Optimizing TincVPN

Derek Chiang, Jasdeep Hundal, Jisun Jung
Motivation

- Supercloud requires a performant networking solution
  - Tinc is easy to configure, but slow
Tinc Architecture

- User Application
- Tinc Daemon
- Tinc Virtual NIC
- (Paravirtualized) Host NIC

User → Tinc Daemon → Kernel
Kernel → Tinc Virtual NIC → (Paravirtualized) Host NIC → To Network
Tinc Architecture Continued

TCP Packet (from User)

Split

UDP Header + Seq # + Data

UDP Header + Seq # + Data

UDP Header + Seq # + Data

send() Called on each packet

To kernel
Tinc Architecture Continued

- Network topology
- Host configuration
- Packet reassembly
Profiling - Kernel Calls w/ Tinc
Profiling - Tinc Function Calls

.tincd
Total samples: 1363
Focusing on: 1363
Dropped nodes with <= 6 abs(samples)
Dropped edges with <= 1 samples

_sendto
166 (12.2%)

__write_nocancel
166 (12.2%)
Potential Optimizations

- `sendmsg` -> `sendmmsg` (buffering)
- `select` -> `epoll`
- more efficient algorithms / data structures
- `event loop` -> `multithreading`
- use more efficient socket implementation
Actual Optimizations

- `sendmsg -> sendmmsg (buffering)`
- `select -> epoll`
Buffered Architecture

Data (from User) → Split → Header + Seq # + Data

sendmmsg() → Called once → Buffer

To kernel
Optimizations Continued

- Parallelized send did not work
- Switch from using select to using epoll
Results

- epoll: consistently about 5% faster
- sendmmsg: single host test was promising
Future Work

- Improve ‘one button’ testability
- Increase MTU
- Take advantage of multicore systems
- In-kernel VPN
Demo!

- Script for performance profiling between two hosts running tinc