- To keep application continuity using channels bundle between two NOL applications to shield the horizontal handoff
- Use NOL proxy to keep data transport continuity between legacy application and NOL application.



## Summary

- Pros
  - NOL host Don't need to change TCP/IP stack and DNS
  - NOL applications can communicate with legacies
  - User controlled multi-path transport
  - NTR can be deployed unilaterally
  - Reduce routing table,
  - support multi-homing, mobility
- Cons
  - Increase DNS entries
  - address translating cost on NTR
  - legacy applications have trouble with initiating a connection to the hosts behind a NTR.

## Deployment roadmap

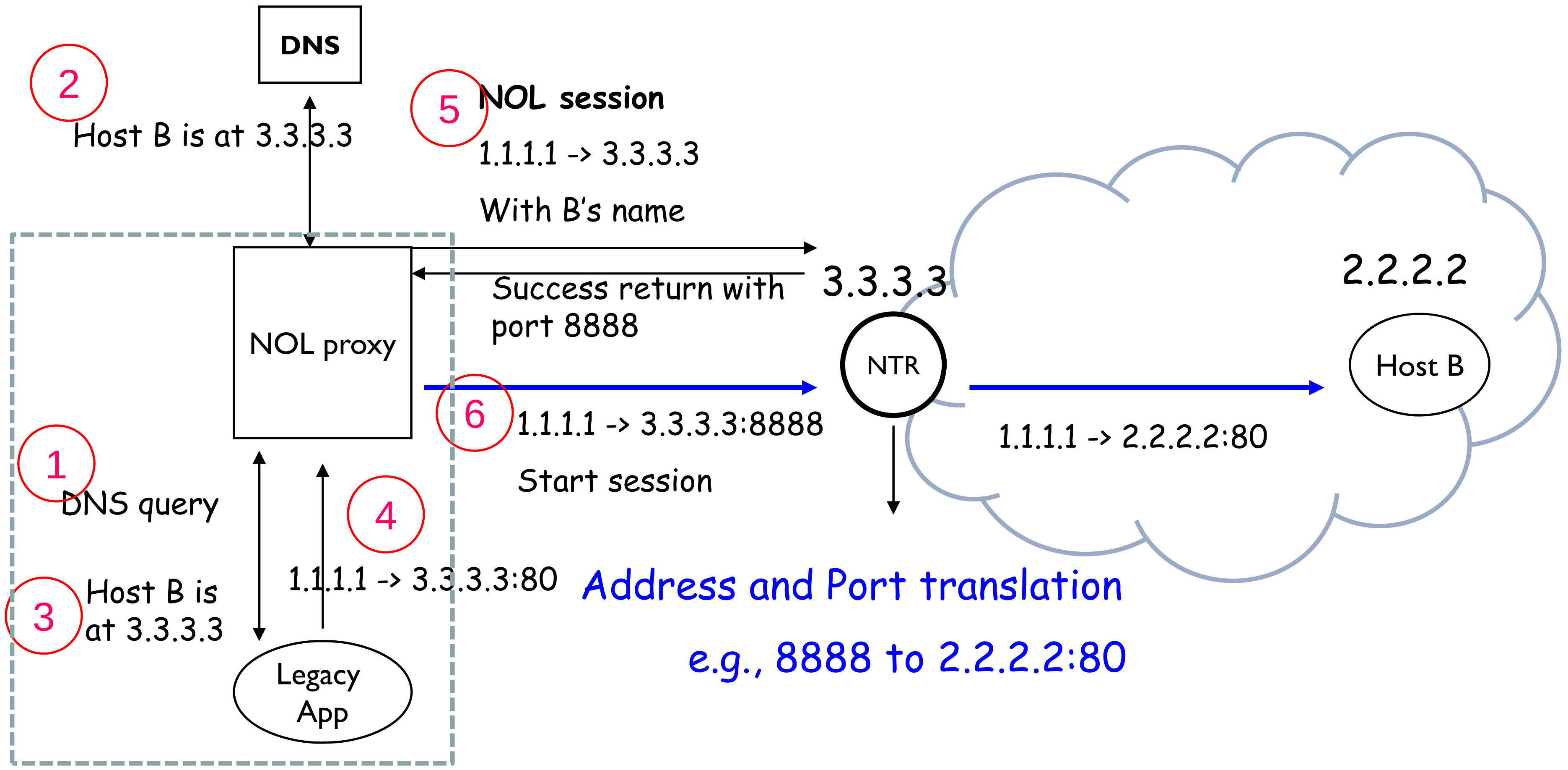
- First step, deploy NTR without blocking PI addresses. Performance-aware applications could use NOL service to optimize performance by multi-path transport through distributed NTRs.
- Second step, after widely deployed, we begin to charge for announcing PI addresses into upstream providers, and to block part of PI addresses without payment

## Backup

# How does legacy applications work (2)

We could also deploy NOL proxy gateway for legacy applications

NOL Proxy initiate access to host B, and translate 1.1.1.1->3.3.3.3:80 to 1.1.1.1->3.3.3.3:8888


DNS
2
Host B is at 3.3.3.3
5
NOL session
1.1.1.1 -> 3.3.3.3
With B's name
Success return with port 8888
3.3.3.3
2.2.2.2
NOL proxy
NTR
Host B
6
1.1.1.1 -> 3.3.3.3:8888
Start session
1.1.1.1 -> 2.2.2.2:80
1
DNS query
4
3
Host B is at 3.3.3.3
1.1.1.1 -> 3.3.3.3:80
Legacy App
Address and Port translation
e.g., 8888 to 2.2.2.2:80