Raconn - Ssh multi hostname

Posted on 2025-10-24 6 minutes read

TLDR: Raconn

Introduction

Recently I wanted to connect to a local machine with ssh but didn't know it's IP, I knew the MAC address which I could translate into the IPv6 link-local address but since the `connect` syscall requires the local iface name for ip6 link local, it would need to be hardcoded in the ssh config, making it inconvenient to change. Also if the same machine is accessible from a wider network, I'd need a second entry in the config and have to remember which one to use depending on which network my computer is connected to.

That’s a lot of complexity and I thought most of it could be solved if ssh config allowed multiple hostnames for a single host entry.

Unfortunately, `OpenSSH`'s `Hostname` directive doesn't support multiple entries:

```
Host machineA
  Hostname machinea.local 192.168.1.65 2a00:1450:4007:81a::200e 142.250.201.174 00:00:07:00:00:00 ...
  Port 22
  User admin
  PubkeyAuthentication yes
  IdentityFile ~/.ssh/id_rsa
```

But there’s a silver lining `ProxyCommand`. It runs a program that’s expected to connect to the remote machine, `OpenSSH` will simply speaks its protocol through the program's `stdin` and `stdout`!

Which means I can do the following:

```
Host machineA
  ProxyCommand program %p machine.company.net 192.168.1.65 2a00:1450:4007:81a::200e 142.250.201.174 00:00:07:00:00:00 ..
  Port 22
  User admin
  PubkeyAuthentication yes
  IdentityFile ~/.ssh/id_rsa
```

I just need to figure out what `program` is.

Socat

It seems like a shell script using `socat` would be perfect for the job, I was able to get a serialized version working, but since it tried every hostname one after another, it could take a long time to connect, especially if only the last one on the command line was reachable.

The main issue I had was that socat executes the script defined on the `exec` statement before the connection is successfully established making it hard to know which socat instance is connected when spawning multiple ones in parallel.

stackoverflow

In the end, I decided to write this program myself.

Raconn

Enter `raconn` (short for RAce CONNect) is a utility that accepts a list of hostnames and IP addresses as arguments. It attempts to connect to all of them in parallel using separate threads and retains the first successful connection cancelling all the remaining threads. Once connected, it proxies data between stdin and stdout, making it suitable for use as an SSH ProxyCommand.

Git repository

It is written in C using cosmopolitan libc, Some reasons for this choice:

- Runs on all major operating systems with a unique static binary
- Easy 'C' development thanks to sane api defaults
- Debug enviroment included

Also I'm addicted to sugar so I added pretty.c in the mix.

Examples

example

Let’s say you have a server with two ifaces, one used for failover, if you specify both ip to raconn, even in a failover situation, your ssh command will succeed by connecting through the second iface and ip. `$ ssh ip4failover`

```
Host ip4failover
  ProxyCommand raconn %p 233.252.3.1 233.252.4.1
  Port 22
  User admin
```

example

You have a machine that is accessible on your local network and from outside. You only known the machine MAC address and its name which resolve to a WAN ip. Locally your are connected through both wifi and ethernet. `$ ssh machine`

```
Host machine.company
  ProxyCommand raconn %p machine.company.net 00:00:07:cf:10:0a
  Port 22
  User admin
```

example

You're trying to figuring out something and `raconn` is getting in the way. `$ ssh -oProxyCommand=none machine.company.net`

```
Host machine.company.net
  ProxyCommand raconn %p 2a00:1450:4007:81a::200e fe80::9e4f:642a:15c2:aad7
  Port 22
  User admin
```

How is it made?

Writing C and dev environment setup

Git

`Raconn` is a small project and it only needs only a few versioned files. For the first time I tried using `.gitignore` as an allowlist:

```
# ignore everything
*
# except for these
!/.gitignore
!/main.c
!/Makefile
!/README.md
!/LICENSE
```

This has the nice property that no new files or artifacts can be tracked by mistake, and `git status` always stays clean.

Makefile

For easy build and QA, I use a `Makefile` with targets for build, format and lint.

It will also downloads the `cosmopolitan` toolchain and sets up the LSP support to get errors and warnings directly in kate my source editor of choice.

After some research, I found that I needed to create two files for `clangd` LSP to correctly support `cosmopolitan`:

```
[]
```

and

```
CompileFlags:
        Add:
                - "-isystemcosmocc/include"
                - "-include libc/c.inc"
```

Principle

`raconn` tries to connect in parallel to all addresses provided on the command line. The first successful connection cancels the others and is then piped through stdin/stdout, making it suitable for use as an ssh `ProxyCommand`.

The first step is to resolve each hostname into a list of IPv4 or IPv6 addresses to feed into the `connect` syscall.

Ipv6 local link

Some notes on IPv6 link local addresses,

IpV6 has a set of special addresses called link local in the range `fe80::/10`. All network interface must be assigned one if it wants to use Ipv6. These addresses are not routed by routers and allow for ipv6 devices to talk to each other without a router on the same link.

Because ip6 link local addresses are usually derived from the network interface's mac address, that makes them predictable and so it would be nice to be able to use them to connect to the remote machine.

The problem is that, unlike IPv4 and global IPv6 addresses (where only the destination address is needed), link local addresses require specifying the interface index in `struct sockaddr_in6` when using `connect`.

Since I don’t want to ask the user which network interface shares a link with the remote machine, I simply perform a `connect` syscall for every network interface on the local machine.

So a list of locally attached network interfaces is needed, but as of 2025 `getifaddrs` in cosmopolitan is only implemented for IPv4.

That became a whole side quest, implemented across 3 PRs:

- getifaddrs Linux (merged)
- getifaddrs BSD
- getifaddrs Windows

DNS resolution

In cosmopolitan, hostname lookup using `getaddrinfo` is the courtesy of the musl libc implementation. As of 2025, it doesn't support `mDns` :/

`getaddrinfo` returns a linked list of `struct addrinfo` which are the ips 4 and 6 the name resolve to.

If it returns an Ipv6 link-local address we have to enter the special case described earlier.

Parallel connections

Because `getaddrinfo` `getaddrlist` `connect` etc are blocking calls, I spawn one thread per arguments on the command line each with a small stack size (see cosmo's greenbean).

This way all names are tried simultaneously, making the connections truly racy.

IO fun

To ensure only one connection is kept, I use an atomic variable that is only set once.

The winning connection then spawns two final threads that that effectively pipe the socket through stdin/stdout.

Timeout

Since `raconn` will mostly be used in interactive contexts, I want it to fail fast when no hostname is reachable. Unfortunately, Posix's `connect` doesn't give any easy way to choose a timeout after which the connect should fail.

So, I spawn a sibling thread before calling `connect` and `getaddrinfo` that will `pthread_cancel` on the connecting thread after a timeout. If `connect` succeeds first, it cancels the timeout thread instead.

Closing words

You can find `raconn` here.

Computer addressing is a bit of a mess, but at least it’s possible to abstract some of it once you understand its quirks.
