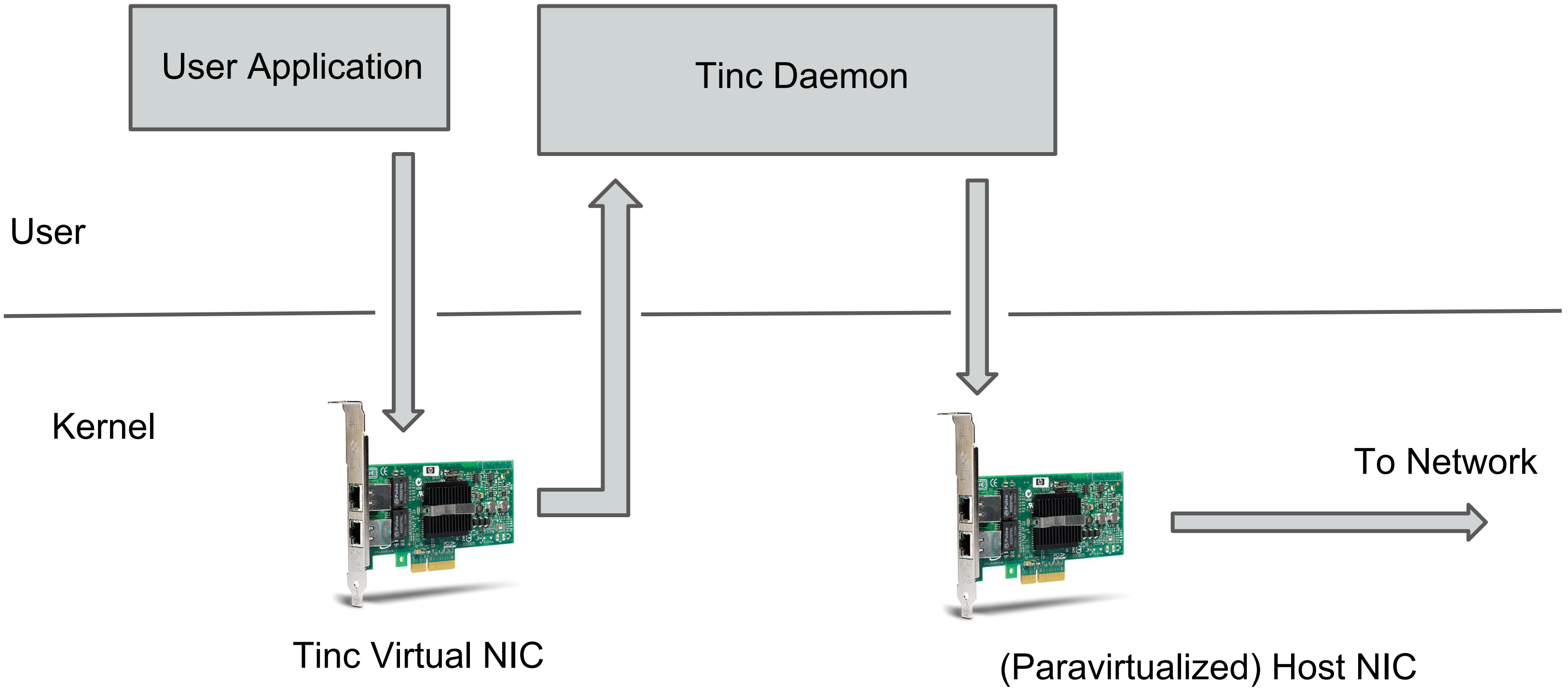
Optimizing TincVPN

Derek Chiang, Jasdeep Hundal, Jisun Jung

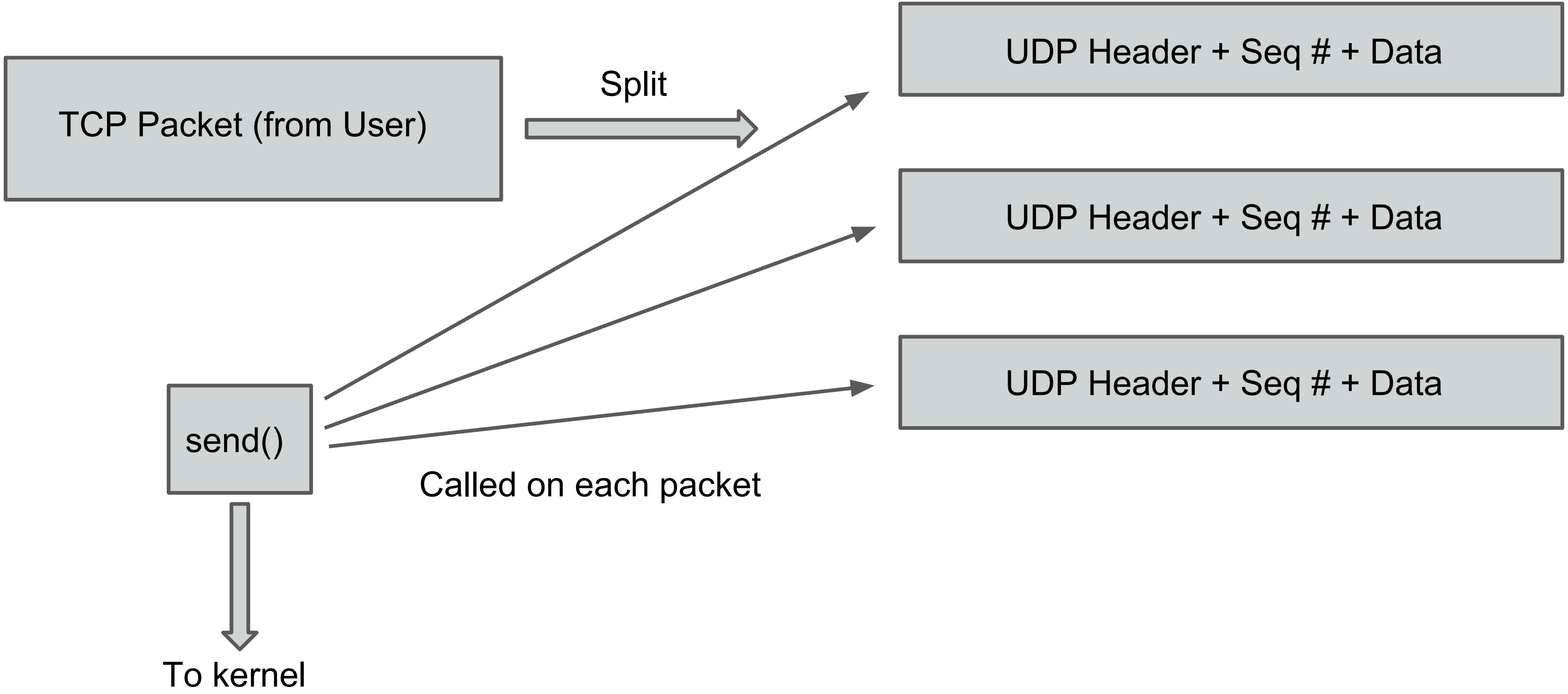
Motivation

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

Tinc Architecture



Tinc Architecture Continued



TCP Packet (from User)

Split

UDP Header + Seq # + Data

UDP Header + Seq # + Data

UDP Header + Seq # + Data

send()

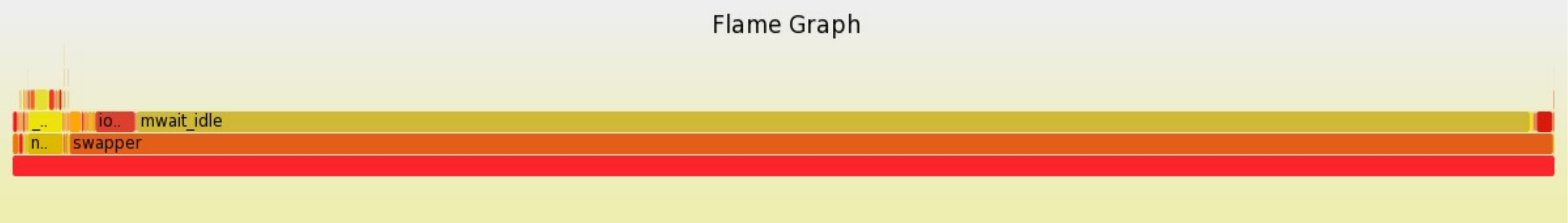
Called on each packet

To kernel

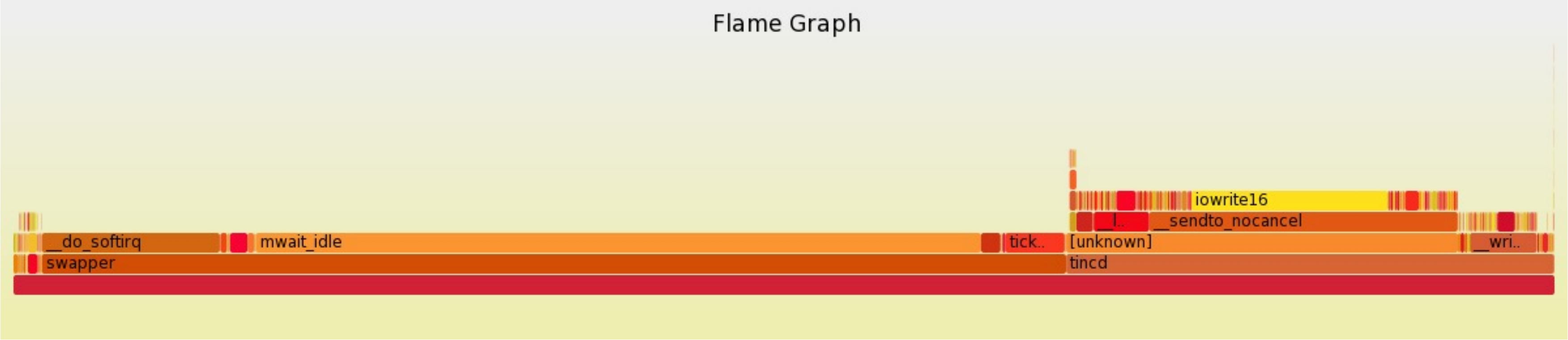
Tinc Architecture Continued

- Network topology
- Host configuration
- Packet reassembly

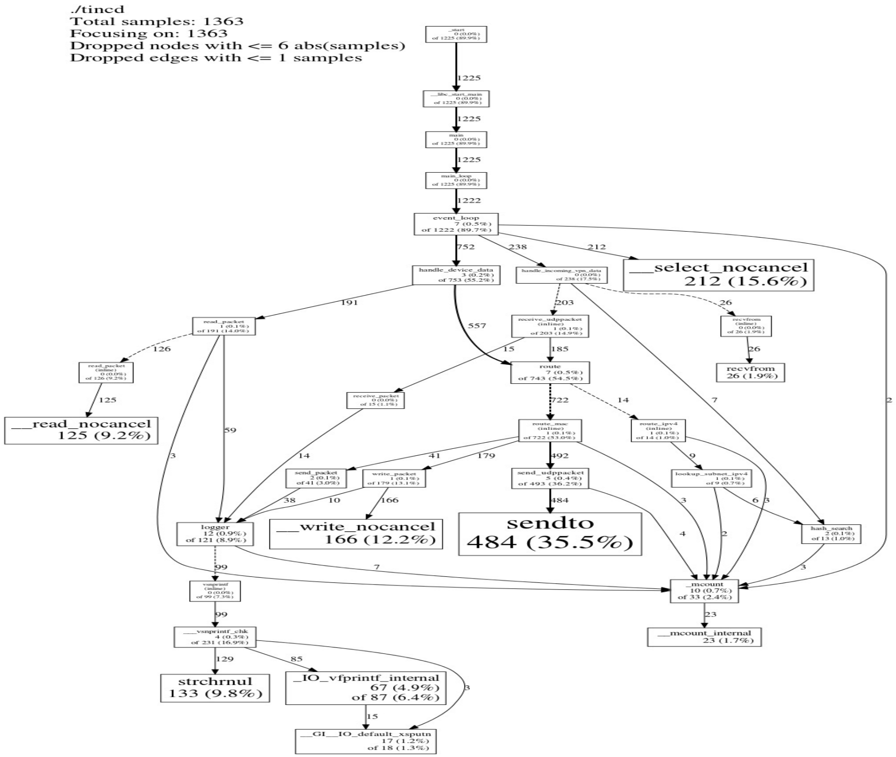
Profiling - Kernel Calls w/o Tinc



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
```

_start 0 (0.0%) of 1225 (89.9%)
1225
__libc_start_main 0 (0.0%) of 1225 (89.9%)
1225
main 0 (0.0%) of 1225 (89.9%)
1225
main_loop 0 (0.0%) of 1225 (89.9%)
1222
event_loop 7 (0.5%) of 1222 (89.7%)
752
238
212
handle_device_data 3 (0.2%) of 753 (55.2%)
handle_incoming_vpn_data 0 (0.0%) of 238 (17.5%)
___select_nocancel 212 (15.6%)
191
203
26
read_packet 1 (0.1%) of 191 (14.0%)
557
receive_udppacket (inline) 1 (0.1%) of 203 (14.9%)
recvfrom (inline) 0 (0.0%) of 26 (1.9%)
126
15
185
26
read_packet (inline) 0 (0.0%) of 126 (9.2%)
route 7 (0.5%) of 743 (54.5%)
recvfrom 26 (1.9%)
receive_packet 0 (0.0%) of 15 (1.1%)
125
722
14
7
2
___read_nocancel 125 (9.2%)
59
3
route_mac (inline) 1 (0.1%) of 722 (53.0%)
route_ipv4 (inline) 1 (0.1%) of 14 (1.0%)
14
41
179
492
9
send_packet 2 (0.1%) of 41 (3.0%)
write_packet 1 (0.1%) of 179 (13.1%)
send_udppacket 5 (0.4%) of 493 (36.2%)
lookup_subnet_ipv4 1 (0.1%) of 9 (0.7%)
38
10
166
484
3
6
3
logger 12 (0.9%) of 121 (8.9%)
___write_nocancel 166 (12.2%)
sendto 484 (35.5%)
4
2
hash_search 2 (0.1%) of 13 (1.0%)
99
7
3
vsnprintf (inline) 0 (0.0%) of 99 (7.3%)
_mcount 10 (0.7%) of 33 (2.4%)
99
23
___vsnprintf_chk 4 (0.3%) of 231 (16.9%)
__mcount_internal 23 (1.7%)
129
85
3
strchrnul 133 (9.8%)
_IO_vfprintf_internal 67 (4.9%) of 87 (6.4%)
15
__GI__IO_default_xsputn 17 (1.2%) of 18 (1.3%)

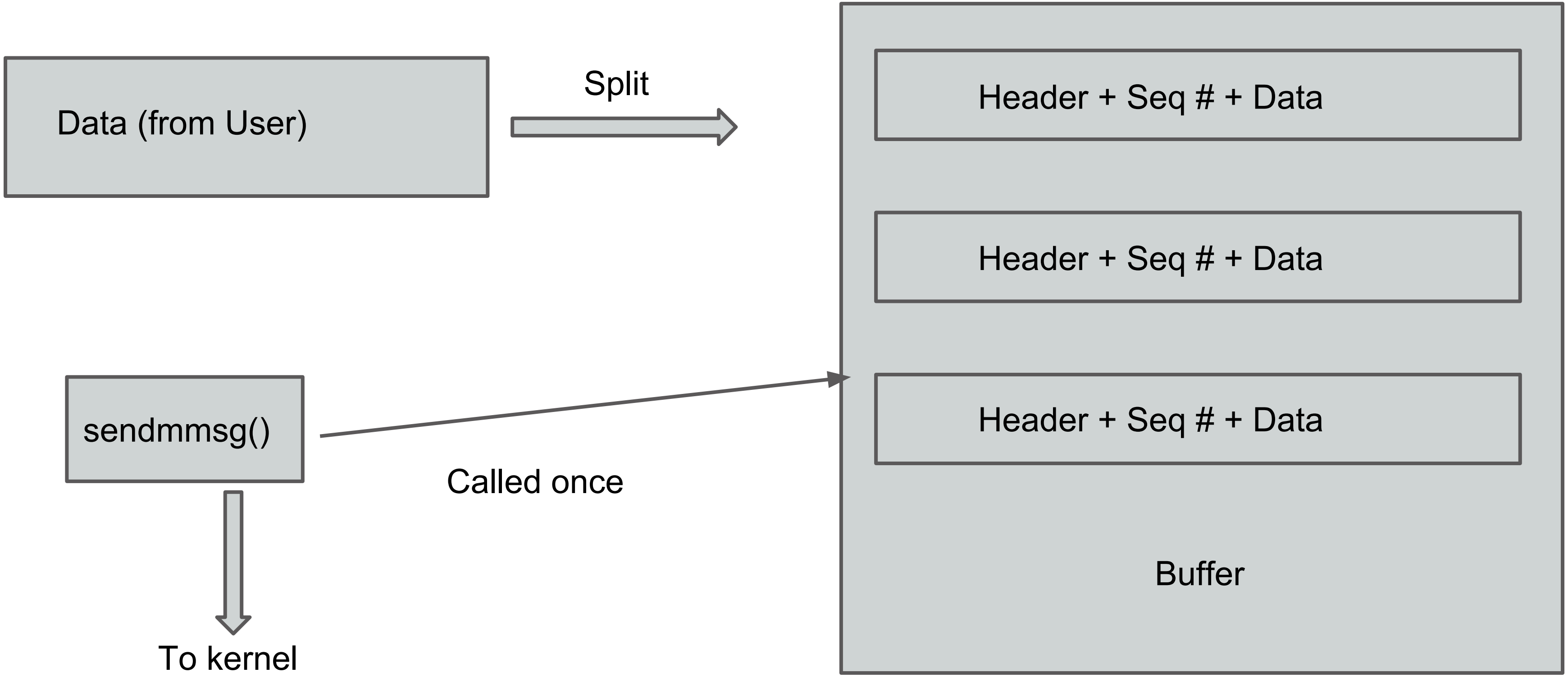
Potential Optimizations

- 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

Header + Seq # + Data

Header + Seq # + Data

Buffer

sendmmsg()

Called once

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