Name based sockets

Javier Ubillos
Swedish Institute of
Computer Science

Zhongxing Ming
Tsinghua University

July 25th, 2010

http://www.ietf.org/id/draft-ubillos-name-based-sockets-01.txt

The general problem

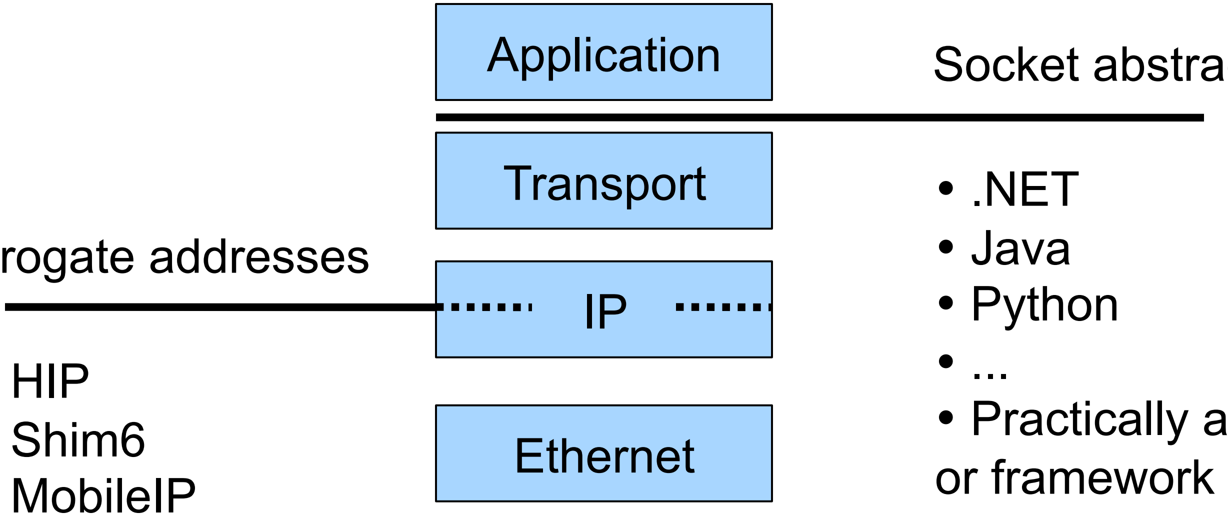
All IP (locator) management is done by the application.

There for, all interesting features need to be implemented by the application.

- Mobility
- Multi-homing
- IPv4/IPv6 interoperability
- NA(P)T traversal
- Path diversity exploitation
- Etc...

```
addr = gethostbyname( someString );
 ...;
connect( ...,  addr, ... );
write( ... );
close( ... );
connect( ..., addr, ...);
write( ... );
close( ... );
```

Two typical approaches



Socket abstractions

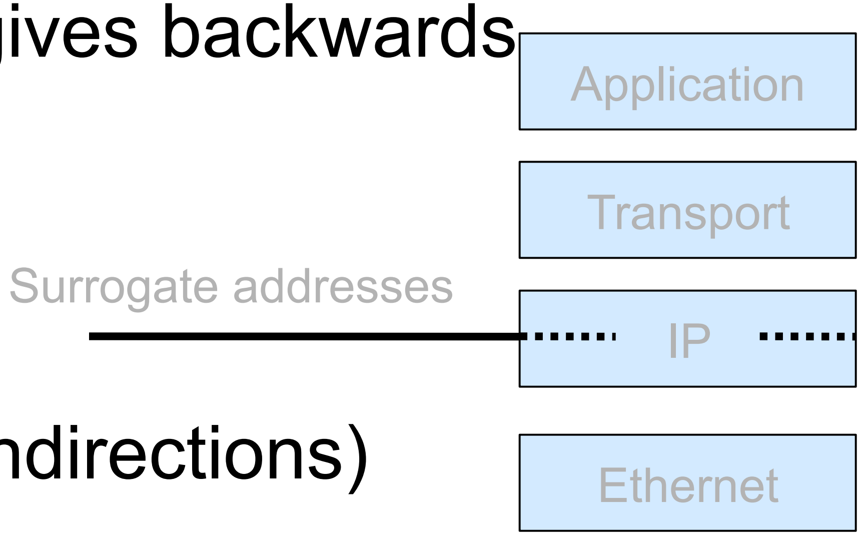
- .NET
- Java
- Python
- ...
- Practically any language or framework

Surrogate addresses

- HIP
- Shim6
- MobileIP
- Provider Independent addresses (e.g. with BGP)

Surrogate addresses

"Application transparency gives backwards compatibility (API)"



- Extra name spaces.
- Extra resolutions (more indirections)
- Applications are not aware, hence still might try to solve issues in app-space.

What do we want?

- No new indirections
- No new delays (e.g. first packet delay)
- Address management
  - Mobility
  - Multi-homing
  - Renumbering
  - IPv4/IPv6 interoperability
  - NAT penetration
- Backwards compatibility

API

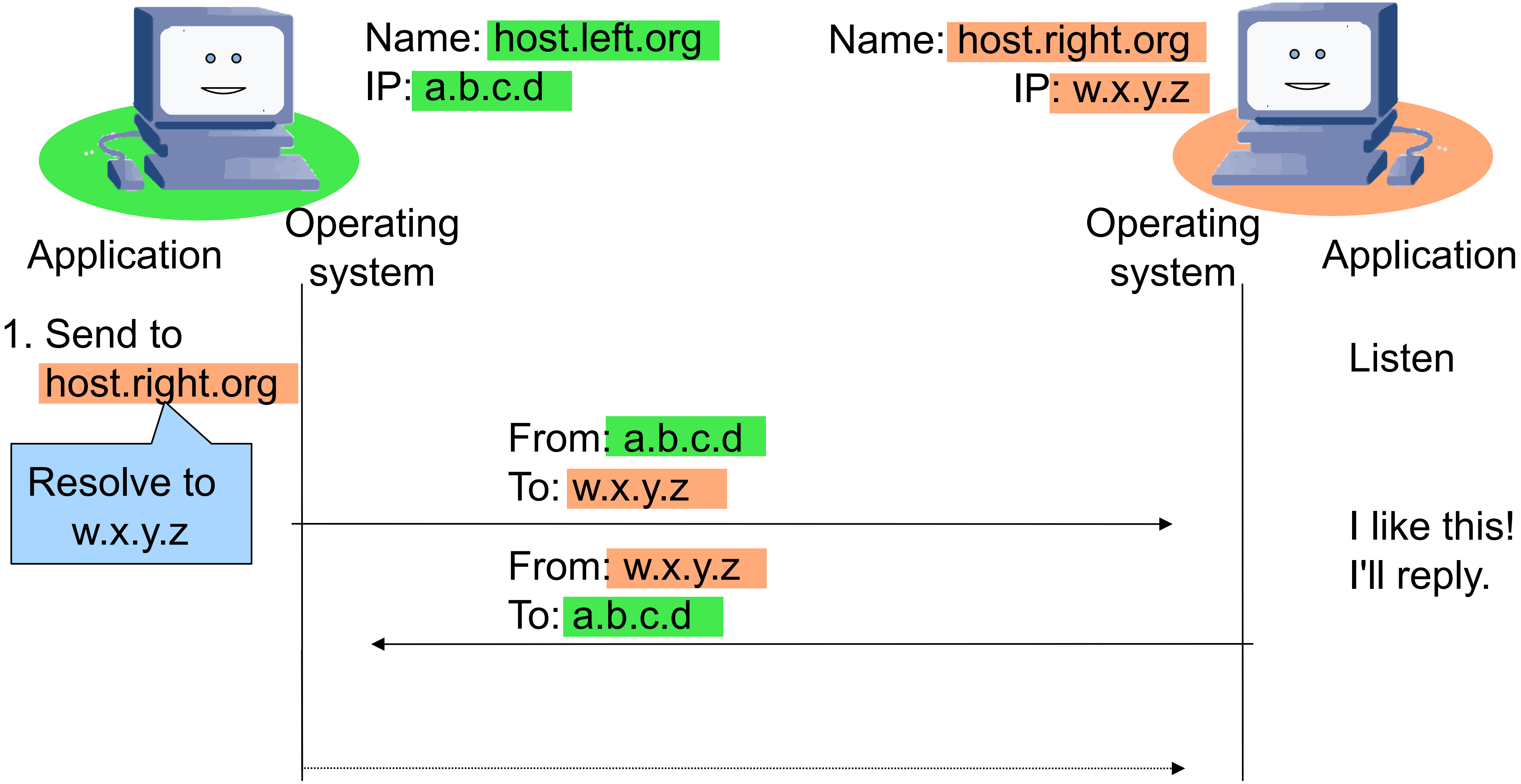
```
fd = socket( AF_NAME, SOCK_STREAM, 0);
struct sockaddr_name name_sock;

// Initialize name_sock with remote name

bind( fd, name_sock, sizeof(name_sock));
connect( fd, name_sock, sizeof(name_sock));

write(fd, send_buffer, len);
read(fd, recv_buffer, len);
```

The components (API)

- listen() - Prep for incoming session
  - fd = listen(local_name, peer_name, service, transport);
- open() - Initiate outgoing session
  - fd = open(local_name, peer_name, service, transport);
- **accept()** - Receive incoming session
  - **accept(peer_name, fd);**
- read() - Receive data
  - data = read(fd);
- write() - Send data
  - write(fd, data);
- close() - Close session
  - close(fd);

Initial name exchange

Traditional



Name: host.left.org
IP: a.b.c.d

Name: host.right.org
IP: w.x.y.z

Application

Operating system

Operating system

Application

1. Send to host.right.org

Resolve to w.x.y.z

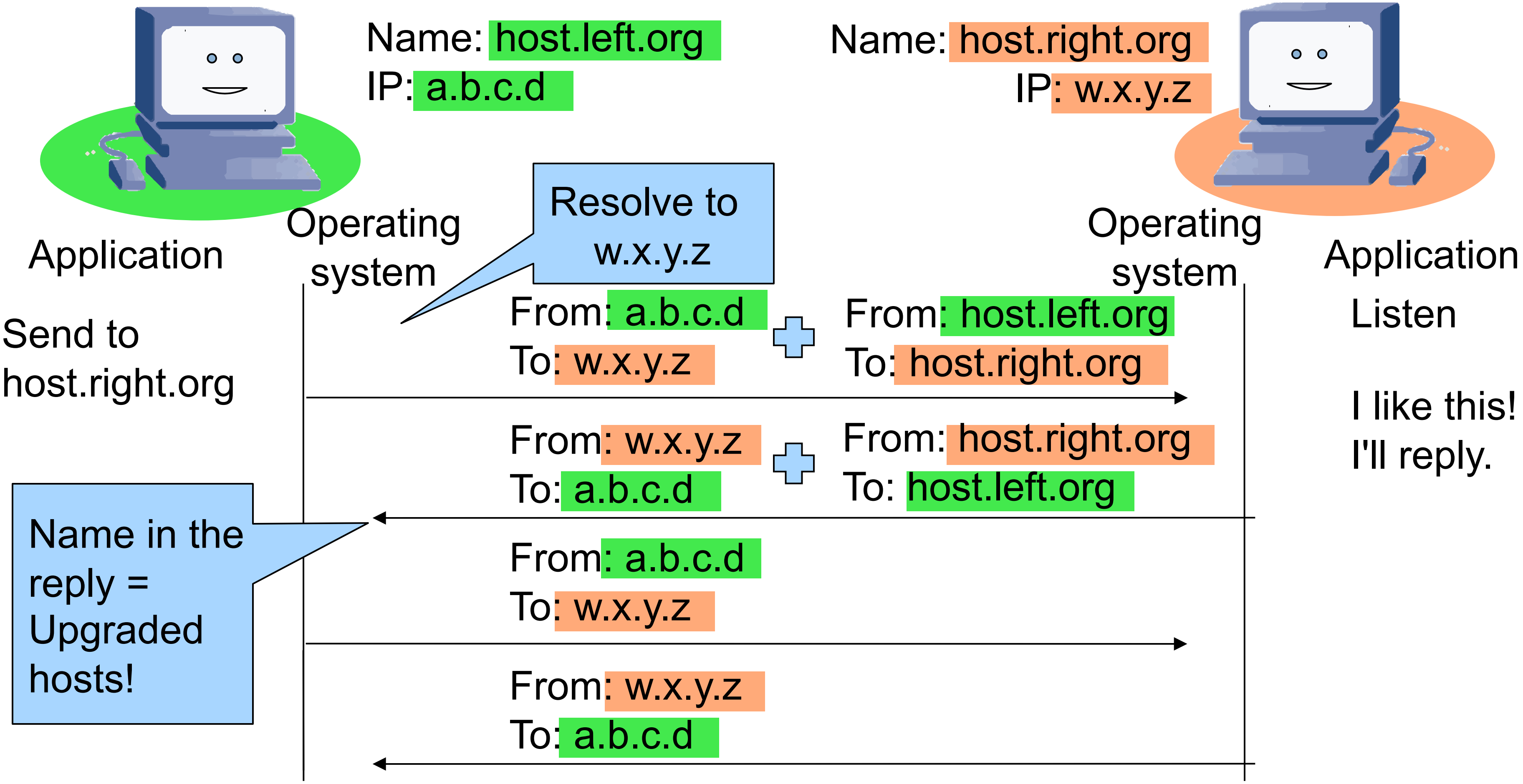
Listen

From: a.b.c.d
To: w.x.y.z

From: w.x.y.z
To: a.b.c.d

I like this!
I'll reply.

Name exchange



Name: host.left.org
IP: a.b.c.d

Name: host.right.org
IP: w.x.y.z

Application

Operating system

Resolve to w.x.y.z

Operating system

Application

Send to host.right.org

Listen

From: a.b.c.d
To: w.x.y.z

From: host.left.org
To: host.right.org

I like this!
I'll reply.

From: w.x.y.z
To: a.b.c.d

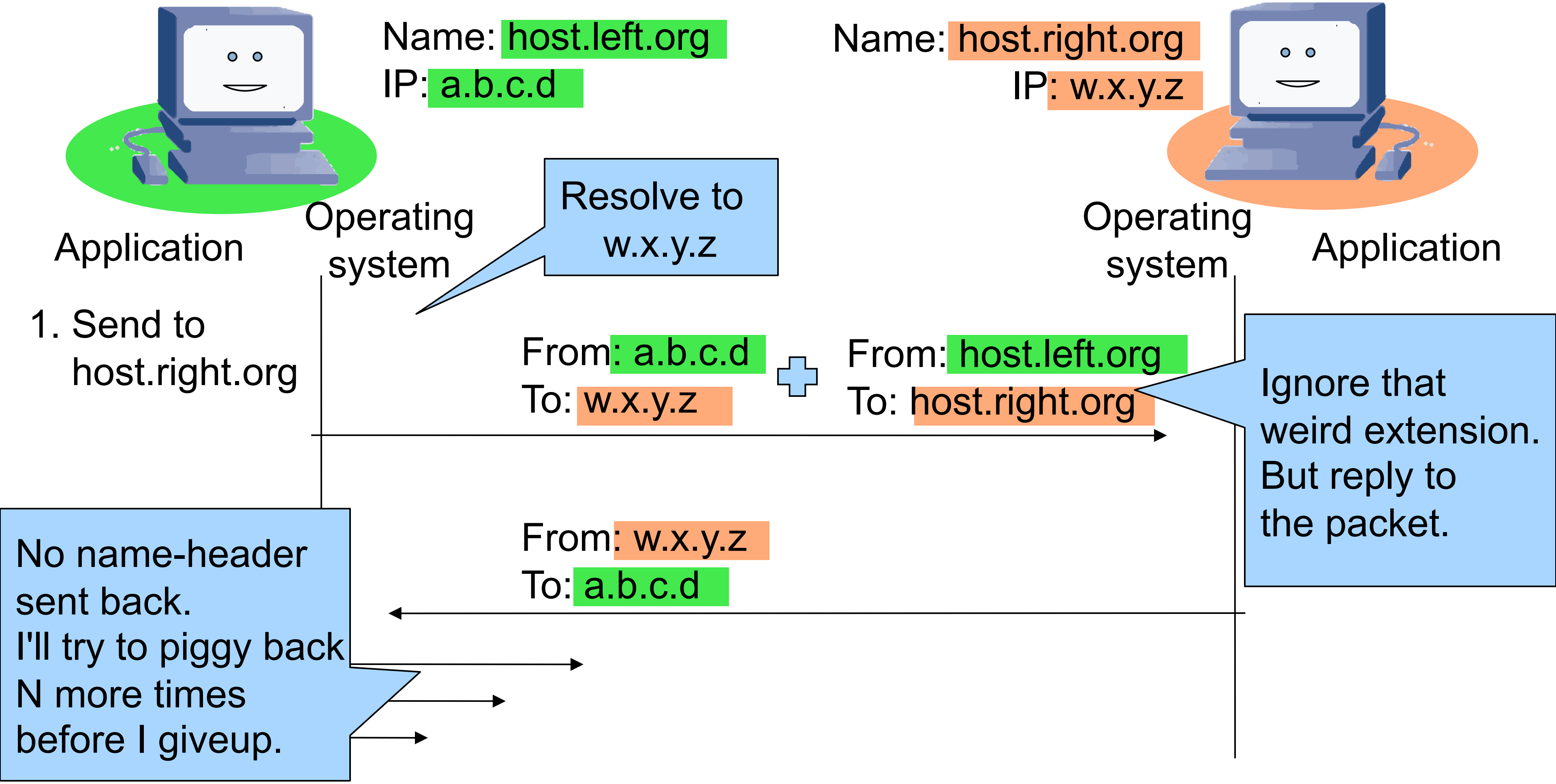
From: host.right.org
To: host.left.org

Name in the reply = Upgraded hosts!

From: a.b.c.d
To: w.x.y.z

From: w.x.y.z
To: a.b.c.d

Backwards compatibility



Name: host.left.org
IP: a.b.c.d

Name: host.right.org
IP: w.x.y.z

Application

Operating system

Resolve to w.x.y.z

Operating system

Application

1. Send to host.right.org

From: a.b.c.d
To: w.x.y.z

+

From: host.left.org
To: host.right.org

Ignore that weird extension. But reply to the packet.

From: w.x.y.z
To: a.b.c.d

No name-header sent back.
I'll try to piggy back N more times before I giveup.

The current prototype

- Supports TCP
  - Uses TCP semantics
    - socket(), listen(), open(), accept(), read(), write()
- Supports Shim6
  - Well, to a certain extent, we are working on it :)
- Exchanges names
- Linux
  - Ubuntu (client/server)
  - Android (client)

Implementation by Juan Lang (UC Davis)
and by Zhongxing Ming (Tsinghua University)

Current development

- Support for UDP
  - Using TCP-like semantics
- Mobility/Multi-homing
  - Shim6
- Collaboration between
  - Ericsson
  - Tsinghua University
  - Swedish Institute of Computer Science

The road map

- IPv4/IPv6 Interoperability
- NAT penetration
- Path diversity utilization
- Naming resolution (depth)
  - Host
  - Application
  - Etc...

- And more... Do you have any suggestions? Please let us know!

Mobile NBS

The proposed name-based socket should provide applications with guaranteed mobility functionality.

This implies that the design should allow mobile devices to move from one network to another while maintaining the connection.

DNS and Shim6 is involved to support mobile NBS

- Shim6 for basic mobility solution(UCL implementation)
- DNS for concurrent move

Why Shim6?

Shim6 provides a general solution for multihoming

- Network layer, transparent to the upper layer protocols
- Mobility is just a special case of multihoming!
- RFC 5533, 5534

Benefits

- No triangular routing!
- Fast handover
- Good reliability – REAP Protocol (RFC 5534)

Security considerations

- CGA /HBA address

Why DNS?

An effective solution for the concurrent moving problem is to have a "stationary infrastructure" to provide address information for all mobile devices.

- Base station for cell phones
- Home agent for mobile IP

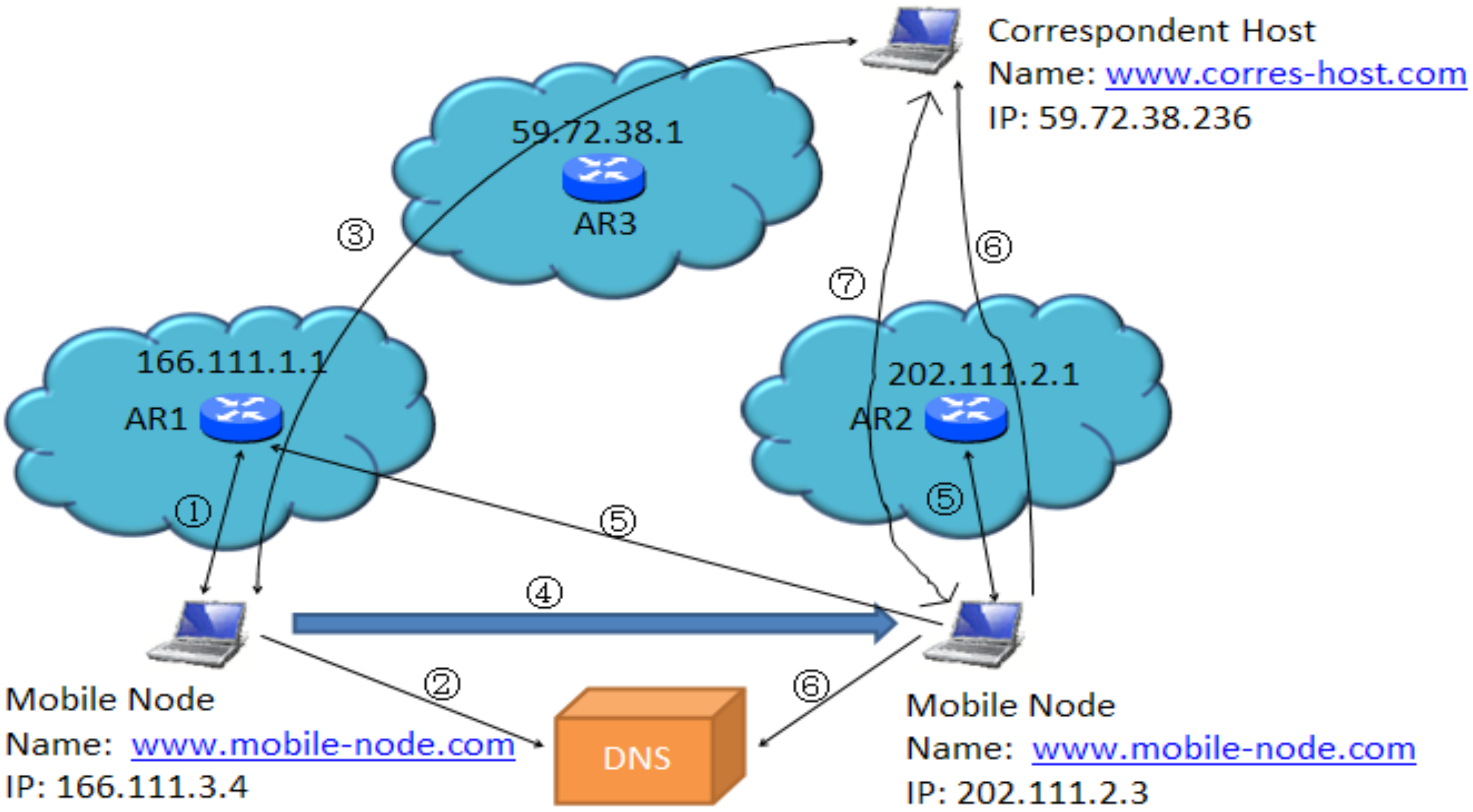
Less overhead is preferable

- Path stretch
- Latency

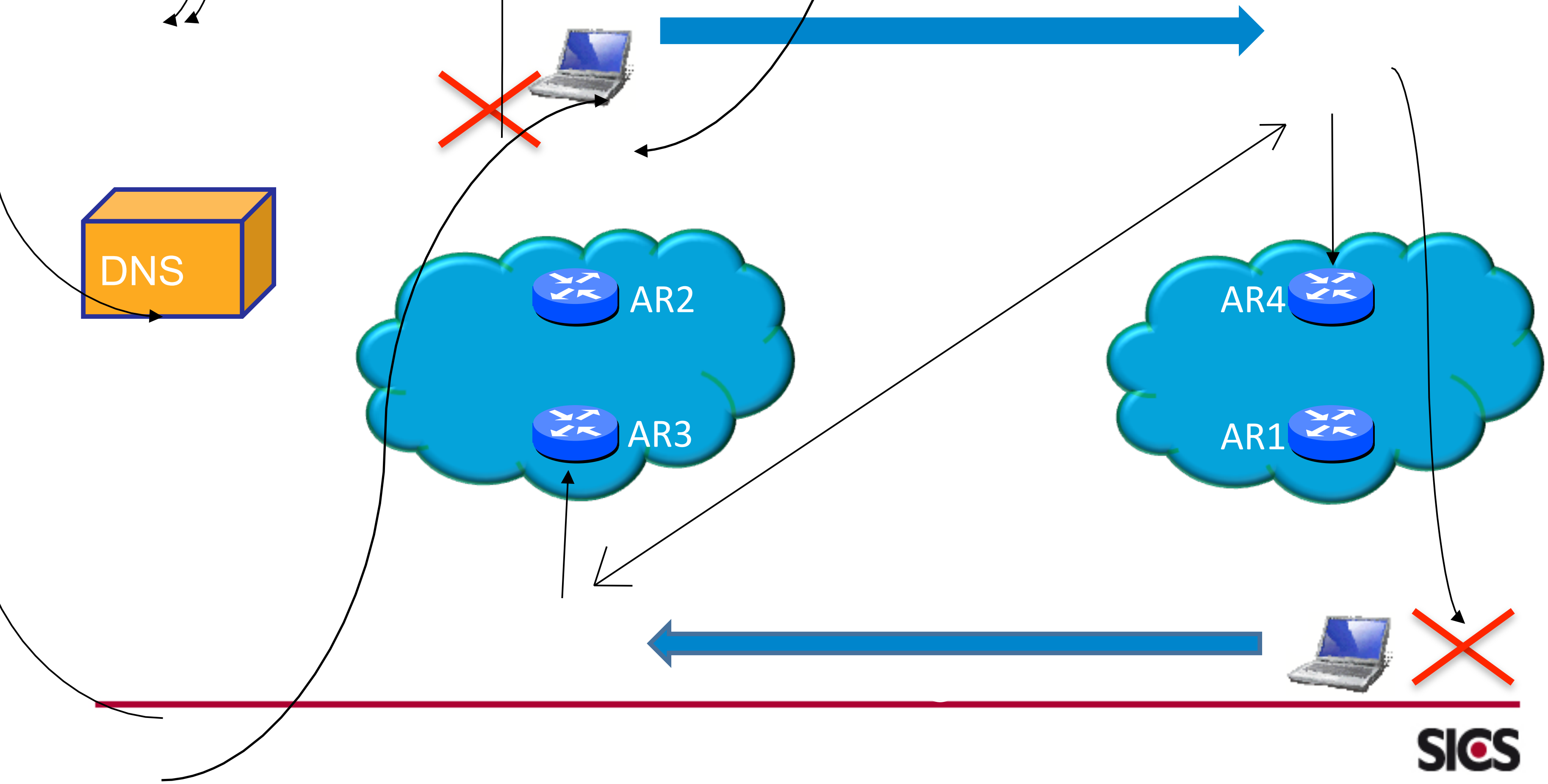
Why not DNS?

- Born for names
- NBS uses names!

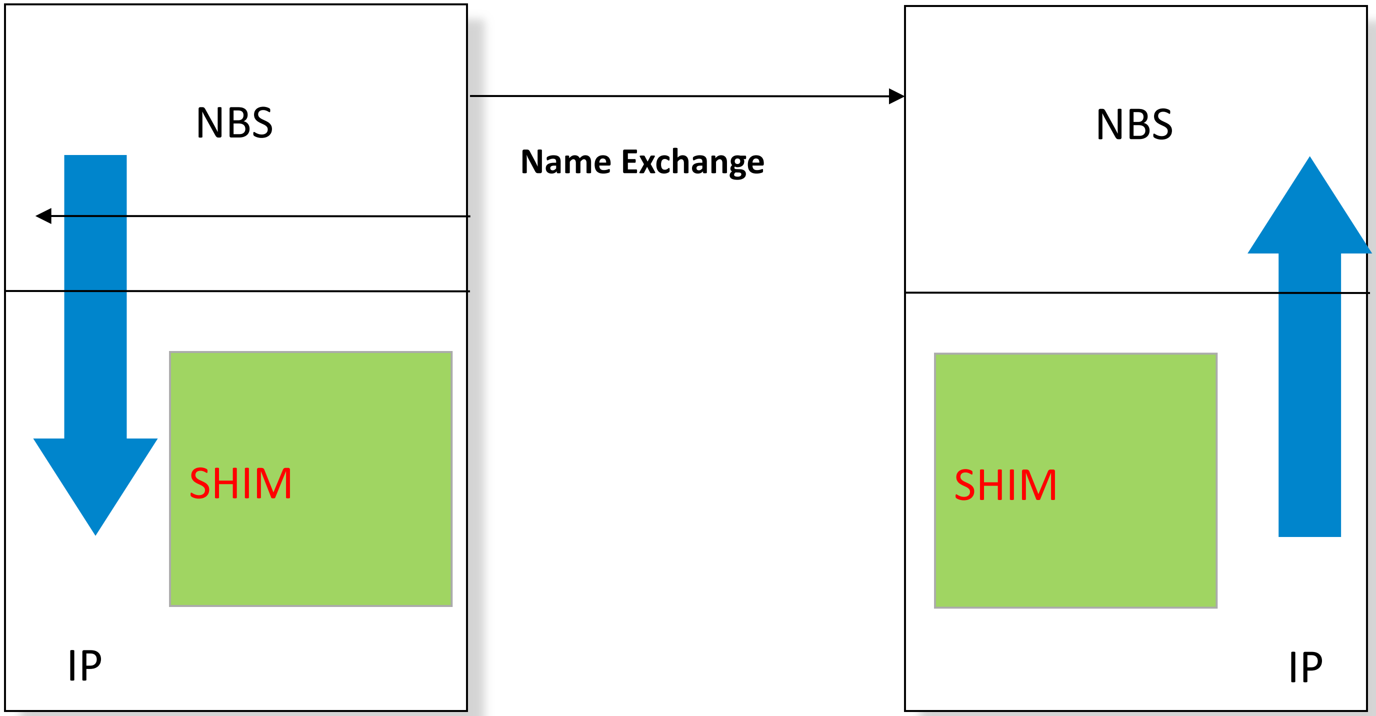
Mobile NBS – Basic Scenario



Concurrent Move

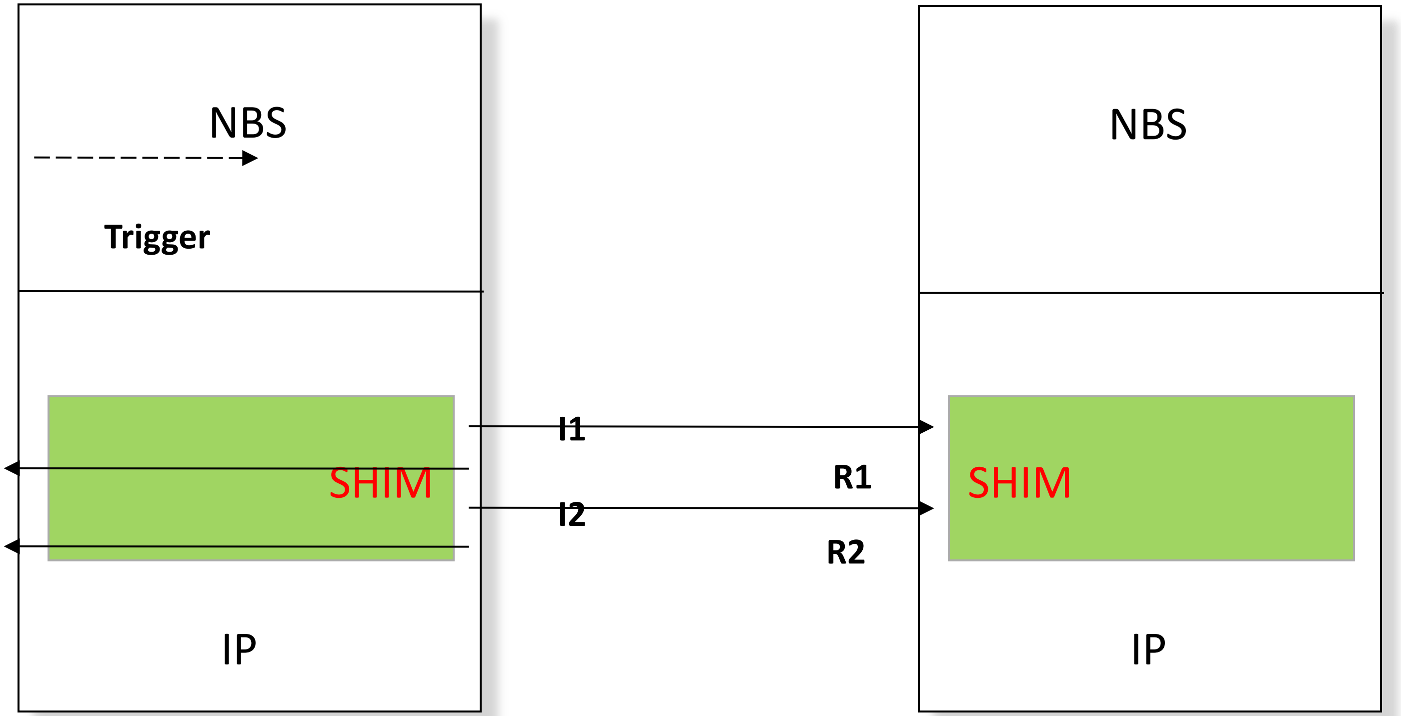


Initial Name Exchange with Shim6 Extension



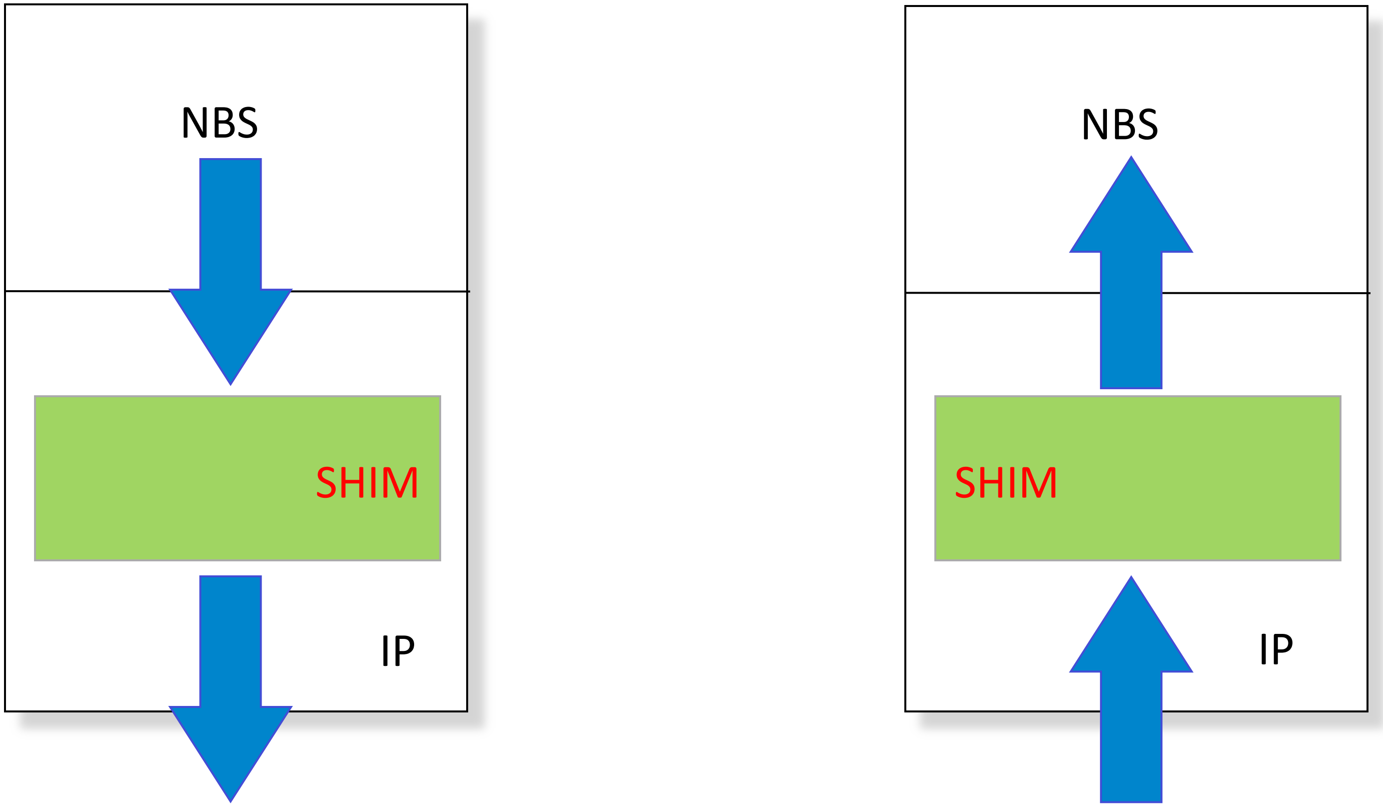
No SHIM state active

Context



Triggered

After Context Establishment



NBS + Shim6

Questions?