Архитектура ядра OpenBSD
Сетевой стек

Белопухов Михаил
mikeb@openbsd.org

Sun Microsystems SPB

OpenKyiv 2008

Содержание

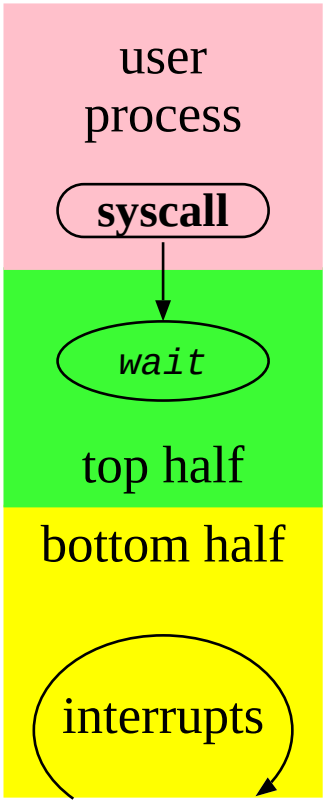
Ядро как часть ОС

- Работает в защищенном режиме
- Монолитно по структуре, есть возможность подгрузки модулей
- Управляет работой переферийных устройств
- Определяет интерфейс взаимодействия и управляет доступом пользовательских приложений к системным ресурсам и оборудованию

Организация ядра

- Архитектуро-зависимый функционал (Machine Dependent, MD): процедуры начальной загрузки, обработка прерываний и т.д.
- Архитектуро-независимый функционал (Machine Independent, MI): сетевой стек, файловые системы и т.д.

Архитектура ядра: Контексты ядра



Контекст прерывания

- Заснуть или заблокироваться нельзя
- Системным планировщиком не обрабатывается
- Используется стек ядра

Контекст процесса

- Может заблокироваться или заснуть в ожидании освобождения ресурса
- Используется стек процесса, пространство адресов ядра

Архитектура ядра: Вход в ядро

Аппаратное прерывание, hardware interrupt

- Поисходит асинхронно по отношению к выполняющемуся в данный момент процессу; не имеет к нему отношения
- Обрабатывается согласно таблице векторов прерываний
- Прерывает работу текущей задачи

Аппаратное исключение, hardware trap

- Поисходит синхронно по отношению к выполняющемуся в данный момент процессу в результате действий процесса

Программное прерывание, software trap/interrupt

- Может быть использовано процессом для перехода на превилегированый уровень исполнения (системный вызов)
- Используется системой для реализации отложенных прерываний

Архитектура ядра: Обработка аппаратных прерываний

Маскируемые прерывания, IRQ

- Имеют ассоциированый бит в IMR
- В BSD используют схему приоритетов Interrupt Priority Level, IPL

Немаскируемые прерывания, NMI

- Невозможно проигнорировать
- Используются для watchdog'ов и входа в ddb

Межпроцессорные прерывания, IPI

- На i386/amd64 для посыла и приема используется LAPIC
- Предназначены для управления работой CPU: синхронизация кэшей, запуск доп. процессов и т.д.

Архитектура ядра: Установка приоритета прерываний

```
splhigh()
splserial()
splsched()
splclock()
splstatclock()
splvm()
spltty()
splsofttty()
splnet()
splbio()
splsoftnet()
splsoftclock()
```

Set Priority Level, SPL

- Позволяют управлять конкурентным доступом к ресурсам
- Могут быть вызваны как в контексте прерывания, так и в контексте процесса
- Осуществляют системо-независимый доступ к IMR
- Упорядочены по приоритетам

Пример использования

```
int s = splnet();
/* критическая секция */
splx(s);
```

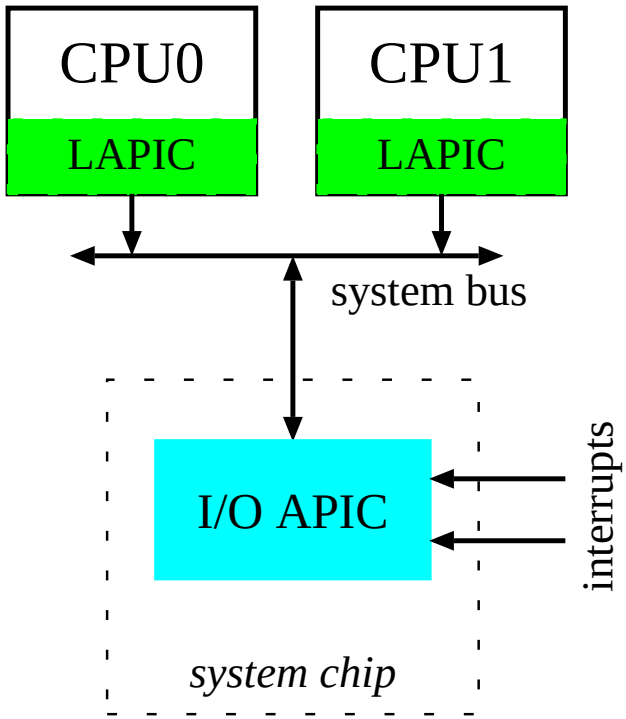
Типы контроллеров прерываний

- PIC (i8259)
- APIC (i82093)

Недостатки PIC

- 16 IRQ линий, для PCI устройств доступно меньше половины
- Использование разделяемых IRQ линий заставляет запускать все обработчики прерываний, зарегистрированные на линии
- Для того чтобы замаскировать прерывание необходимо произвести запись в регистр PIC
- Может быть использован только в однопроцессорных системах

Архитектура ядра: APIC



Преимущества APIC

- Поддерживает SMP: в каждом процессоре реализован LAPIC, в чипсете реализуется один или несколько I/O APIC
- LAPIC поддерживает 256 IRQ линий (32 зарезервированы)
- Поддерживается распределение прерываний по CPU
- Маскирование прерываний доступно на уровне CPU (в LAPIC)
- Встроенный таймер высокого разрешения

Архитектура ядра: Системный таймер

- Прерывается HZ раз в секунду (по умолчанию HZ=100)
- Обрабатывается функцией `hardclock(9)`
- Важно закончить обработку прерывания до прихода следующего
- Управляется подсистемой timecounters (`tc_init(9)`)
- Данные читаются из "железа"при каждом вызове `microtime(9)`
- По умолчанию используется медленный таймер i8254
- Возможно использование быстрых и дешевых таймеров: `acpitimer(4)`, `acpihpet(4)`, `amdpcib(4)`, `ichpcib(4)`, `glxpcib(4)` и др.
- Параллельно используется RTC (Real Time Clock) таймер
- `hz(9)` описывает глобальные переменные ядра, отражающие текущее состояние таймеров в системе

kern.timecounter

```
kern.timecounter.hardware=ICHPM
kern.timecounter.choice=i8254(0) ICHPM(1000) dummy(-1000000)
```

Архитектура ядра: hardclock(), statclock() и softclock()

hardclock()

- Обновляет системное время
- Обновляет данные таймеров, установленных *текущем* процессом
- Запускает statclock() softclock(), если нет отдельных таймеров

statclock()

- Уменьшает доступное процессорное время текущему процессу
- Собирает статистку по системе в момент прерывания
- Запускает schedclock() если нет отдельного таймера

softclock()

- Запускается как программное прерывание
- Обрабатывает очередь timeout(9)

Архитектура ядра: Программные прерывания (softintr)

- Выполняют низко-приоритетные задачи, поставленные в очередь
- Зуапускается когда меняется уровень приоритета прерываний: по выходу из ядра в пользовательское пространство или просто при понижении IPL (из `spllower()`)
- Существует три очереди программных прерываний: softclock, softnet и softtty
- Работают в контексте прерывания
- Запускаются на том же процессоре, на котором был вызван `softintr_schedule()`; в OpenBSD только на boot процессоре
- Могут быть прерваны аппаратным прерыванием

Архитектура ядра: Системный планировщик

- Приоритеты интерактивных программ повышаются, в то время как приоритеты программ, активно использующих процессор, понижаются
- Переодически запускающаяся `schedclock()` пересчитывает приоритет текущего процесса
- Приоритет процесса определяется значениями `p_estcpu` и `p_nice`:

$$p\_usrpri = PUSER + \left[\frac{p\_estcpu}{4}\right] + 2p\_nice$$

  - `p_estcpu` показывает сколько процессорного времени было использовано процессом *недавно*
  - `p_nice` устанавливается пользователем через утилиты `nice(1)` и `renice(8)`
- Процесс находится либо в очереди на исполнение (run queue) либо, ожидая освобождения ресурса, в очереди ожидания (sleep queue)

Архитектура ядра: Очереди исполнения и ожидания

Run queue

- 32 очереди, отосортированные по приоритетам
- Процесс, отработавший весь квант времени помещается в конец очереди
- Планировщик выбирает первый процесс с максимальным приоритетом
- Квант времени равен 10/HZ секунд (0.1 с)

Sleep queue

- Характеризуется каналом ожидания (wait channel)
- Вызывая `tsleep(9)`, процесс автоматически попадает в очередь ожидания
- `wakeup(9)` предназначен для извещения ожидающего процесса об освобождении ресурса

Архитектура ядра: Контекстные переключения

Контекст процесса может быть переключен:

- явно (вызвав `sched_yield()`)
- вынужденно (ожидая освобождения ресурса, процесс вызывает `tsleep(9)`)
- принудительно если был израсходован отведенный ему квант времени

Основные функции

- `mi_switch()` выполняет архитектурно-независимые операции, подготавливая ядро к переключению контекста
- `cpu_switchto()` выполняет непосредственное переключение, загружая новое адресное пространство процесса, восстанавливает стек, состояние процессора и т.д.

Архитектура ядра: Критические секции

- На однопроцессорных системах конкурентный доступ к ресурсам управляется повышением текущего уровня приоритета прерываниый, IPL
- Biglock актуален только для многопроцессорных систем и берется в начале каждой процедуры вхождения в ядро (системный вызов, прерывание и т.п.)
- Системные вызовы, не нуждающиеся в biglock'е, помечаются как `NOLOCK`
- Для того чтобы уйти от biglock'а необходимо разбить его на *б*ольшее количество более мелких блокировок (fine grained locking)

Архитектура ядра: Синхронизация

- Основа синхронизации – атомарный CAS (cmpxchg в x86)
- Compare and swap:

```
int cas(int *flag, int o, int n)
{
    if (*flag != o)
        return (1);
    *flag = n;
    return (0);
}
```

Sleeping lock

```
while (cas(flag, 0, 1))
    sleep(&flag);
```

Spinning lock

```
while (cas(flag, 0, 1))
    /* do nothing */;
```

- Biglock в OpenBSD реализован функцией __mp_lock() и является spinning lock'ом. На i386 и amd64 оптимизирован:

```
__asm __volatile("pause": : :"memory")
```

Архитектура ядра: Блокировки (Locking)

Типы блокировок доступных в ядре

- Sleeping lock (`rwlock(9)`)
- Spinning lock (`mutex(9)`)

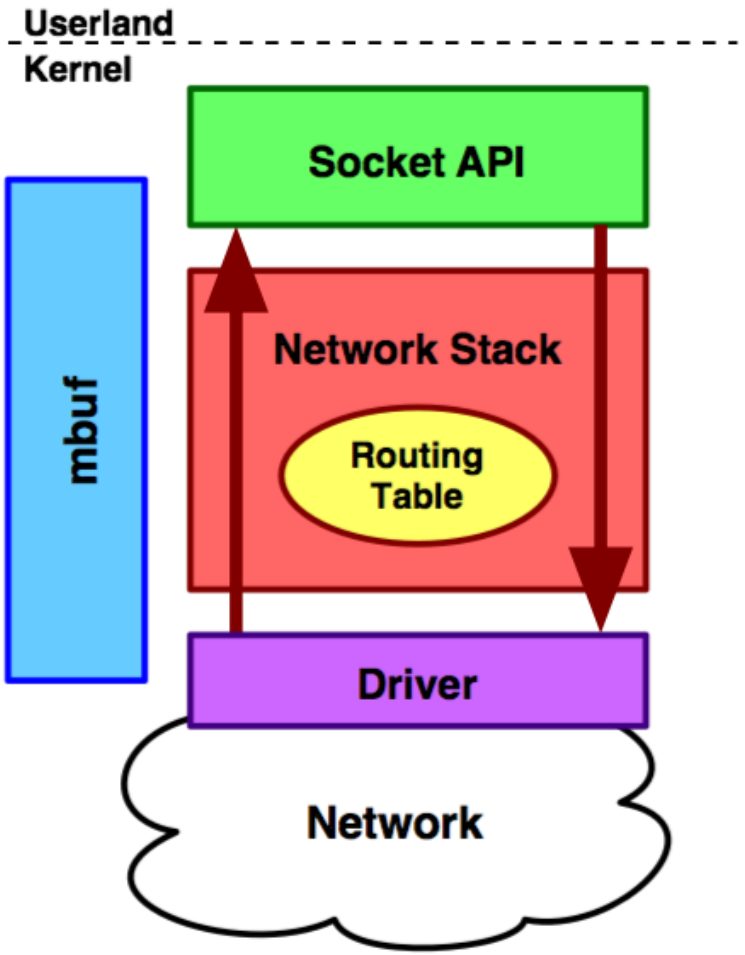
`rwlock(9)`

- Использование возможно только в контексте процесса
- Помещает процесс в sleeping queue
- Не использует ресурсы процессора при ожидании

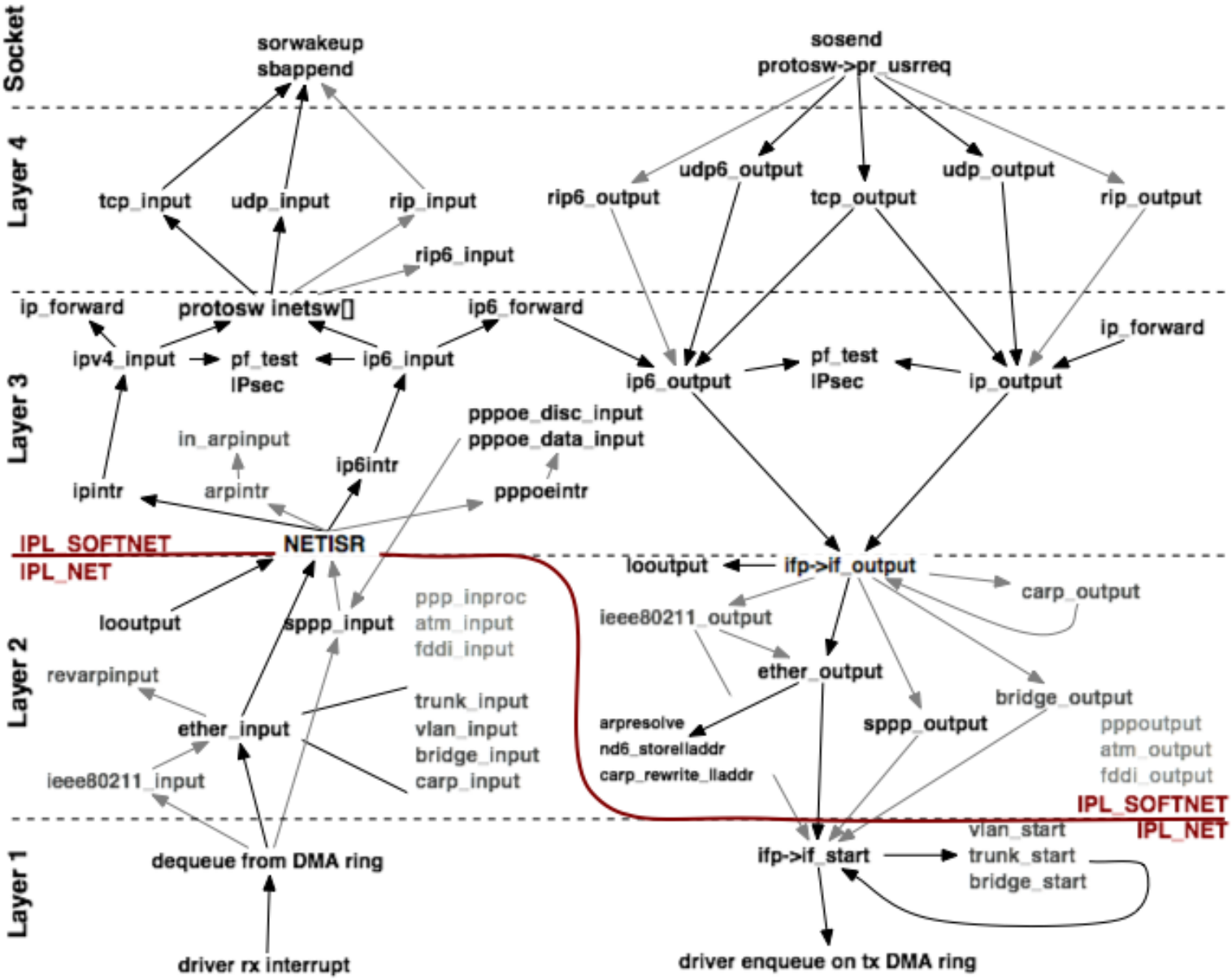
`mutex(9)`

- Использование возможно как в контексте прерывания, так и в контексте процесса
- Использует ресурсы процессора при ожидании
- В контексте процесса используется для быстрых критических секций

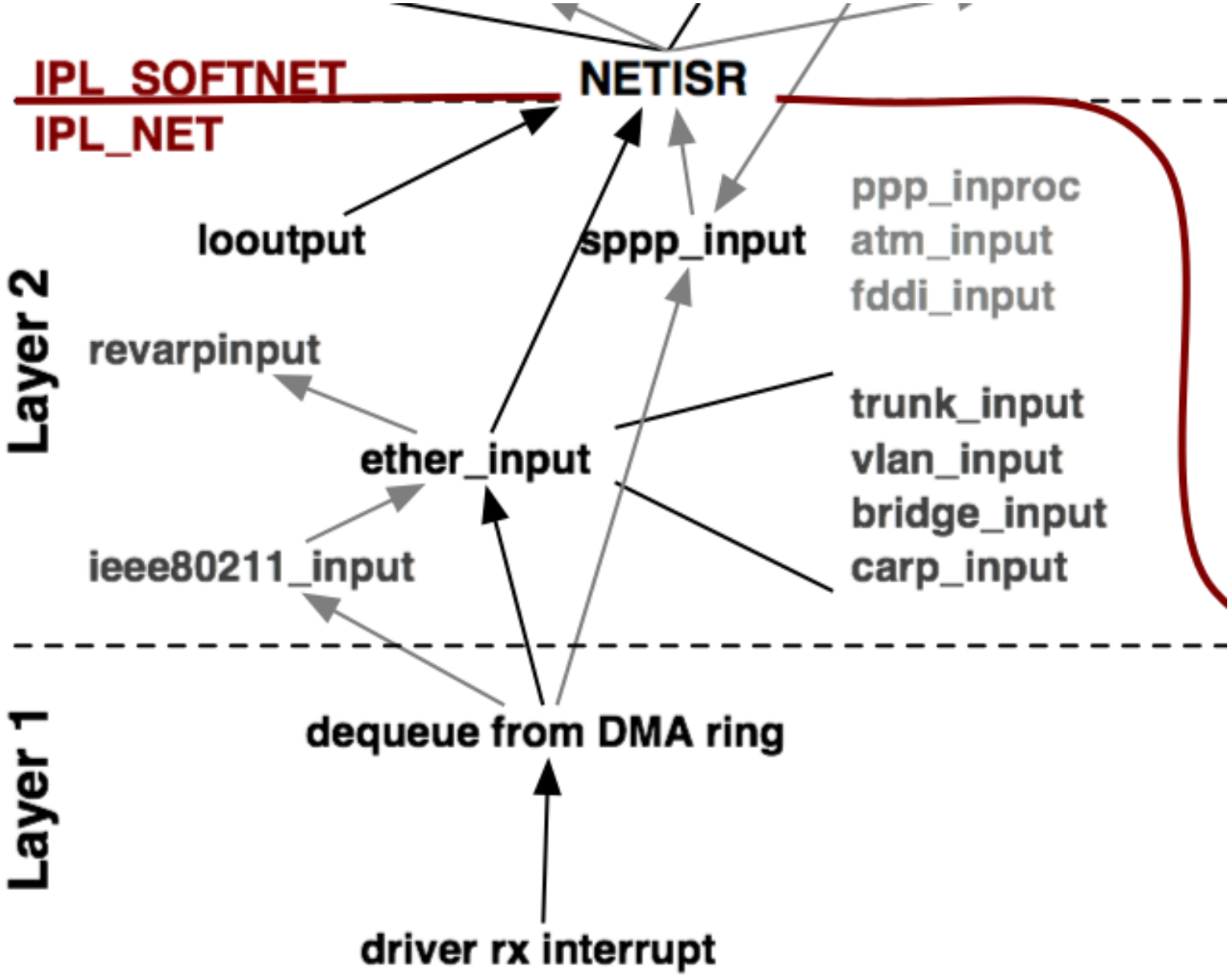
Сетевой стек: Структура стека



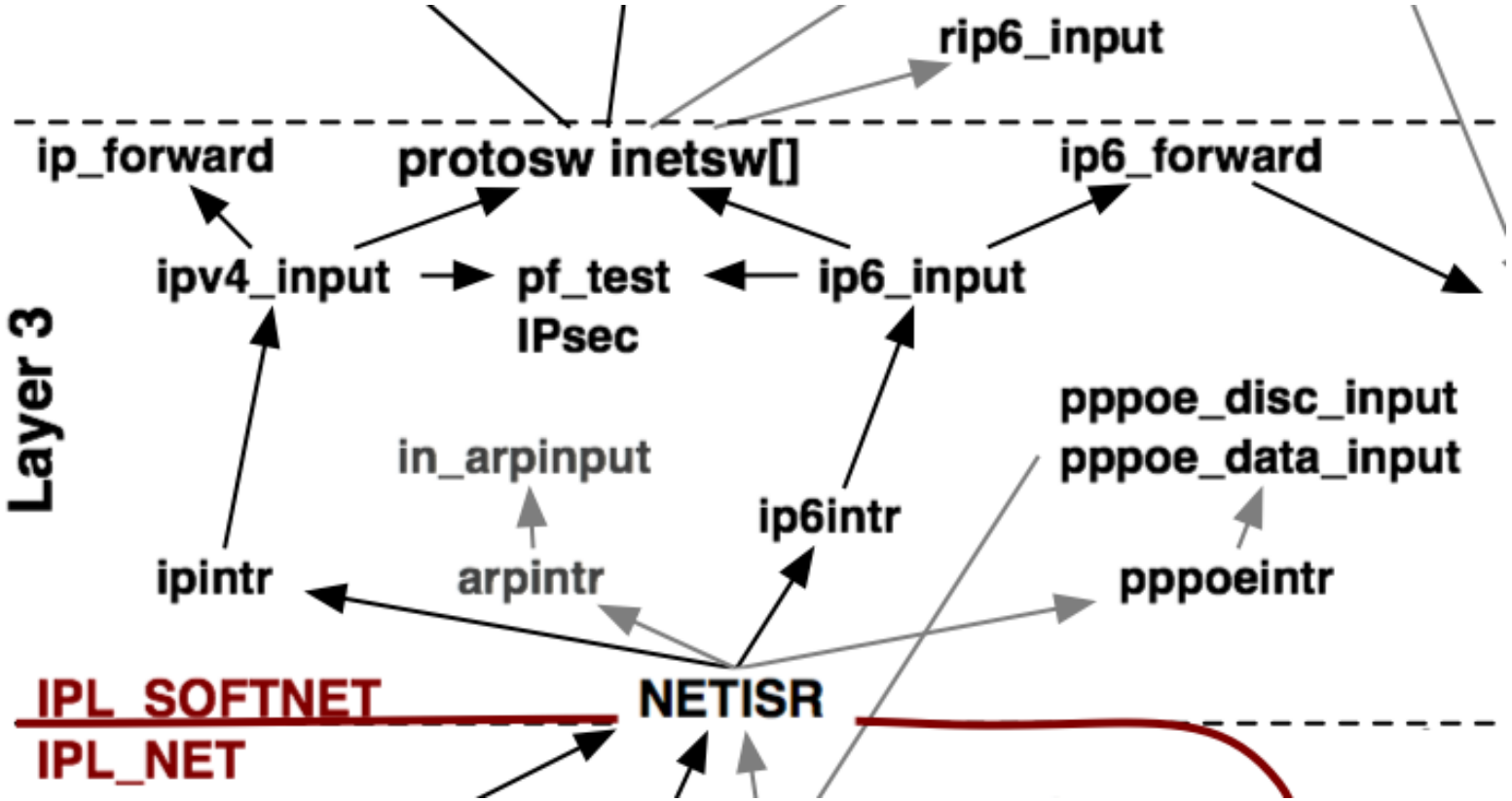
Сетевой стек: Общая картина



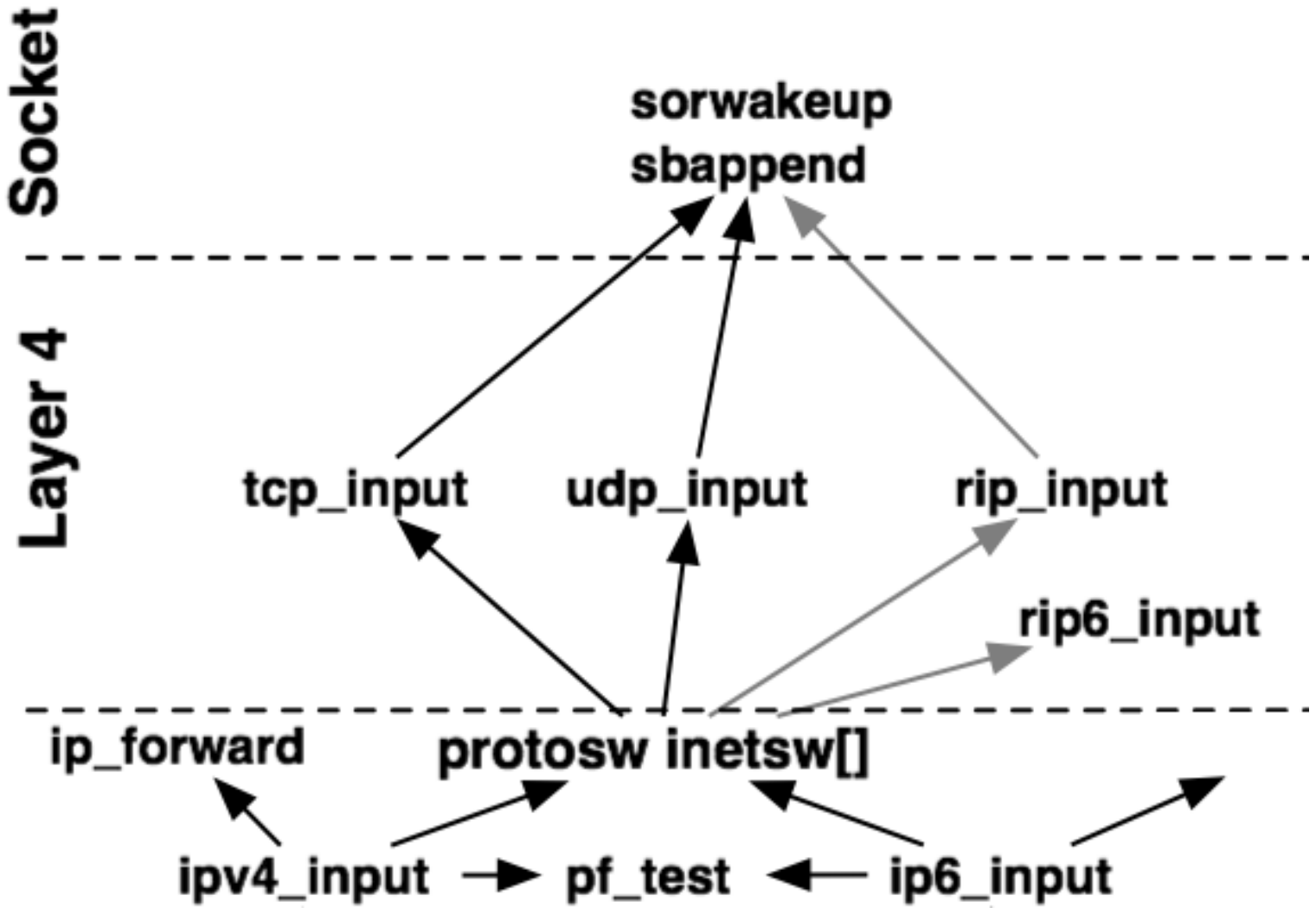
Сетевой стек: Прием пакета, канальный уровень



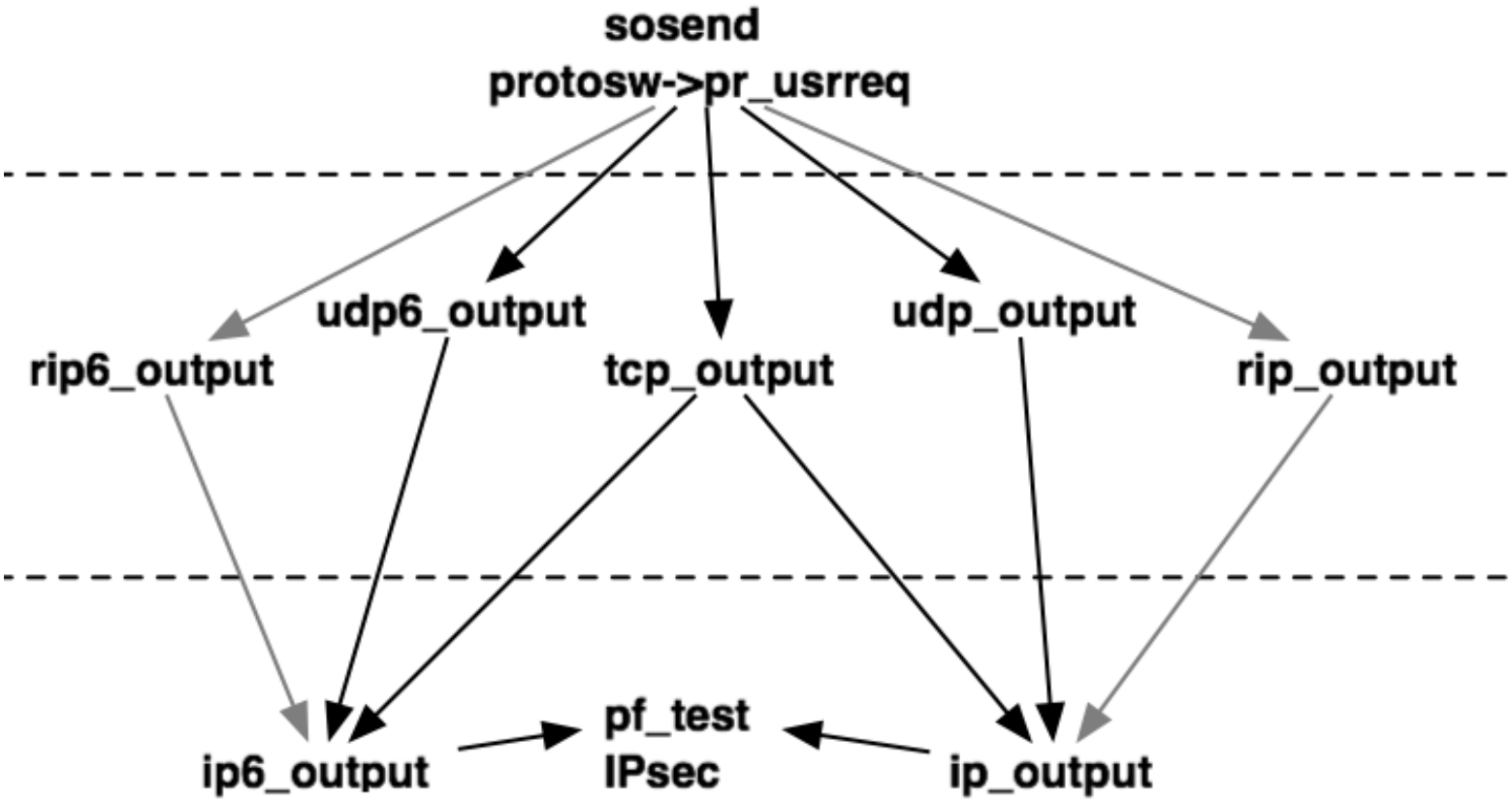
Сетевой стек: Прием пакета, сетевой уровень



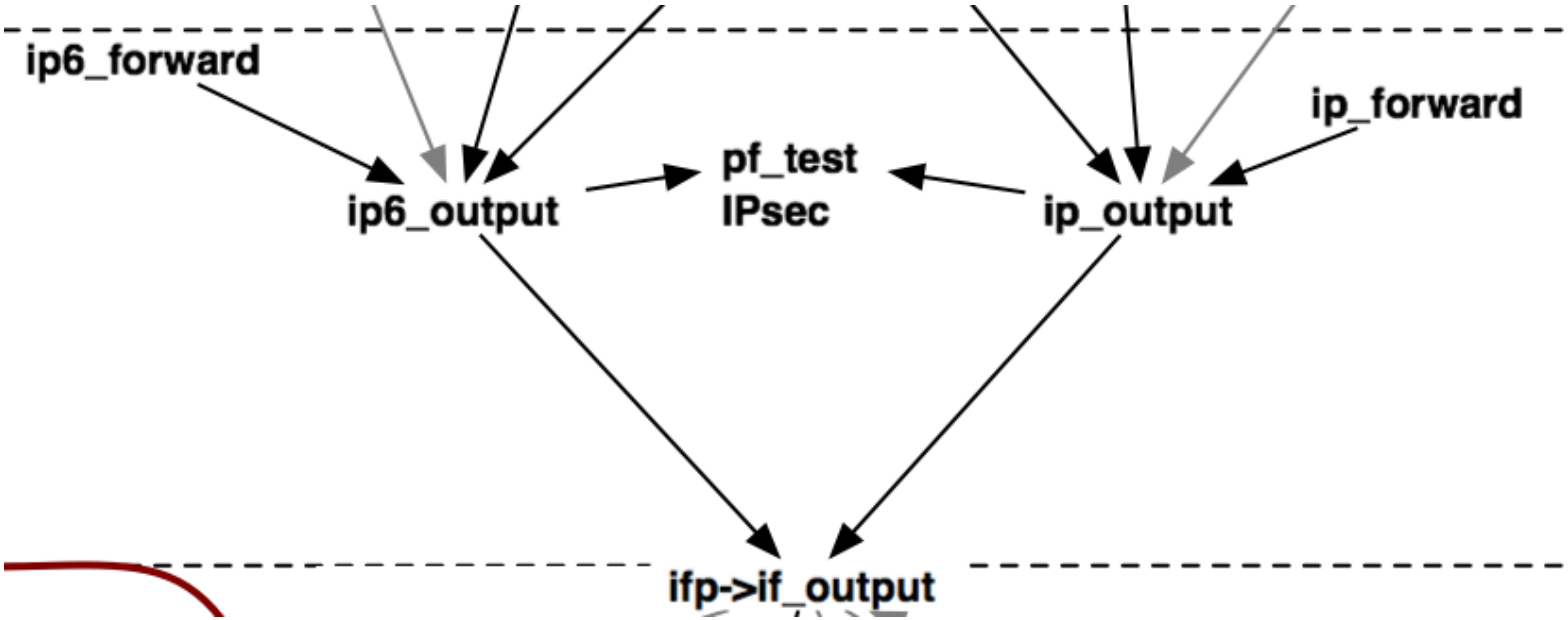
Сетевой стек: Прием пакета, транспортный уровень



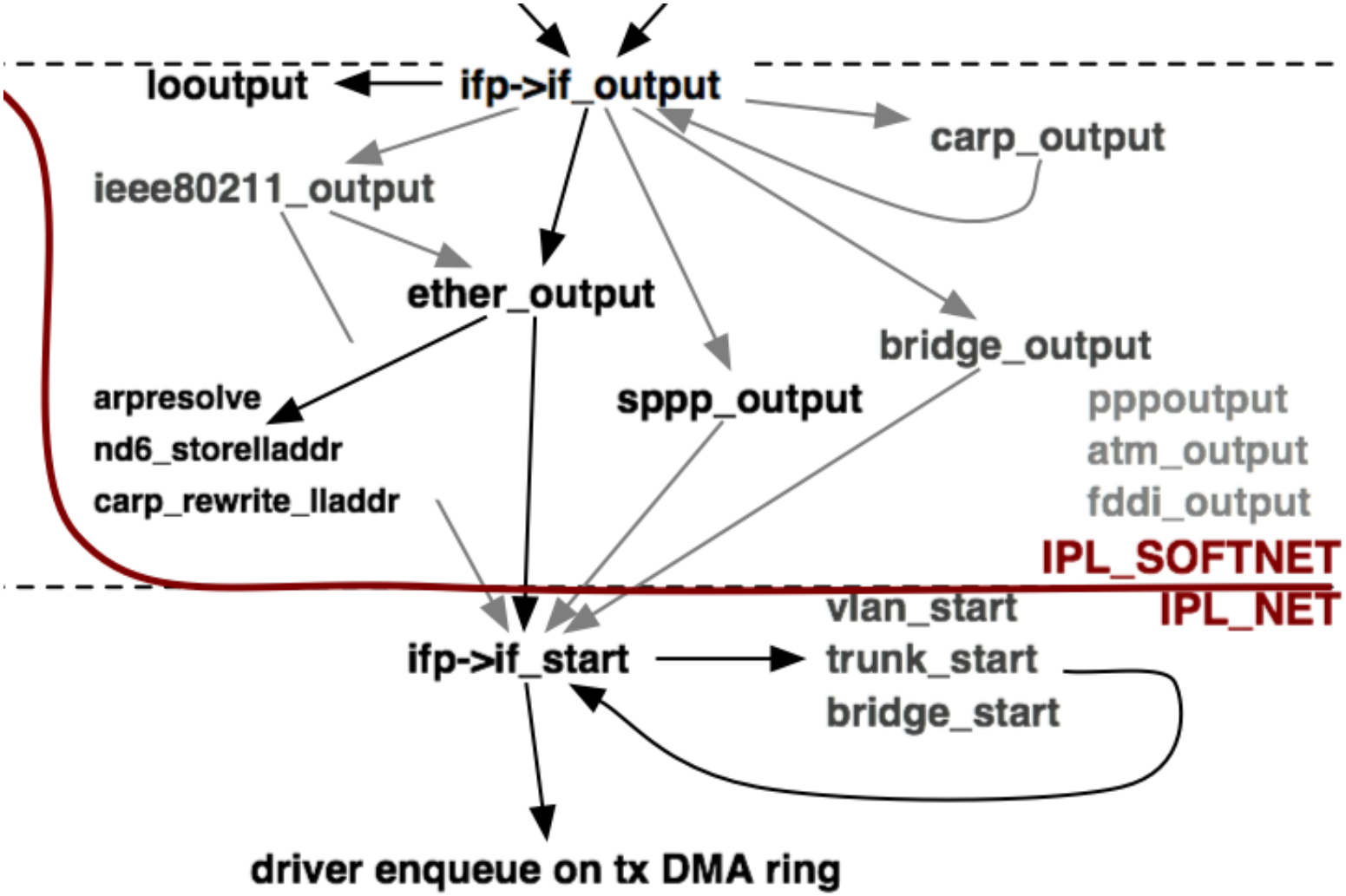
Сетевой стек: Отправка пакета, транспортный уровень



Сетевой стек: Отправка пакета, сетевой уровень



Сетевой стек: Отправка пакета, канальный уровень



Сетевой стек: Пути улучшения производительности

- Устранение biglock, переход к fine grained locking
- Использование дополнительных процессоров для обработки прерываний
- Использование node-bound процессов ядра для обработки softintr
- Использование информации об архитектуре I/O на конкретной системе (io-bound threads)
- Использование информации о локальности памяти (NUMA-aware) и использование node-bound пользовательских процессов
- Использование поллинга при большой нагрузке
- Улучшение качества драйверов
- Использование качественного оборудования

Заключение

Let's have a beer, shall we?