

Fastsocket

Speed up your socket

SINA

Xiaofeng Lin

jerrylin.lxf@gmail.com

Contributor

SINA:

Xiaofeng Lin
Xiaodong Li

Tsinghua:

Yu Chen
Junjie Mao
Jiaquan He

Socket API Problem

Socket API is fundamental for most network applications

- Kernel eats too much CPU cycles
- Less CPU cycles left to Application
- Call for more efficient kernel socket API

Performance on Multicore System

Theoretical Overall Performance:

*Single core performance * Available CPU cores*

How close can we reach the limit: ***Scalability.***

Outline

- Scalability Performance
- Single Core Performance
- Production System Feasibility
- Future Work

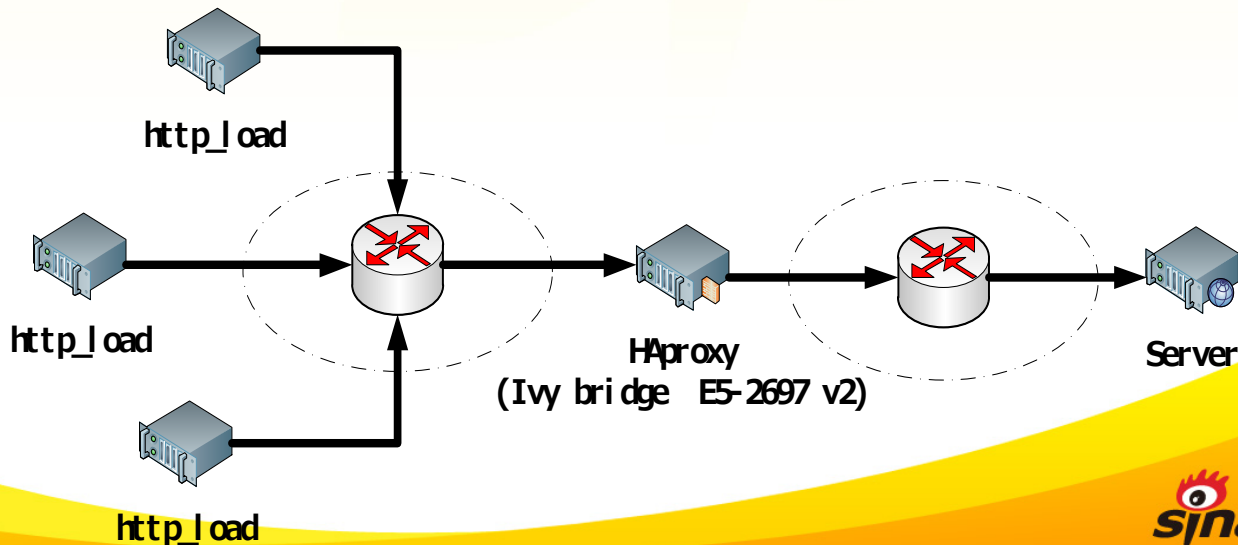
Fastsocket



- Scalability Performance
- Single Core Performance
- Production System Feasibility
- Future Work

Testing Environment

- HAProxy: Open source TCP/HTTP loadbalancer.
- OS: CentOS-6.2 (Kernel : 2.6.32-220.23.1.el6)
- CPU: Intel Ivy-Bridge E5-2697-v2 (12 core) * 2
- NIC: Intel X520 (Support Flow-Director)
- Scenario: short TCP connections



Kernel Inefficiency

More than 90% CPU is consumed by Kernel

```
top - 16:28:14 up 2:53, 1 user, load average: 23.25, 16.36, 11.30
Tasks: 472 total, 25 running, 447 sleeping, 0 stopped, 0 zombie
Cpu0  :  5.8%us, 81.5%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi, 12.7%si,  0.0%st
Cpu1  :  4.9%us, 82.8%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi, 12.3%si,  0.0%st
Cpu2  :  5.2%us, 82.5%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi, 12.3%si,  0.0%st
Cpu3  :  4.9%us, 83.4%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi, 11.7%si,  0.0%st
Cpu4  :  5.2%us, 82.5%sy,  0.0%ni,  0.3%id,  0.0%wa,  0.0%hi, 12.0%si,  0.0%st
Cpu5  :  5.2%us, 82.5%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi, 12.3%si,  0.0%st
Cpu6  :  4.9%us, 82.5%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi, 12.6%si,  0.0%st
Cpu7  :  4.5%us, 83.2%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi, 12.3%si,  0.0%st
```

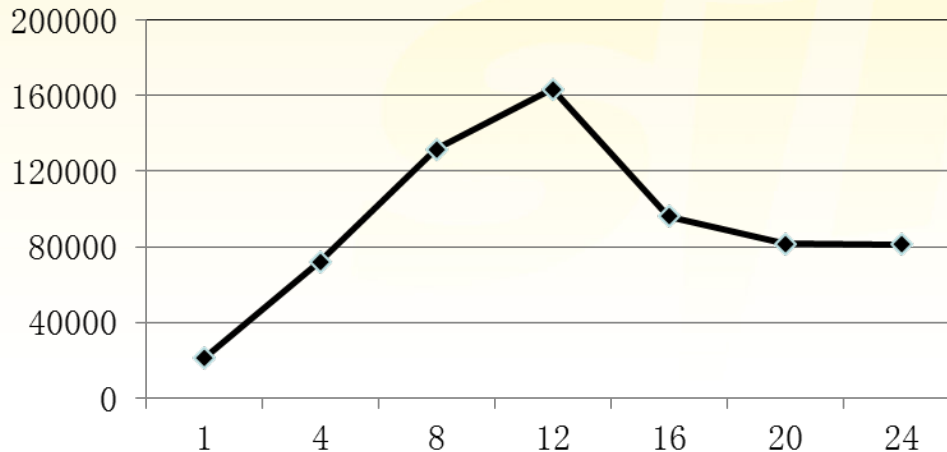

Synchronization Overhead

```
Samples: 786K of event 'cycles', Event count (approx.): 380344251534
77.60% [kernel] [k] _spin_lock
0.77% haproxy [.] process_session
0.73% [kernel] [k] _spin_lock_irqsave
0.59% [kernel] [k] kmem_cache_free
0.37% [kernel] [k] d_alloc
0.37% [ixgbe] [k] ixgbe_poll
0.30% [kernel] [k] _spin_lock_bh
0.30% [kernel] [k] kfree
0.28% [kernel] [k] __inet_lookup_established
0.28% [kernel] [k] tcp_transmit_skb
0.26% [kernel] [k] tcp_ack
```

Almost 80% CPU time is spent on locking.

Scalability Problem

HTTP CPS (Connection Per Second) throughput with different number of CPU cores.



Scalability is KEY to multicore system capacity.

Dilemma

How to update single machine capacity?

- Spend more for more CPU cores
- Capacity dose not improve but get worse
- Just awkward



What to do

- Hardware Assistance
Limited effects. (NIC offload feature)
- Data Plane Mode
You need to implement your own TCP/IP stack. (DPDK)
Lacks of kernel generality and full features.
Other production limits.
- Change Linux kernel
Keep improving but not enough. (Linux upstream)
Need to modify application code. (Megapipe OSDI)

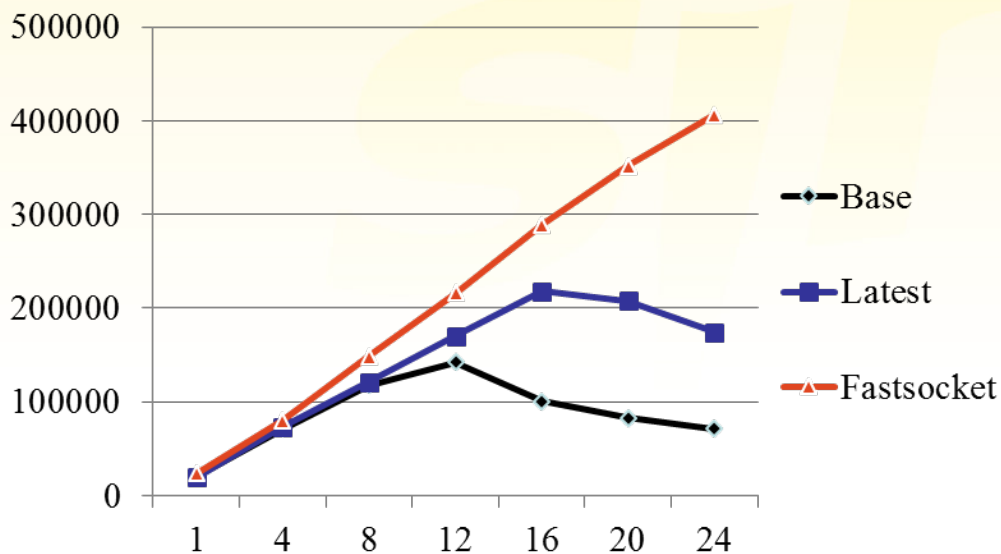
Fastsocket Scalability

```
Samples: 397K of event 'cycles', Event count (approx.): 216655154494
4.11% haproxy          [.] process_session
3.82% [kernel]         [k] _spin_lock
2.68% [kernel]         [k] kmem_cache_free
2.40% [fastsocket]     [k] 0x00000000000000244
1.71% [kernel]         [k] __inet_lookup_established
1.47% [ixgbe]          [k] ixgbe_poll
1.31% [kernel]         [k] dst_release
1.30% [kernel]         [k] ip_route_input
1.29% [kernel]         [k] _spin_lock_bh
1.23% haproxy          [.] eb_walk_down
1.20% [kernel]         [k] tcp_transmit_skb
```

Great! CPU is doing more for haproxy.

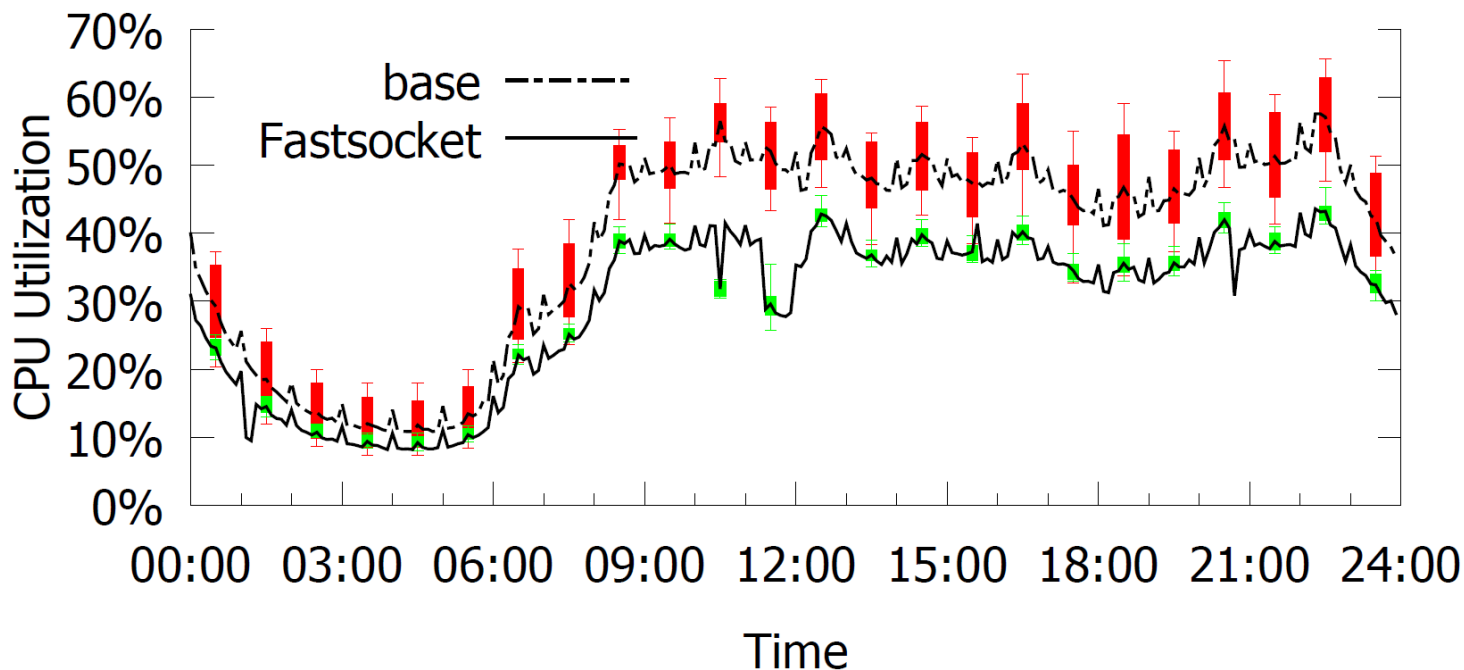
Fastsocket Scalability

Scalability is key for multicore performance.



Fastsocket: 5.8X Base and 2.3X Latest.

Production System Evaluation



Two 8-core HAProxy (HTTP proxy) servers handling same amount of traffic, with and without Fastsocket.

Kernel Bottleneck

- Non Local Process of Connections
- Global TCP Control Block (TCB) Management
- Synchronization Overhead from VFS

Non Local Process of Connections

A given connection is processed in two phases:

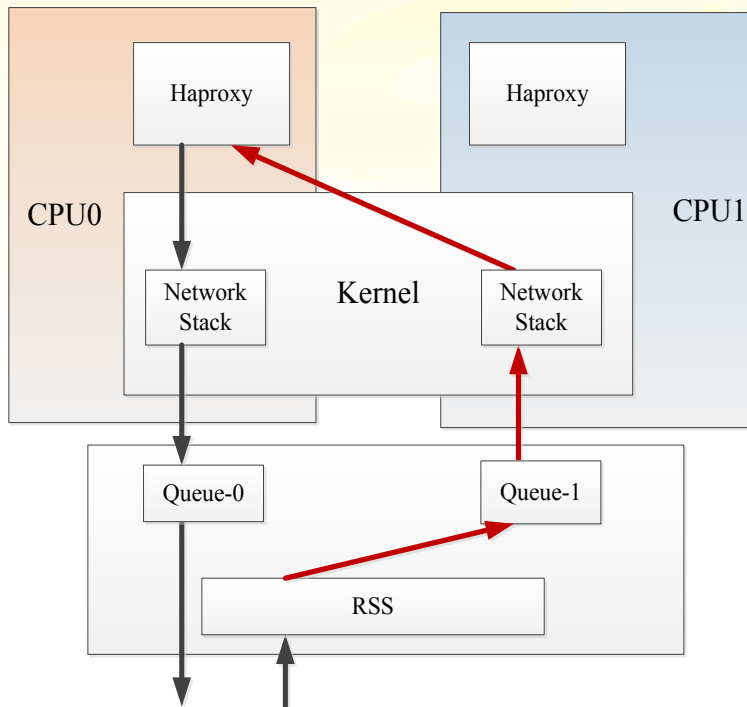
- Net-RX SoftIRQ in interrupt context
- Application and System Call in process context.

Two phases are often handles by different CPU cores:

- Introduces lock contentions.
- Causes CPU cache bouncing.

Non Local Process of Connections

Scenario that server actively connects out:



Global TCB Management

TCB is represented as socket in Linux kernel:

- Listen Socket:

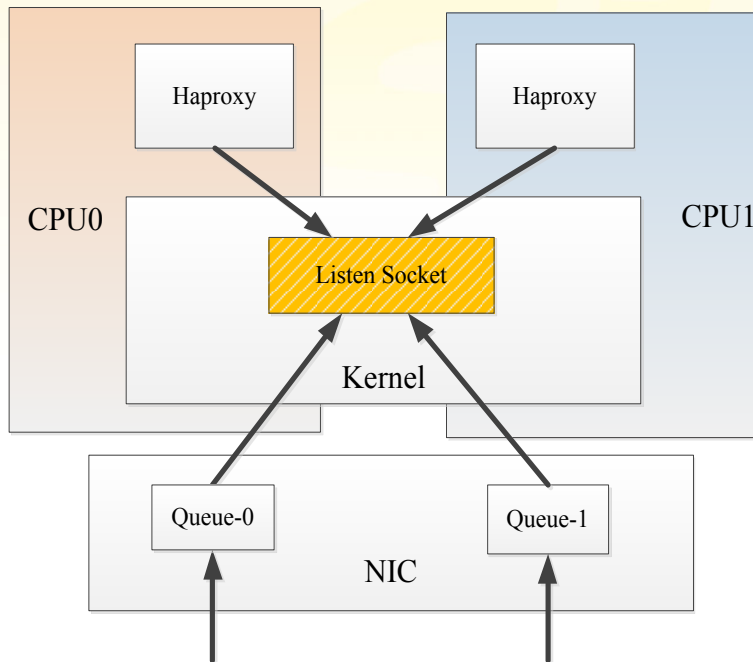
- A single listen socket is used for connection setup

- Established Socket:

- A global hash table is used for connection management

Global TCB Management

Single listen socket HOTSPOT:



VFS Overhead

- Socket is abstracted under VFS
- Intensive synchronization for *Inode* and *Dentry* in VFS
- These overhead are inherited by socket

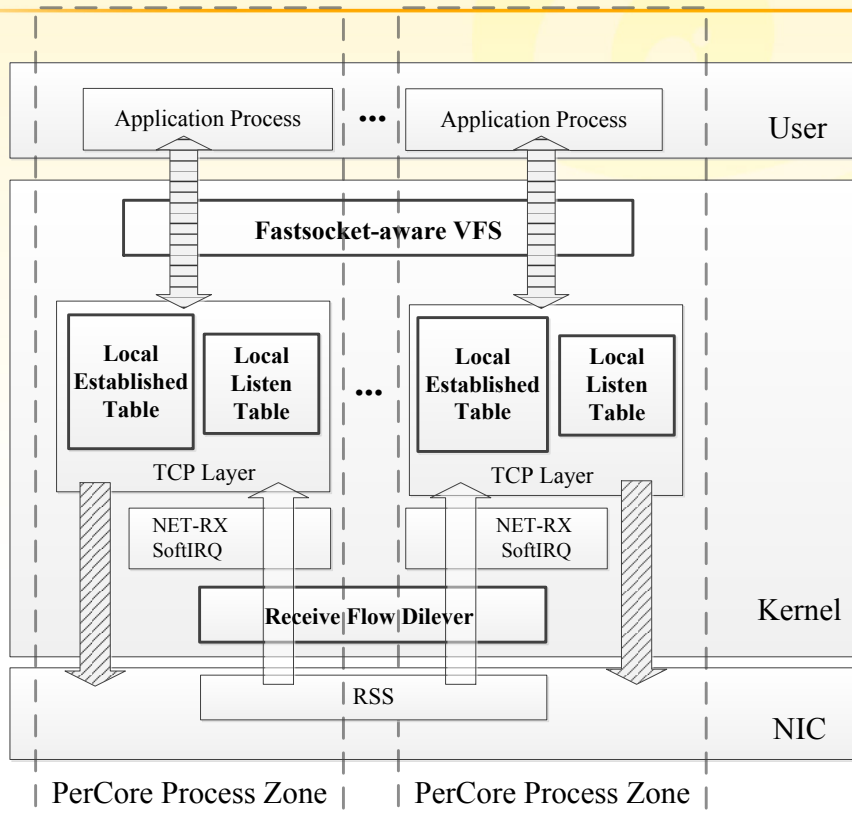
Methodology

- Resources Partition
- Local Processing

Design Component

- Receive Flow Deliver
- Local Listen Table & Local Established Table
- Fastsocket-aware VFS

Fastsocket Architecture



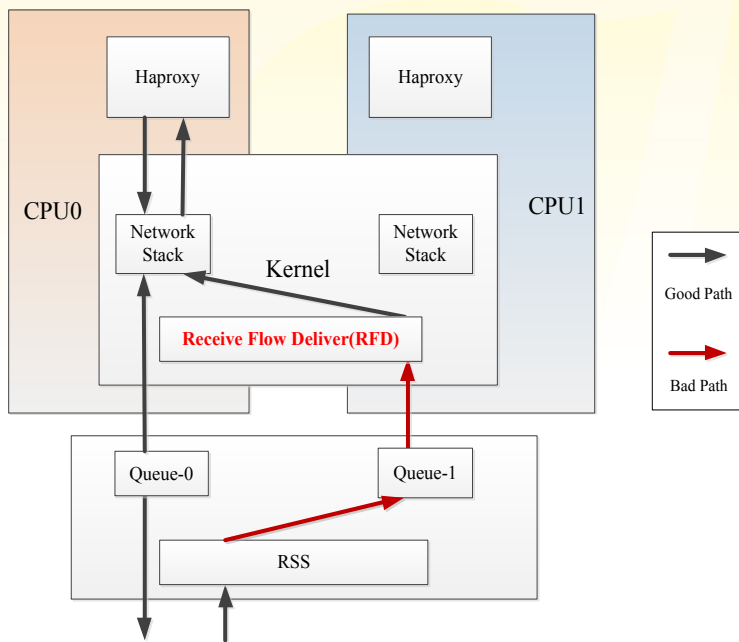
Outgoing packets to NIC



Incoming packets from NIC

Receive Flow Deliver (RFD)

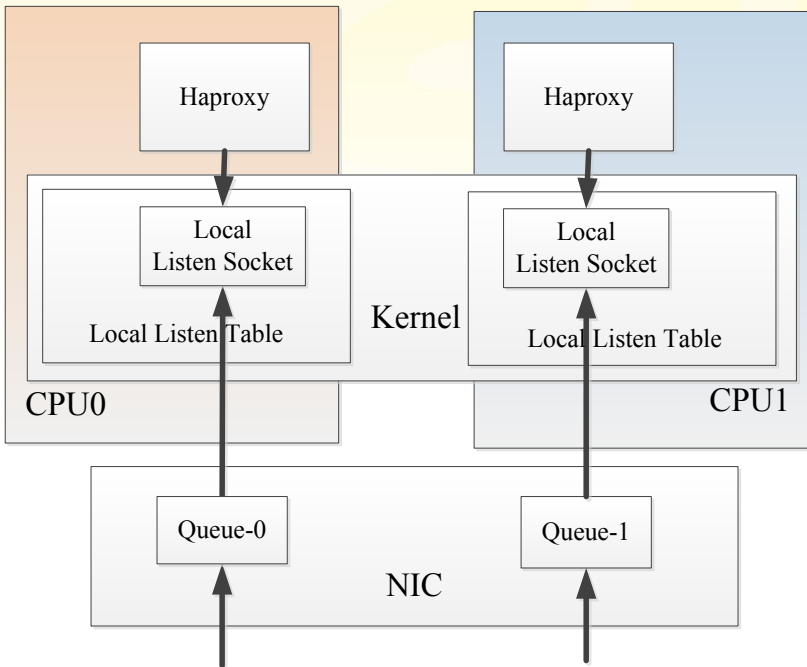
RFD delivers packets to the CPU core where application will further process them.



RFD can leverage advanced NIC features (Flow-Director)

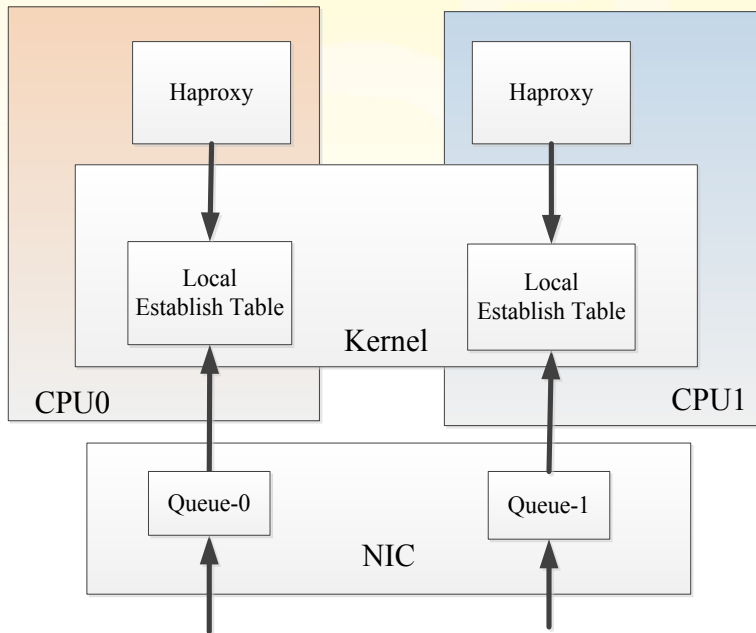
Local Listen Table

Clone the listen socket for each CPU core in LOCAL table.



Local Established Table

Established sockets are managed in LOCAL table.

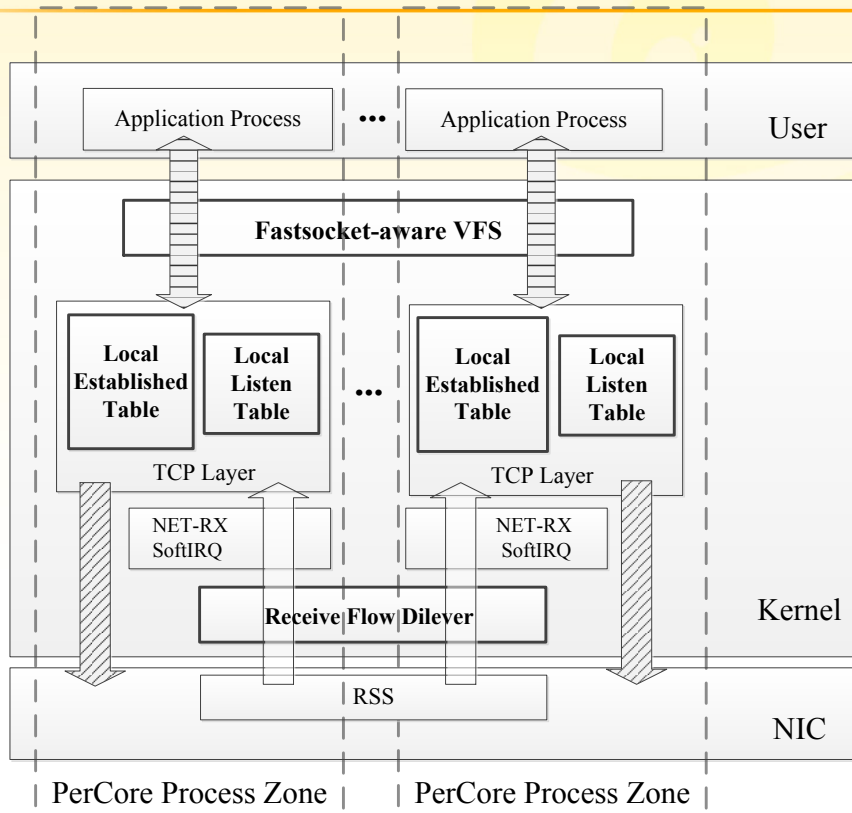


Fastsocket-aware VFS

Provide a FAST path for socket in VFS:

- *Inode* and *Dentry* are useless for socket
- Bypass the unnecessary lock-intensive routines
- Retain enough to be compatible

Fastsocket Architecture



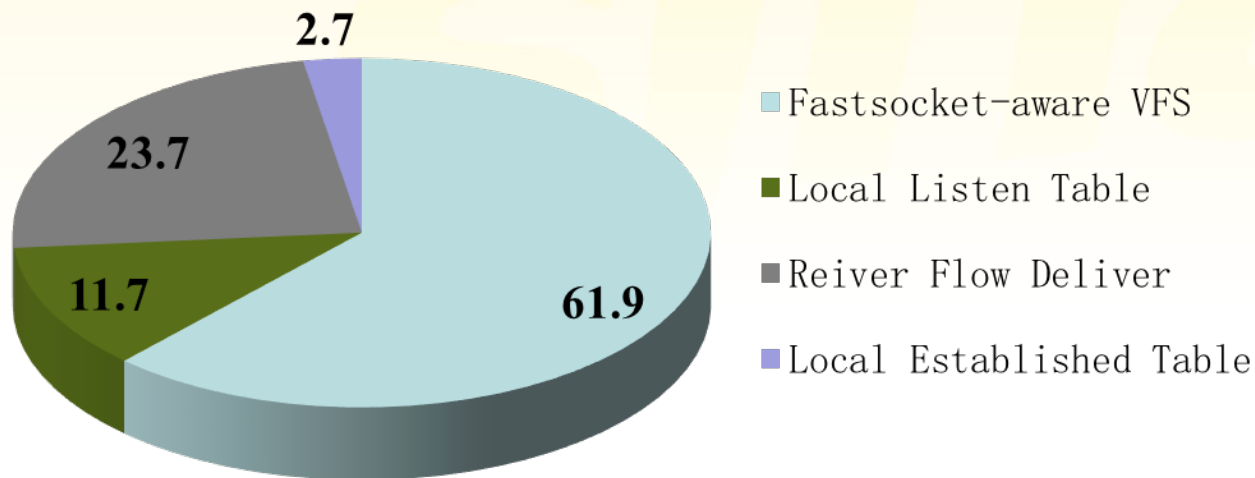
Outgoing packets to NIC



Incoming packets from NIC

Optimization Effects

Optimization effects percentage



Intel Hyper-Threading

Further boot performance 20% with Intel HT.

| | |
|---------------|----------------|
| | E5-2697-v2 |
| Fastsocket | 406632 |
| Fastsocket-HT | 476331(117.2%) |

Fastsocket-HT:476.3k cps, 3.87m pps, 3.1G bps (short connection)

Fastsocket

- Scalability Performance



- Single Core Performance
- Production System Feasibility
- Future Work

Methodology

- Network Specialization
- Cross-Layer Design

Network Specialization

General service provided inside Linux kernel:

- Slab
- Epoll
- VFS
- Timer* etc.

Customize these general services for Network

Fastsocket Skb-Pool

- Percore skb pool
- Combine skb header and data
- Local Pool and Recycle Pool (Flow-Director)

Fastsocket Fast-Epoll

Motivation:

- Epoll entries are managed in RB-Tree
- Search the RB-Tree each time when calling *epoll_ctl()*
- Memory access is a killer for performance

Solution:

- TCP Connection rarely shared across processes
- Cache Epoll entry in *file* structure to save the search

Cross-Layer Optimization

- What is cross-layer optimization?
- Overall optimization with Cross-Layer design
 - Direct-TCP
 - Receive-CPU-Selection

Fastsocket Direct-TCP

- Record input route information in TCP socket
- Lookup socket directly before network stack
- Read input route information from socket
- Mark the packet as routed

Fastsocket Receive-CPU-Selection

Similar to Google RFS

- Application marks current CPU id in the socket
- Lookup socket directly before network stack
- Read CPU id from socket and deliver accordingly

Lighter, more accurate and thus faster than Google RFS

Redis Benchmark

Testing Environment:

- Redis: Key-value cache and store
- CPU: Intel E5 2640 v2 (6 core) * 2
- NIC: Intel X520

Configuration:

- Persist TCP connections
- 8 redis instances serving on difference ports
- Only 8 CPU cores are used

Redis Benchmark


Disable Flow-Director:

20% throughput increase

Enable Flow-Director:

45% throughput increase

Fastsocket

- Scalability Performance
- Single Core Performance
-  • Production System Feasibility
- Future Work

Compatibility & Generality

- Full BSD-Socket API compatible
- Full kernel network feature support
- Require no change of application code
- Nginx, HAProxy, Lighttpd, Redis, Memcached, etc.

Deployment

- Install RPM (Kernel, *fastsocket.ko* and *libfsocket.so*)
- Load *fastsocket.ko*
- Start application with *PRELOAD libfsocket.so*

LD_PRELOAD=./libfsocket.so haproxy

Maintainability

- Flexible Control
 - Selectively enable Fastsocket to certain applications (nginx)
 - Compatible with regular socket (ssh)
- Quick Rollback
 - Restart the application without *libfsocket.so*
- Easy Updating
 - Most of codes are in *fastsocket.ko*
 - Updating *fastsocket.ko* and *libfsocket.so* is enough

SINA Deployment

- Adopted in HTTP load balance service (HAProxy)
- Deployed in half of the major IDCs
- Stable running for 8 months
- Fastsocket will update all HAProxy by the end of year

Fastsocket

- Scalability Performance
- Single Core Performance
- Production System Feasibility



- Future Work

Future Work

- Make it faster

- Improving Interrupt Mechanism
- System-Call Batching
- Zero Copy
- Further Customization and Cross-Layer Optimization

- Linux Mainstream

Open Source

<https://github.com/fastos/fastsocket>

Recruitment

Join us !

xiaodong2@staff.sina.com

Fastsocket

Thank you!

Q & A