Linux Networking Dietary Restrictions

David S. Miller (Red Hat Inc.)

Small is Beautiful

Critical Linux Networking Structures

```
struct sk_buff          216 bytes

struct dst_entry        160 bytes

     struct rtable      216 bytes

     struct rt6_info    384 bytes (!)

struct sock_common      136 bytes

     struct sock        704 bytes (!!)

struct net_device       1920 bytes (!!!!!!)
```

Things Have Clearly Spun Out of Control

Why Does This Matter?

Performance, Performance, Performance…

Smaller structures, less cache misses

Size is also about complexity, bigger stuff is usually more complicated and harder to maintain

Memory is a resource

Techniques For Fixing This

- Un-commoning
- State Compression
- Unused Pointer Bit Storage
- Unionize State Specific Members
- Lookup Keys Instead of Object Pointers

Un-Commoning

Member of common base class

Only used by some subclasses

→ Push it down into subclasses

struct dst_entry (before)

```
struct net_device       *dev;                              /* 0x00 */
struct rcu_head         rcu_head;                          /* 0x08 */
struct dst_entry        *child;                            /* 0x18 */
struct dst_ops          *ops;                              /* 0x20 */
unsigned long           _metrics, expires;                 /* 0x28 */
struct dst_entry        *path, *from;                      /* 0x38 */
struct xfrm_state       *xfrm;                             /* 0x48 */
int                     (*input), (*output);               /* 0x50 */
unsigned short          flags;                             /* 0x60 */
short                   error, obsolete;                   /* 0x62 */
unsigned short          header_len, trailer_len, __pad3;   /* 0x66 */
__u32                   tclassid; unsigned long __pad2[2];/* 0x6c */
atomic_t                __refcnt; …                        /* 0x80 */
```

struct dst_entry (before, part 2)

```
int                     __use;                  /* 0x84 */
unsigned long           lastuse;                /* 0x88 */
struct lwtunnel_state   *lwtstate;              /* 0x90 */
union {                                         /* 0x98 */
     struct dst_entry         *next;
     struct rtable __rcu      *rt_next;
     struct rt6_info          *rt6_next;
     struct dn_route __rcu    *dn_next;
};

Total: 160 bytes
```

struct dst_entry

- dst_entry->next
  - Not used by ipv4 routes (struct rtable)
  - Push down into subclasses
- dst_entry->child
  - Only used by IPSEC routes
  - Push down into xfrm_dst

struct dst_entry (cont.)

- dst_entry->from
  - Only used by rt6_info
  - Push it down
- dst_entry->path
  - Only used by xfrm_dst
  - Push it down

dst_entry Pains

dst->refcnt alignment

```
BUILD_BUG_ON(offsetof(struct dst_entry, __refcnt) & 63);
```

Existing padding on 64-bit to achieve this:

```
long __pad_to_align_refcnt[2];
```

Increases to ‘5’ when child/from/path removed!

Uncommon’d dst_entry (152 bytes)

```
struct net_device       *dev;                       /* 0x00 */
struct rcu_head         rcu_head;                   /* 0x08 */
struct dst_ops          *ops;                       /* 0x18 */
unsigned long           _metrics, expires;          /* 0x20 */
struct xfrm_state       *xfrm;                      /* 0x30 */
int                     (*input), (*output);        /* 0x38 */
unsigned short          flags;                      /* 0x48 */
short                   error, obsolete;            /* 0x4a */
unsigned short          header_len, trailer_len, pad;   /* 0x4e, 0x50, 0x52 */
__u32                   tclassid;                   /* 0x54 */
long                    __pad_to_align_refcnt[5];   /* 0x58 */
atomic_t                __refcnt;                   /* 0x80 */
int                     __use;                      /* 0x84 */
unsigned long           lastuse;                    /* 0x88 */
struct lwtunnel_state   *lwstate;                   /* 0x90 */
```

Eliminate Useless Padding

40 bytes wasted

Let’s put write-heavy cacheline at offset 64

Consolidate write-heavy members with lightly accessed read-only members

New dst_entry layout (112 bytes)

```
struct net_device       *dev;                   /* 0x00 */
struct dst_ops          *ops;                   /* 0x08 */
unsigned long           _metrics, expires;      /* 0x10 */
struct xfrm_state       *xfrm;                  /* 0x20 */
int                     (*input),(*output);     /* 0x28 */
unsigned short          flags;                  /* 0x38 */
short                   obsolete;               /* 0x3a */
unsigned short          header_len, trailer_len;/* 0x3c */
atomic_t                refcnt;                 /* 0x40 */
int                     __use;                  /* 0x44 */
unsigned long           lastuse;                /* 0x48 */
struct lwtunnel_state   *state;                 /* 0x50 */
struct rcu_head         rcu_head;               /* 0x58 */
short                   error, __pad;           /* 0x68 */
__u32                   tclassid;               /* 0x6c */
```

Final Sizes

```
struct dst_entry → 112 bytes (-48)
struct rtable →    168 bytes (-48)
struct rt6_info →  320 bytes (-64)
struct xfrm_dst →  448 bytes
```

Side Note: ‘pahole’

Real men compute structure offsets by hand.

For the rest of you there is ‘pahole’

Part of the ‘dwarves’ set of utilities.

Computes structure layout using debugging info.

Using ‘pahole’

Install ‘dwarves’ package.

Build kernel with debugging info. (CONFIG_DEBUG_INFO)

bash$ make net/core/dst.o

bash$ pahole -c 64 -C dst_entry --hex net/core/dst.o

State Compression

‘Bool’ is great but too expensive in critical data-structures.

Even ‘u8’ boolean values can be too large

→ use C bitfields or group into ‘flags’ (‘u8’, ‘u16’, ‘u32’, as needed)

Understand the value range for struct members:

→ a ‘u16’ or ‘u8’ may be more appropriate than ‘int’ or ‘u32’

Evaluate whether the value’s range is unnecessarily large (f.e. Low bits unused).

Unused Pointer Bits

Common paradigm is “pointer” plus some small piece of state.

For example, some boolean values.

Or, alternatively, a small type encoding.

Either way, every pointer in the kernel has 3 unused low bits for information storage.

Requires special encoding and helper accessor functions to make error free.

Pointer Bits Example

Metrics for “dst_entry”

Encoded in dst->_metrics which is “unsigned long”

This is a pointer plus two state bits:

→ DST_METRICS_READ_ONLY 0x01L

→ DST_METRICS_REFCOUNTED 0x02L

Allows sharing of in-route and ‘const’ metrics, plus reference counting of dynamic metrics.

Unionization of State Specific State

Compress structure when members are used at different times.

For example, state specific values or sub-class specific values.

Example (struct sock):

```
union {
        struct sk_buff *sk_send_head;
        struct rb_root tcp_rtx_queue;
};
```

Lookup Keys Instead of Pointers

If an object can be found cheaply based upon index, store the index instead of the object pointer.

As an example struct sk_buff stores the incoming device index instead of the netdev pointer.

Thank You

NIPA and Director Cho

Willem de Bruijn and Eric Dumazet

Anyone who has shrunk data structures

And all who will do so in the future

Linus Torvalds