Name based sockets

apropos MPTCP API
Interface discussions

Late last year (nov/dec) there was a vivid discussion about API.

Provide a new API or keep the socket() interface untouched.

– In other words, change the semantics or not.
Not mutually exclusive

This is not a Yes/No question. MPTCP can have both!

For example HIP
– socket() API which is unchanged
– Native API

There is nothing to loose!
Not mutually exclusive

This is already what most developers use. The majority of frameworks provide socket abstractions.

Java / .NET / Python / You name it...
Name Based Stack

Started in RRG as a means to abstract locator substrate (IP) to permit multi-homing/mobility without

– Adding new infrastructure
– Impacting routing scalability

I would like to share what we did there and get your opinions.
Name Based Sockets !
The problem

FQDN resolution and IP management is dealt with by the application.

All cool stuff have to be implemented by the application.

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

addr = gethostbyname( someString );
... 
connect( ..., addr, ... );
write( ... );
close( ... );
connect( ..., addr, ... );
write( ... );
close( ... );
Two typical approaches

Surrogate addresses
- HIP
- Shim6
- MobileIP

Socket abstractions
- .NET
- Java
- Python
- Practically any framework
Surrogate addresses

“Application transparency gives backwards compatibility (API)”

• Extra namespaces.
• Extra resolutions (more indirections)
• Applications are not aware, hence still might try to solve issues in app-space.
Socket abstractions

Developers seem to like them...

• One implementation for every framework
• More often than not
  – Resolve once
  – Reuse IP
  – Reuse IP
  – Reuse IP
  – Reuse IP
  – Reuse IP
  – Reuse IP
  – Reuse IP
  – Reuse IP
  – Reuse IP
Cherry picking

- Provide the socket abstraction developers like.
- Do allow all the cool functions of surrogate addresses
  - But don't introduce new indirections
  - And be explicit about that it is different
Components

- API
- Initial name exchange
- More transport protocols (being worked on)
- Address updates (being worked on)
- Backwards compatibility (on the road map)
- Hosts without a registered DNS name (FQDN) (on the road map)
- Security (never ending story)
Components

- API
- Initial name exchange
- More transport protocols (being worked on)
- Address updates (being worked on)
- Backwards compatibility (on the road map)
- Hosts without a registered DNS name (FQDN) (on the road map)
- Security (never ending story)
The components (API)

- **listen()** - Prep for incoming session
  \[
  \text{fd} = \text{listen}( \text{src\_name}, \text{dst\_name}, \text{local\_port}, \text{transport} );
  \]

- **open()** - Initiate outgoing session
  \[
  \text{fd} = \text{open}( \text{src\_name}, \text{dst\_name}, \text{remote\_port}, \text{transport} );
  \]

- **accept()** - Receive incoming session
  \[
  ( \text{src\_name}, \text{dst\_name}, \text{fd} ) = \text{accept}( \text{fd} );
  \]

- **read()** - Receive data
  \[
  \text{data} = \text{read}( \text{fd} );
  \]

- **write()** - Send data
  \[
  \text{write}( \text{fd}, \text{data} );
  \]

- **close()** - Close session
  \[
  \text{close}( \text{fd} );
  \]
Components

• API
• Initial name exchange
• More transport protocols (being worked on)
• Address updates (being worked on)
• Backwards compatibility (on the road map)
• Hosts without a registered DNS name (FQDN) (on the road map)
• Security (never ending story)
Initial Name Exchange

host.left.net has address a.b.c.d

peer.right.net has address v.x.y.z

Application

Operating System

open from host.left.net to peer.right.net

resolve peer.right.net

listen from any to peer.right.net

verify host.left.net

accept from host.left.net to peer.right.net

source a.b.c.d destination v.x.y.z

source v.x.y.z destination a.b.c.d

source a.b.c.d destination v.x.y.z

2010-03-26
Initial Name Exchange

**host.left.net** has address **a.b.c.d**

**peer.right.net** has address **v.x.y.z**

**application**

- open from **host.left.net** to **peer.right.net**
- write
- send names until receive packet

**operating system**

- source **a.b.c.d** destination **v.x.y.z** from **host.left.net** to **peer.right.net**
- read

**application**

- listen from any to **peer.right.net**
- accept from **host.left.net** to **peer.right.net**
- read
- write

**operating system**

- source **v.x.y.z** destination **a.b.c.d** from **peer.right.net** to **host.left.net**
- send names until receive packet without names
Backwards Compatibility

- `host.left.net` has address `a.b.c.d`
- `legacy host` has address `v.x.y.z`

**Diagram**:
- Application
  - Operating System
    - Open from `host.left.net` to `v.x.y.z.names`
    - Resolve `peer.right.net`
    - Read from `host.left.net` to `v.x.y.z.names`
- Operating System
  - Application
    - Bind
    - Listen
    - Accept from `a.b.c.d`
    - Write
    - Source `a.b.c.d` destination `v.x.y.z`
    - Source `v.x.y.z` destination `a.b.c.d`
    - Source `v.x.y.z` destination `a.b.c.d`
Components

• API
• Initial name exchange
• More transport protocols (being worked on)
• Address updates (being worked on)
• Backwards compatibility (on the road map)
• Hosts without a registered DNS name (FQDN) (on the road map)
• Security (never ending story)
Current development

• Support for UDP
  – Using TCP-like semantics

• Mobility/Multi-homing
  – Evaluating existing solutions
    • Shim6, MIPv6, MPTCP or something else entirely.

• Collaboration between
  – Ericsson
  – Tsinghua University
  – Swedish Institute of Computer Science
The current prototype

• Supports TCP
  – Uses TCP semantics
    • socket(), listen(), open(), accept(), read(), write()

• Exchanges names

• Linux
  – Ubuntu (client/server)
  – Android (client)

Implemented by Juan Lang.
Components

• API
• Initial name exchange
• More transport protocols (being worked on)
• Address updates (being worked on)
• Backwards compatibility (on the road map)
• Hosts without a registered DNS name (FQDN) (on the road map)
• Security (never ending story)
Security

- Initial name exchange
  - Trivial to forge your own name
    DNS verification required
  - Same weakness as for initiating host

a. Is it acceptable security? (I think yes)

b. Does it even matter?
  - I'm playing with the thought that maybe it might not matter (I'm open for flames :)

Questions?