LC-trie
seminar 2006 KTH

Robert Olsson

Experiments, Development and Experiences
with FIB lookup and LC-trie

Background

Been involved testing and development and productions for many years

Contributions to Linux network stack and routing daemons etc.

NAPI, pktgen, device drivers tulip, e1000
etc

HW testing and hi-end production systems.

What we hear/got

dst cache overflow reports
- RCU related
- mistuned, misunderstod etc.

fib_lookup complaints
- what to expect
- BSD comparisons. Radix-tree
- ToS/semantic questionable
- current fib_hash considered bad

Route hash vs fib_lookup

First packet in a **row** takes a longer path (slow path) goes through careful sanity inspection, route lookup (fib_lookup) and RP-checks etc. On success a dst_entry is created. This holds intelligence how packet should traverse the kernel stack.

Packet inside Linux kernel are stored in socket buffers skb's. A skb has a pointer to dst_entry. dst_entries are kept in the dst (route) hash.

Following packets can be lookup from dst hash and avoid slow path with fib_lookups etc.

Route hash vs fib_lookup

Some stateful information is kept
in the dst cache

dst cache needs garbage collection

tuning for work load is often needed

Route hash vs fib_lookup

Example from rtstat from Uppsala Universitet core router 2006 19/4 11:30

```
size
95992

IN: hit      tot    mc no_rt bcast madst masrc
444119    23836     4     2     0     0     0

OUT: hit      tot    mc
    3903        5     0

GC: tot ignored goal_miss ovrf
  13896   13894         0    0

HASH: in_search     out_search
         167418           4337
```

Route hash vs fib_lookup

Example from rtstat from Uppsala Universitet core router 2006 19/4 11:30

```
size
95992

IN: hit      tot    mc no_rt bcast madst masrc
 444119   23836     4     2     0     0     0

hit = warm cache hit
tot = route lookup/fib_lookup

Relation is very interesting. It helps monitoring
of rDoS, length of flows etc.

This minute a packet load of 444k + 24k = 468 kpps
```

Route hash vs fib_lookup

Example from rtstat from Uppsala Universitet core router 2006 19/4 11:30

```
OUT: hit      tot     mc
     3903       5      0
```

Output from router. Typically router daemon bgpd/ospf or ssh connection

Route hash vs fib_lookup

Example from rtstat from Uppsala Universitet core router 2006 19/4 11:30

```
GC: tot ignored goal_miss ovrf
  13896   13894          0    0
```

Garbage collection of route hash entries a very crucial process. Needed on own semiar.

Route hash vs fib_lookup

Example from rtstat from Uppsala Universitet core router 2006 19/4 11:30

Spinning in the hash (linear search) kills perfromance as it create cache misses.

in_search = spinning caused by incoming pkts.

```
HASH: in_search     out_search
         167418           4337
```

Policy routing

Linux can have 256 routing tables by default and do lookup via tos, src fwmark etc

```
ip rule
0:      from all lookup local
32766:  from all lookup main
32767:  from all lookup default

# Create a new table
ip rule add from 130.238.122.160/27 table 200
# Add deault route to table
ip route add 0/0 dev wlan0 table 200

 ip rule
0:      from all lookup local
32765:  from 130.238.122.160/27 lookup 200
32766:  from all lookup main
32767:  from all lookup default
```

Policy routing

Linux can have 256 routing tables by default and do lookup via tos, src fwmark etc

Route cache hides lookup cost from when many tables are used tables. Can be improved...

Also RCU lock recently added for rule lookup.

Policy routing can be disabled in kernel config if you don't need it.

fib_hash (linux default)

Fast - Yes
General purpose
Very integrated

33 linked hash tables

33->0 (default route)

Tables are linked i Linux
32->24->0 Normal for host

tables are resized.

Other activities informal linux agenda

Ericsson is willing to open patent for Linux
Jamal have the contacs via Ericsson Montreal

DaveM has discussions with Washington university about who is willing to grant another patent for use with Linux

Discussed LC-trie with Alexey Kuznetsov.

LC-trie investigations. Got GPL from authors.

LC-trie

In theory variable key length, 32, 128 bits etc

Algo for dynamic trie written in Java. Memory leak and stack handling were problems.

prefix matching based on fib_sematic_match

Cisco CEF has fixed 256 childs 8-8-8-8 or 16-8-8 (GSR)
LC-trie is child size is dynamic 2-12 bits seen

Is insert/delete performance good enough? Switch tables after each insert/delete.

Infrastructure for test & development

Handy well used flexible el cheapo lab

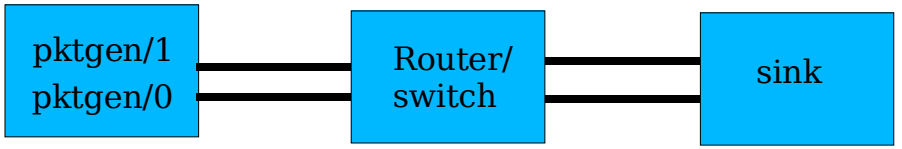
Preroute patches with Jamal at OLS 2004
to disable route hash and just do fib_lookup

pktgen DoS, scripts w. routing table

Long-time Linux API work to prepare to
plugin new algos most from DaveM.

So much research
Still so little usable for Linux

Lab setup



Forwarding in parallel setup

Can use multiple CPU's on sender(s) and on multiple CPU's on router.

rDoS or single flow(s)

All measurements measured in forwarding context with 64 byte packets,

First take

Started with dyntree. Java. Jens had luckily ported some Java programs before. 1:1 port

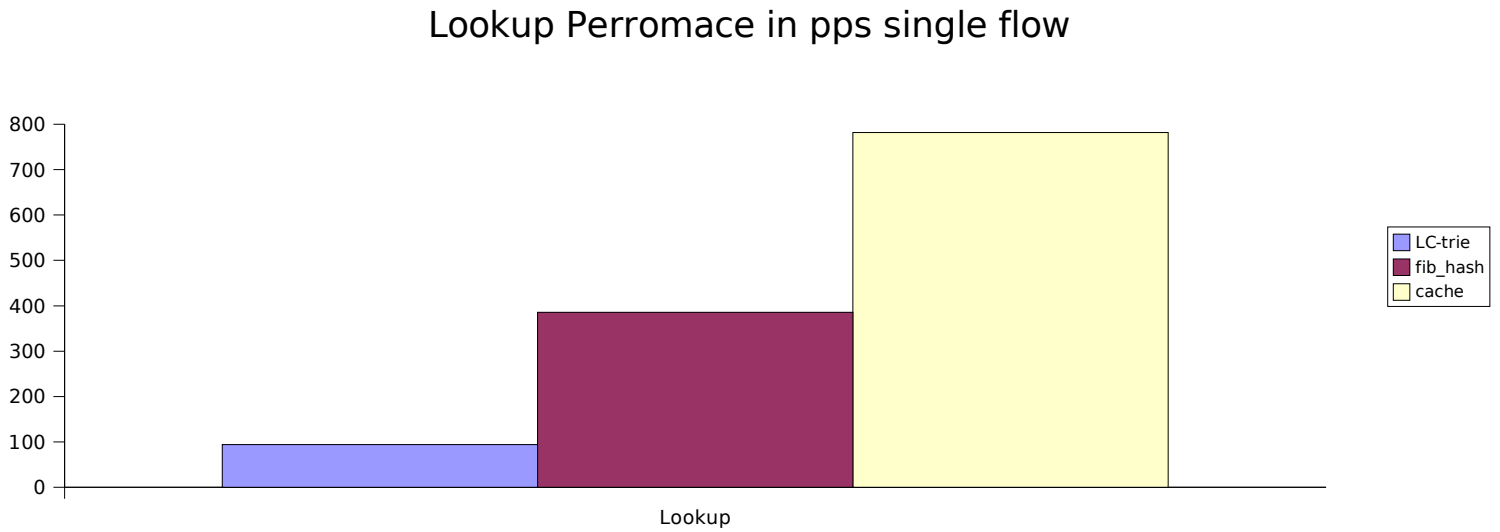
Memory leaks to kill us

Stack handling to almost kill us

Not so easy to interface the Linux API

Performance is hopeless bad

First shot/shock



White route cache
Red fib_hash
Blue fib_trie

Profiles gave no clue...

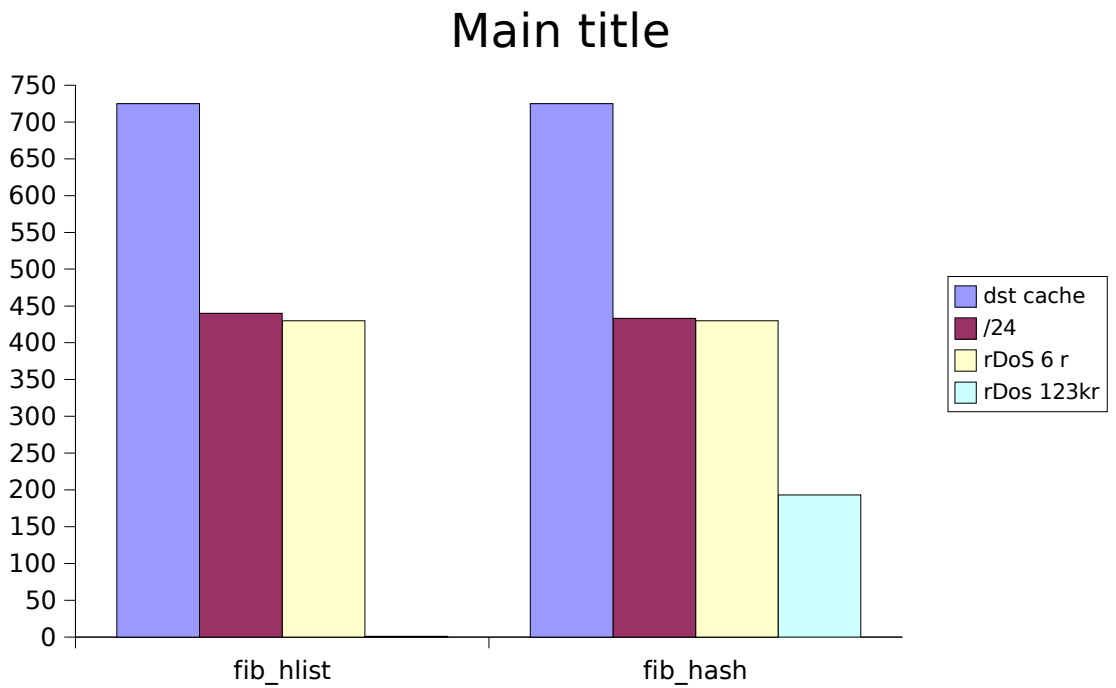
Doubts after hard fight
Gave up LC-trie for now

fib_hlist
fib_hash2
fib_trie
classifier lookup?
unified lookup?

fib_hlist
simplest routing algo
in linux ever

Idea was a tutorial
KISS
hlist with semantic_match
Very fast with small tables
For embedded system etc?

fib_hlist performance



Note!
Zero for fib_hlist :) Still decent many apps.

NEXT: fib_hash2 huge hash with prefix expansion

Vargese inspired, use what got

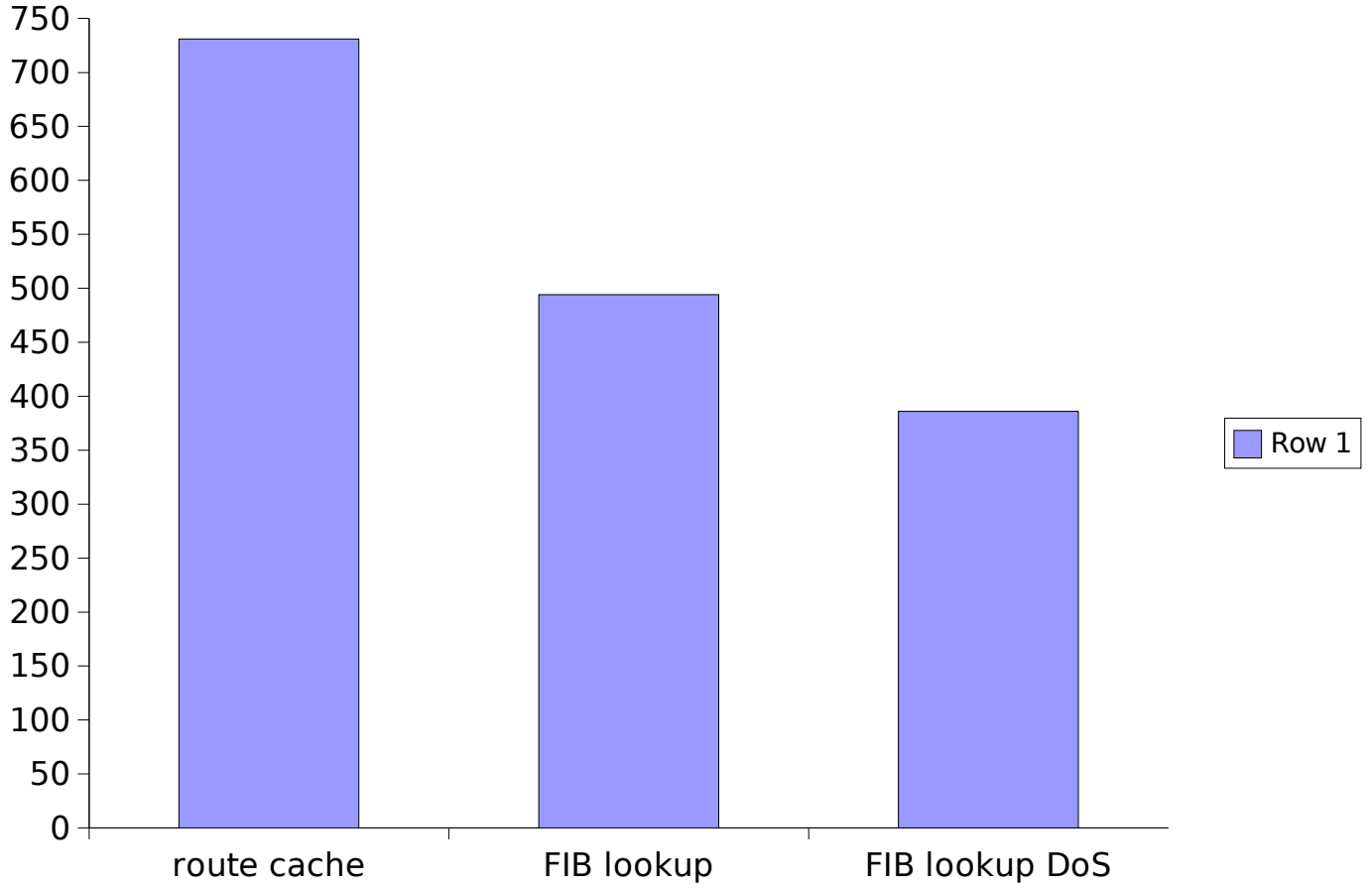
2^24 hash lookup w. sorted hlist
Makes /24 entries of plens 1-23
/0 special case. Huge...
TABLE_LOCAL with a few entries

Idea was to test performance
with the fastest algo we could think of.

Not for embedded system etc? :-)

Reduced it became fib_hlist

fib_hash2/route cache compare



fib_trie implementation redesign

Needed lot of energi

- Redesign of data structures
- Remove all Java inheritage
- Rewrite/remove stack use
- Introduce parent-pointer
- Use Linux list functions etc
- Rework of lookup. Backtracking. Hasse Liss did clever lookahead.

fib_trie
beginning to see the light

How do we do a fair comparison?

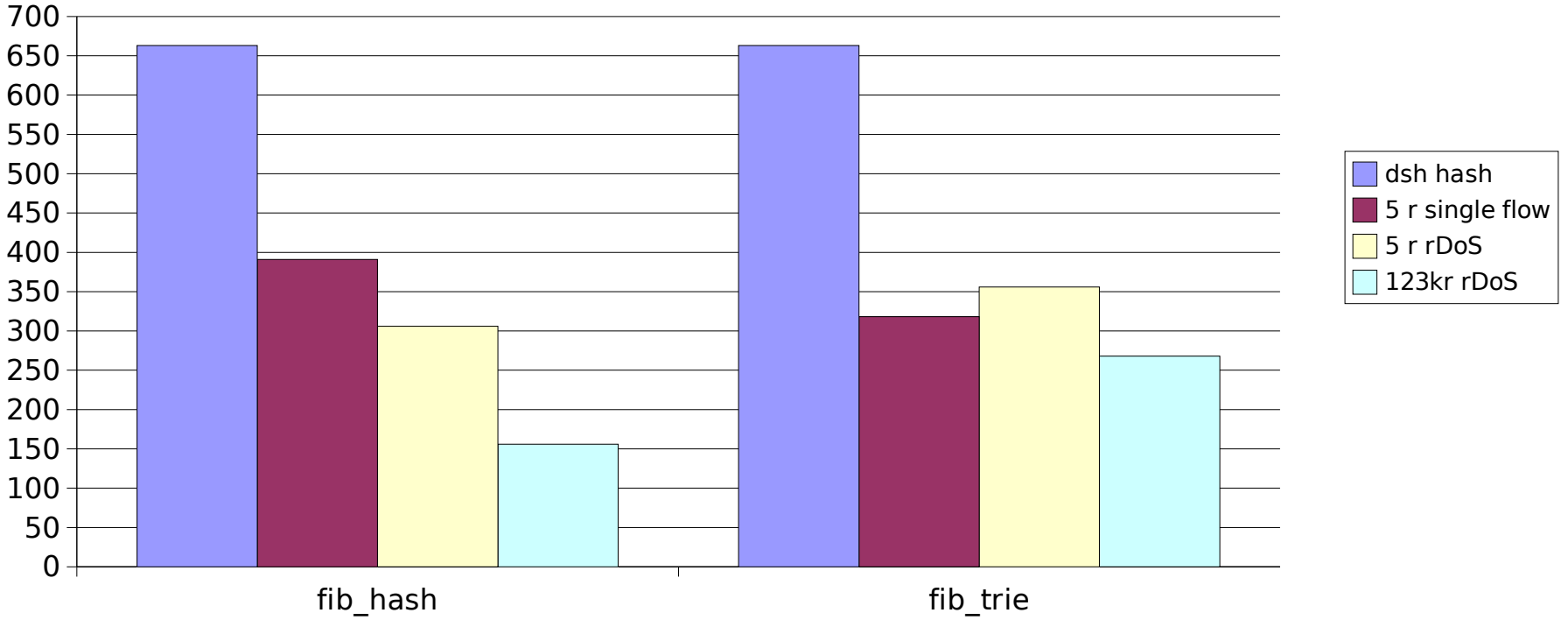
Every paper has the fastest algo :)

Scientific inflation or marketing?

fib_trie performance comparison

forwarding kpps

Linux 2.6.16 1 CPU used(SMP) Opteron 1.6 GHz e1000



Preroute pathes to disable route hash

LOCAL/MAIN tables

fib_lookup() in ip_fib.h

Always looks up LOCAL table before MAIN

Extra lookup costs performance when not to localhost.

We discussed this with Alexey...

LOCAL/MAIN tables

Aver depth: 4.48
Max depth: 6
Leaves: 25
Internal nodes: 18

Aver depth: 3.22
Max depth: 7
Leaves: 158936
Internal nodes: 39440

fib_trie verfication

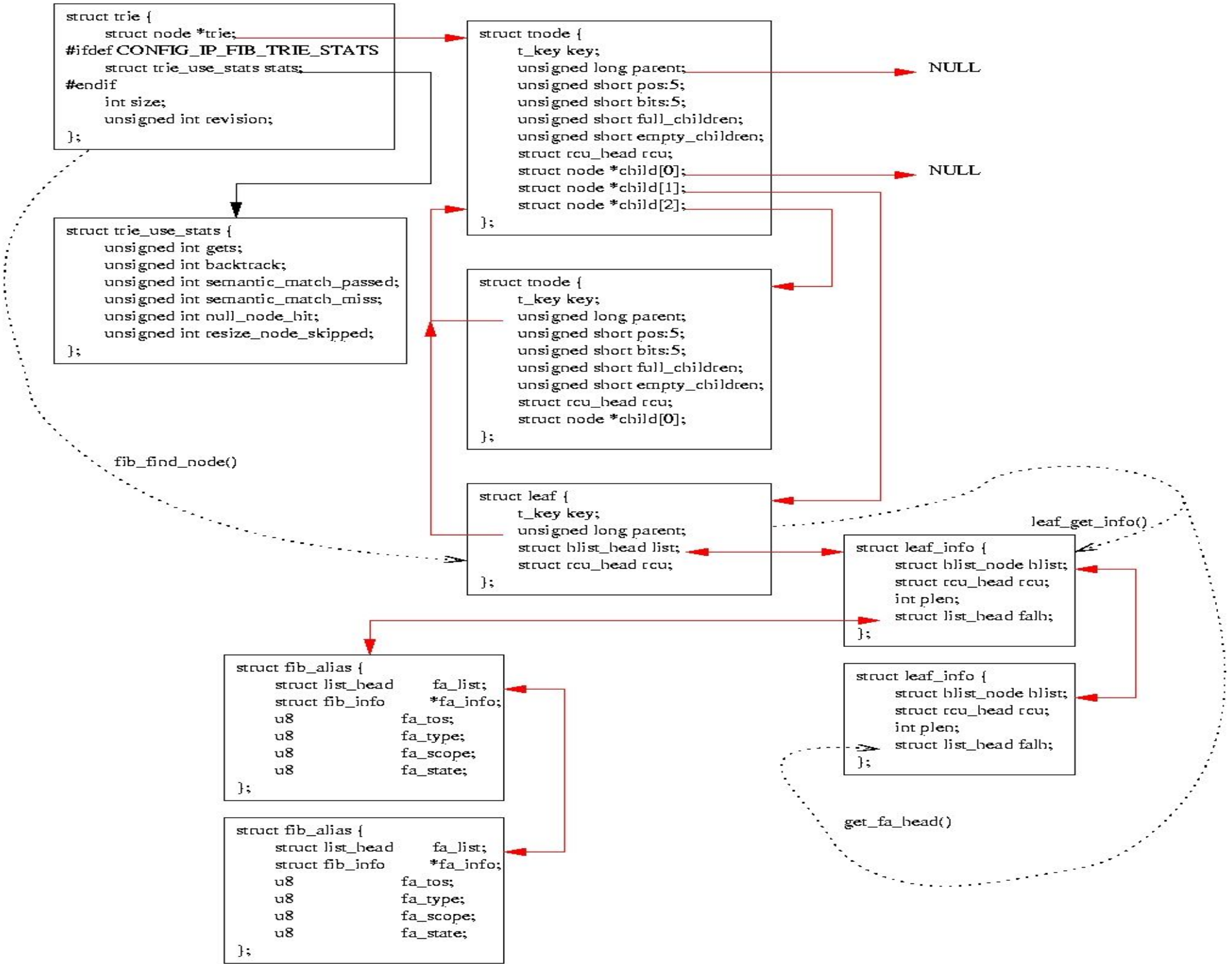
C-program to compare LPM (longest prefix match) with loaded table.

Linux kernel inclusion demanded verification technique

New netlink call to do fib_lookup

fib_lookup can done for userland app. Normally only lookups are done kernel. (Softirq) This way the “full” process insert/lookup/delete can be tested and verified.

But forgot to test the fib_semantic_match fully. :)


struct trie {
struct node *trie;
#ifdef CONFIG_IP_FIB_TRIE_STATS
struct trie_use_stats stats;
#endif
int size;
unsigned int revision;
};
struct trie_use_stats {
unsigned int gets;
unsigned int backtrack;
unsigned int semantic_match_passed;
unsigned int semantic_match_miss;
unsigned int null_node_hit;
unsigned int resize_node_skipped;
};
struct tnode {
t_key key;
unsigned long parent;
unsigned short pos:5;
unsigned short bits:5;
unsigned short full_children;
unsigned short empty_children;
struct rcu_head rcu;
struct node *child[0];
struct node *child[1];
struct node *child[2];
};
NULL
NULL
struct tnode {
t_key key;
unsigned long parent;
unsigned short pos:5;
unsigned short bits:5;
unsigned short full_children;
unsigned short empty_children;
struct rcu_head rcu;
struct node *child[0];
};
fib_find_node()
struct leaf {
t_key key;
unsigned long parent;
struct hlist_head list;
struct rcu_head rcu;
};
leaf_get_info()
struct leaf_info {
struct hlist_node hlist;
struct rcu_head rcu;
int plen;
struct list_head falh;
};
struct leaf_info {
struct hlist_node hlist;
struct rcu_head rcu;
int plen;
struct list_head falh;
};
struct fib_alias {
struct list_head fa_list;
struct fib_info *fa_info;
u8 fa_tos;
u8 fa_type;
u8 fa_scope;
u8 fa_state;
};
struct fib_alias {
struct list_head fa_list;
struct fib_info *fa_info;
u8 fa_tos;
u8 fa_type;
u8 fa_scope;
u8 fa_state;
};
get_fa_head()

Locking issues

Started with RW lock as fib_hash was “planned” implemenation for RCU

Read Copy Update. IBM research donated by IBM. Paul McKenny author

RCU port by Stephen Hemminger/Paul M. and me.

Lot's of disussions... Maybe contributed to RCU too.

Locking issues

For read-mosty locks

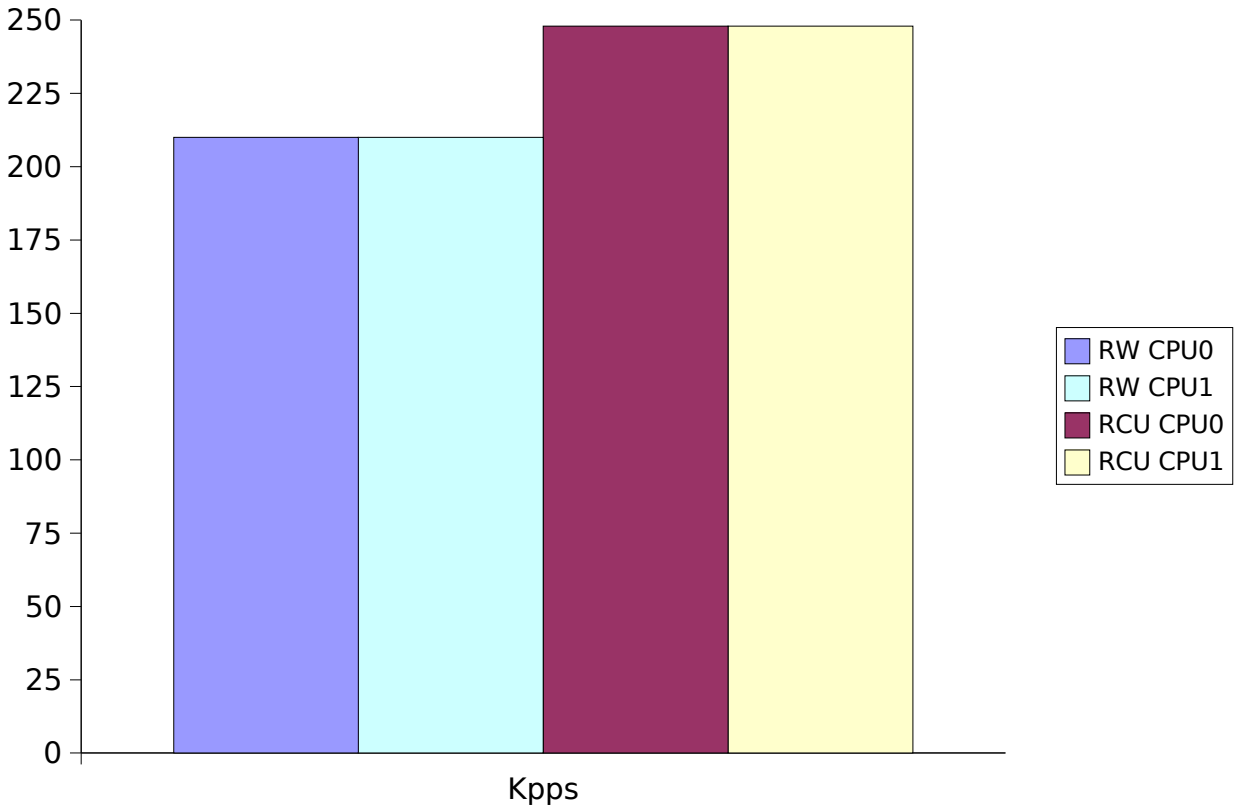
RCU makes pointer updates atomic
Needs deferred deletion.

Read-side without locks
Writer functions need serialzation

Memory ordering different on CPU arch
Lot's of macros.

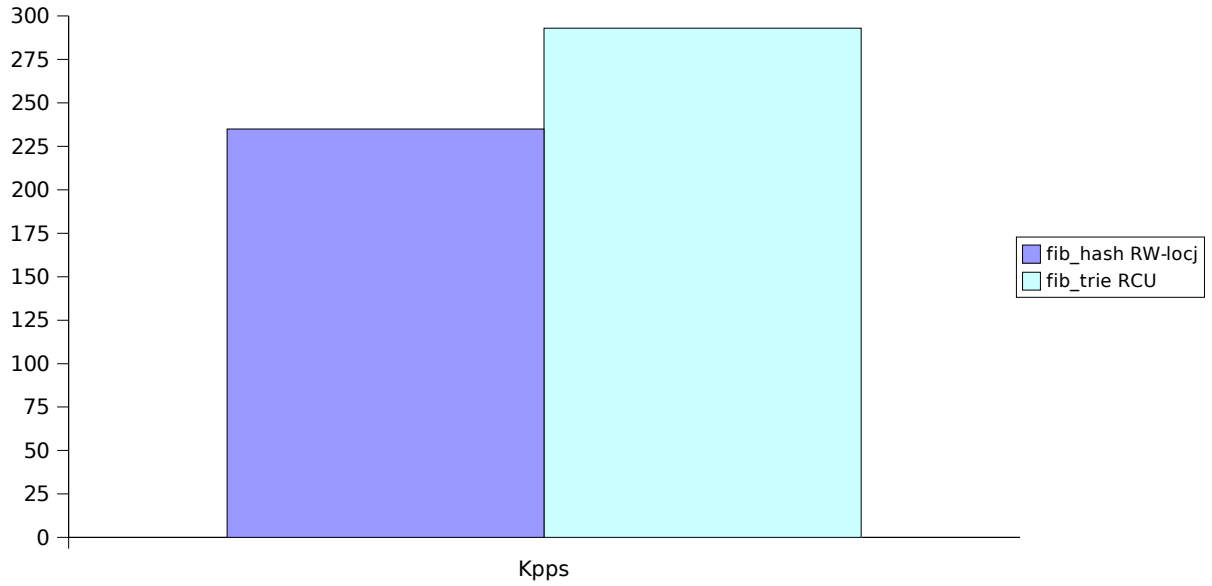
Performance RCU/RW-lock

Forwarding rDoS on DUAL 1.6 Ghz Opteron
rDoS on same /8 prefix.



Performance
fib_hash vs fib_trie

Forwarding rDoS on DUAL 1.6 Ghz Opteron
Two CPU's aggregated. Most output on one NIC



Test & Current use

Pktgen in Uppsala U. lab

1) rDoS at 2x680 kpps -- lookup
2) rDoS -- insert/replace/delete
3) rDoS dump

Linux kernel 2.6.14 (RCU variant)

Deploy at SLU core routers
Now in UU core routes

IBM/Austin working on a test setup
with several CPU's.

Other applications

Ipv6
Stateful lookup
?
Integration with filters etc

Current work
LC-trie for unified lookup with 64/128/256 etc keys

```
Example above from linux kernel experiments with rDoS
Key is 128 bit src/dst/dport/sport/proto/ifce

trie:
        Aver depth:     1.30
        Max depth:      4
        Leaves:         99190
        Internal nodes: 14356
          1: 13154  2: 1196  3: 5  18: 1
        Pointers: 293276
Null ptrs: 179731
Total size: 5638  kB
```

Aver depth of 1.3 gives very fast lookup.