

Kernel TCP/IP evolution

Making the code understandable

Lower latency

Kernel Networking APIs

**Erik Nordmark
Distinguished Engineer
Solaris networking
September 2007**

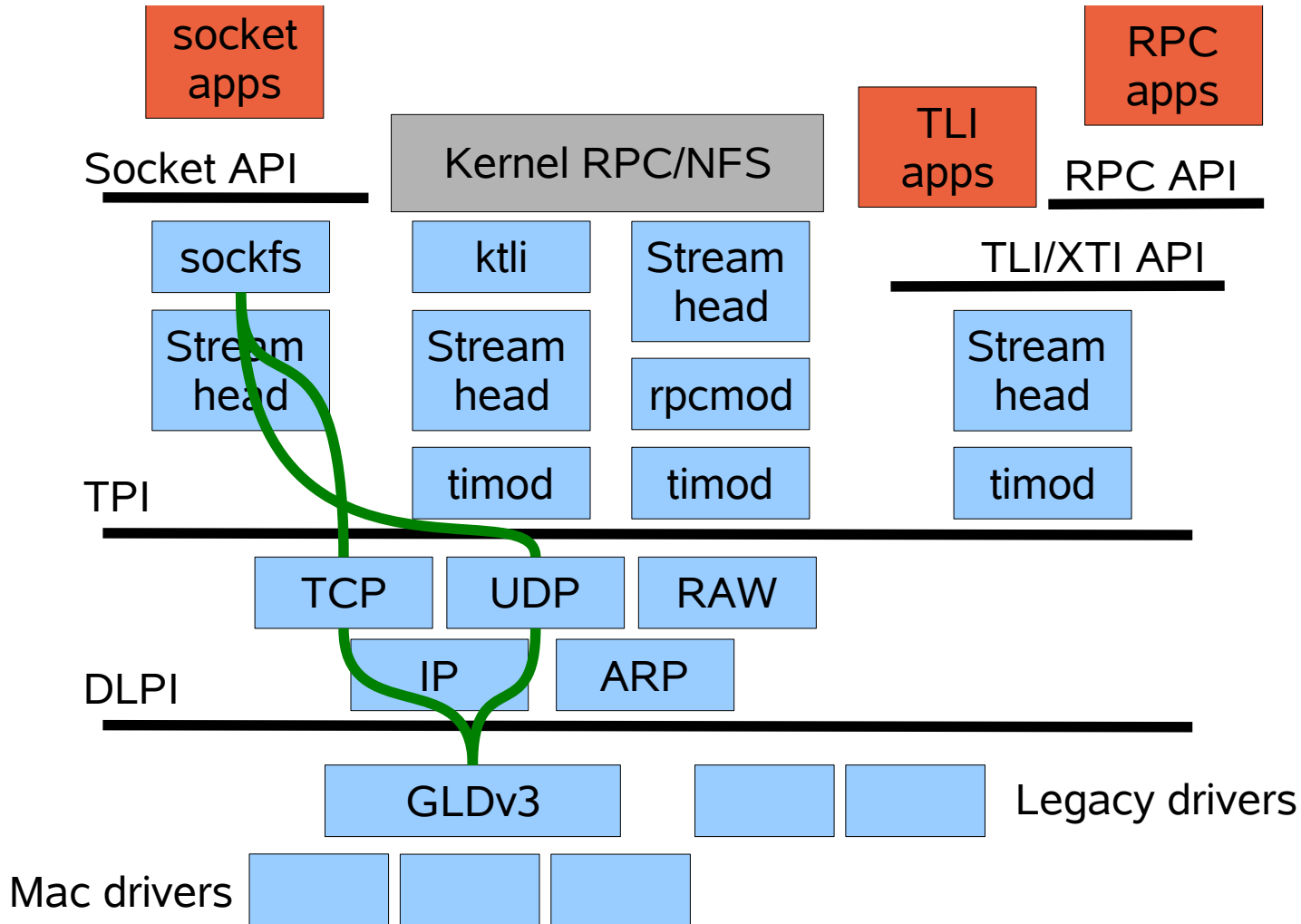
Agenda

- The big picture for kernel TCP/IP networking
- Questions *YOU* can help answer
- Project Volo – pluggable sockets and latency
- GLDv3 – the network device driver framework
- IP Datapath Refactoring

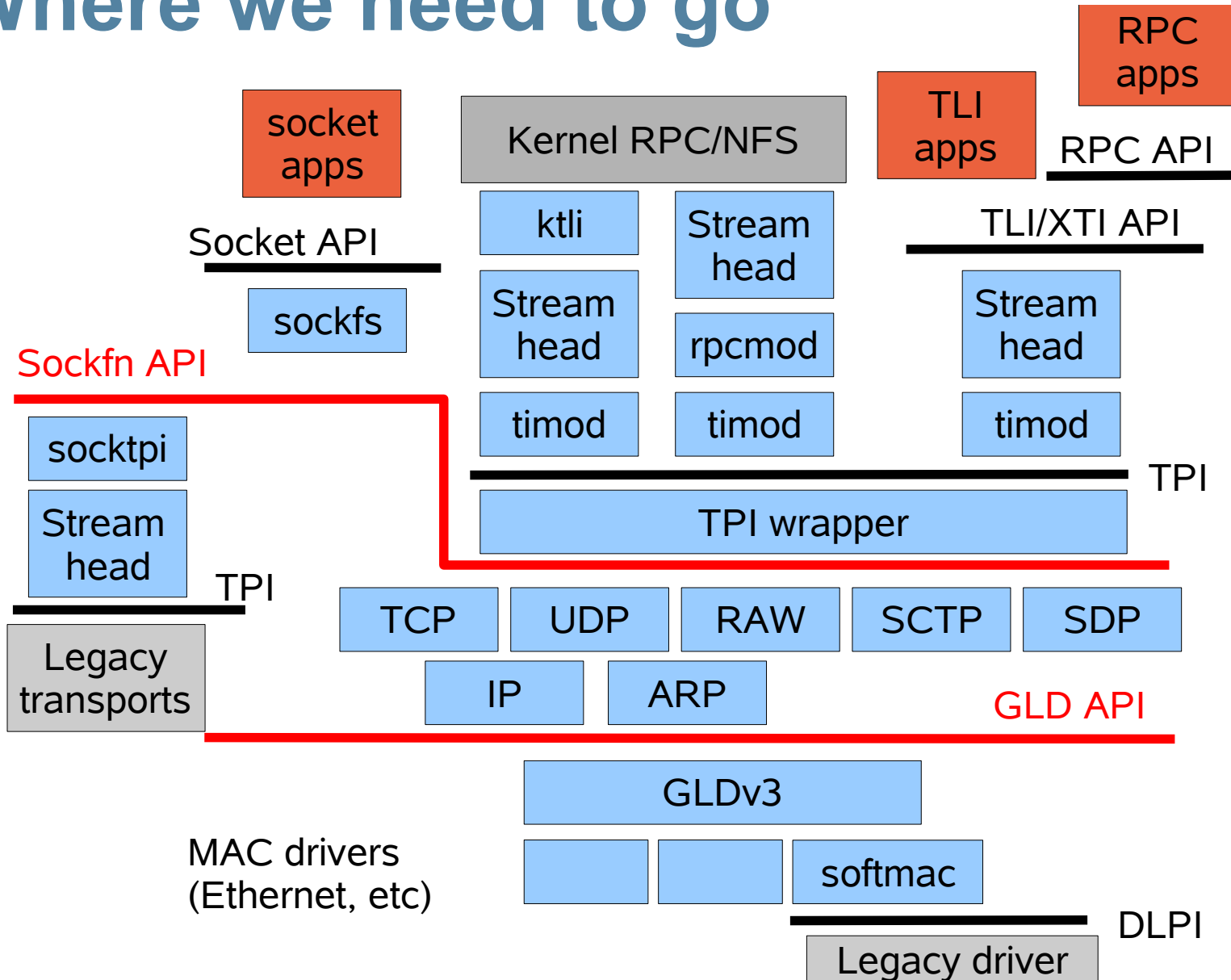
Big Picture

- We want to move to a simpler to understand and better performing structure for the code
- The current hybrid of STREAMS and function calls makes life even more complex than the straight STREAMS implementation of old
- Goal is to separate TPI and DLPI code and have all of the socket datapaths avoid using TPI and DLPI
 - > TLI/XTI/RPC will still use TPI from above
 - > Sockfs needs to be able to talk to 3rd party TPI providers
 - > Presumably continue to support I_PUSH on a socket
 - > DLPI perhaps only in softmac down the road
- Define function call “transport” interfaces between sockfs and transports

Solaris 10 stack



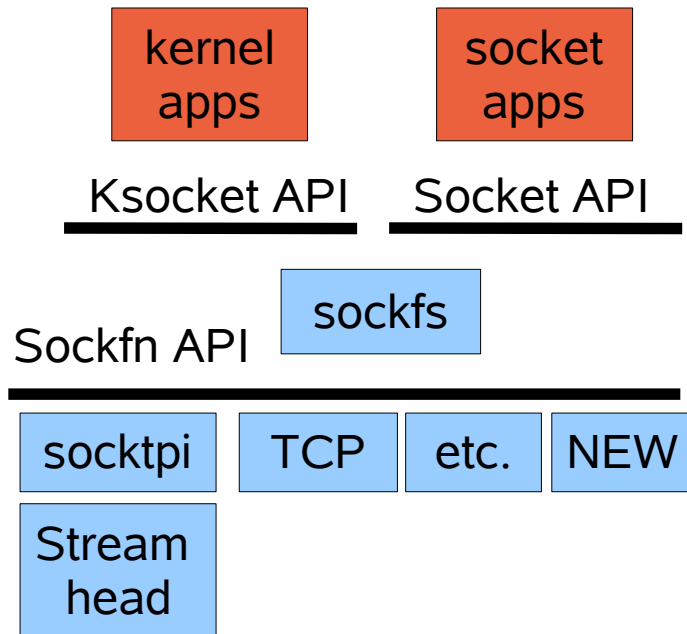
Where we need to go



Questions for the Audience

- As we evolve, easy to add APIs/hooks
- But where should we add them?
 - > Making it easier to develop a network device driver?
 - > Send/receive raw Ethernet packets?
 - > Allow new socket providers? (AF_ some new protocol)
 - > Put (part of) a socket application in the kernel
 - > Hooks for IP packet processing (in, out, forward, ???)
 - > Hooks in TCP?
 - > Kernel support for proxies (splice() system call?)
- Does the rest of the presentation make you think of new ones?

Project Volo



- Ability to plugin NEW socket provider
 - > For new protocol and/or address family
 - > No STREAMS required
- Complete function call interface
 - > Use in TCP, UDP, etc.
- Kernel sockets API
 - > Socket “applications” that run in the kernel for performance
- Lower latency

GLDv3 Evolution

- Solaris already has
 - > Link aggregation, VLANs, interrupt blanking
 - > MAC plugin – used by WiFi
- Underway
 - > Clearview IP-in-IP tunnels (using MAC plugin)
 - > Crossbow
 - > VNICs, polling, resource controls, classification APIs
 - > STREMS-free flow control
 - > Brussels – common code for driver tunables
 - > E.g., jumboframes on/off
 - > Nemo unification – softmac driver around legacy DLPI
 - > Vanity naming – mostly outside of the kernel
- Once this is nailed down, a stable API
- Document GLDv3 for driver writers

Goals for IP Datapath Refactoring

IP

- Delete code – lots of it
 - > More than 5 times as much code in IP as in TCP
 - > Fastpaths should not be necessary
- Make the code much easier to understand and support
 - > For the community, new hires, support, myself!
- Make it easier to evolve
 - > E.g., where in the transmit path to insert my new code?
- Will it be faster?
 - > At least more uniform speed – today if you fall off a fastpath the performance drop can be spectacular
 - > The code should be easier to evolve towards even better performance
 - > And avoid the “can I take the fastpath checks”

Source of complexity

- We've added a lot of new IP features over the years
 - > IPv6, IPsec, IPMP, CGTP, routing sockets, SO_DONTROUTE, IP_NEXTHOP, zones, TX, etc
- In light of those additions some original design decisions seem suboptimal
 - > Placement of asynchrony to do ARP - ip_newroute
 - > Tying routing and source address selection together – ire_src_addr
 - > Initial assumption of a single IRE_CACHE for a unicast destination
 - > IRE_CACHE as the holder of path MTU information
- The result is incomprehensible complexity in the IP data paths

Trouble caused by ip_newroute

- Causes asynchrony before we are done with the conn_t
 - > Need to save away IPsec and socket option information
 - > Causes use of M_CTL all over IP (even for non-IPsec stuff) and use of ip6i_t for similar purposes for IPv6
- Made worse by transports passing additional information using M_CTL, ip6i_t, etc.
 - > But with function calls to ip_output() we could use function call arguments instead
- Caused multirt (CGTP) to be much more complex than otherwise
- Follow Surya's lead and do ARP at the end – just before calling the driver
 - > Can create an initial NCE early

Issues with source address selection

- Original idea was that `ip_wput()` could easily set IP source from the IRE
 - > But in most cases the transport has already set this
- Today a lot more factors taken into account than in 1991
 - > Zoneid, IPMP groups, VNI, deprecated, IPv6 table
- We have source address selection in both `ip_bind` and in `ip_newroute`
- With a single `ip_select_source()` function plus caching in the `conn_t`, no more need to keep a source in the IRE
- Note: need to maintain `RTF_SETSRC`; a new `ire_setsrc_addr` address field will be used for that

Single IRE_CACHE per unicast destination?

- Before SO_DONTROUTE, IP_XMIT_IF, IP_NEXTHOP, etc we had a single IRE_CACHE for a destination IP address
- Today we make distinction between the “normal” one and specific ones tied to specific socket option settings
 - > A number of IRE_MARKs for this
 - > In some cases we do ip_newroute for every packet – IRE_MARK_NOADD
- Why this complexity? Can't we move the route selection to one place and cache the result in the conn_t and remove the marks?
- Hint; we already cache an IRE pointer for TCP

Path MTU and IRE_CACHE

- Path MTU is defined as the minimum of the observed path MTU over all (recently) used paths through the network; via different routers and links
- With a single IRE_CACHE per destination, the only 1991 issue was source routed packets
 - > The IRE_CACHE we use is for the first hop in the source route, not the destination
 - > We turn DF off for those
- With IP_NEXTHOP etc the IRE_CACHE can be completely unrelated to the final destination
- Tying path MTU to routing seems problematic because of all the socket options that affect how packets are routed out of the box

Other sources of complexity

- Using `queue_t` as a handle in IP – some times it is an `ill_t` and other times a `conn_t`
 - > Should simplify so that functions either take a `conn_t` or an `ill_t`
 - > Only real use of `queue_t` is IP flow control for UDP/ICMP; crossbow-gate code doesn't use the queue
- Having `ill` and `ipif` pointers in too many places
 - > `conn_t` in particular. Should be sufficient to record an `ifindex` for e.g., `IP_BOUND_IF` and an IP address for `IP_MULTICAST_IF`.
 - > With function calls to `ip_output` we can easily cache the result of looking up e.g., `IP_BOUND_IF` to find the IRE.

What are we changing

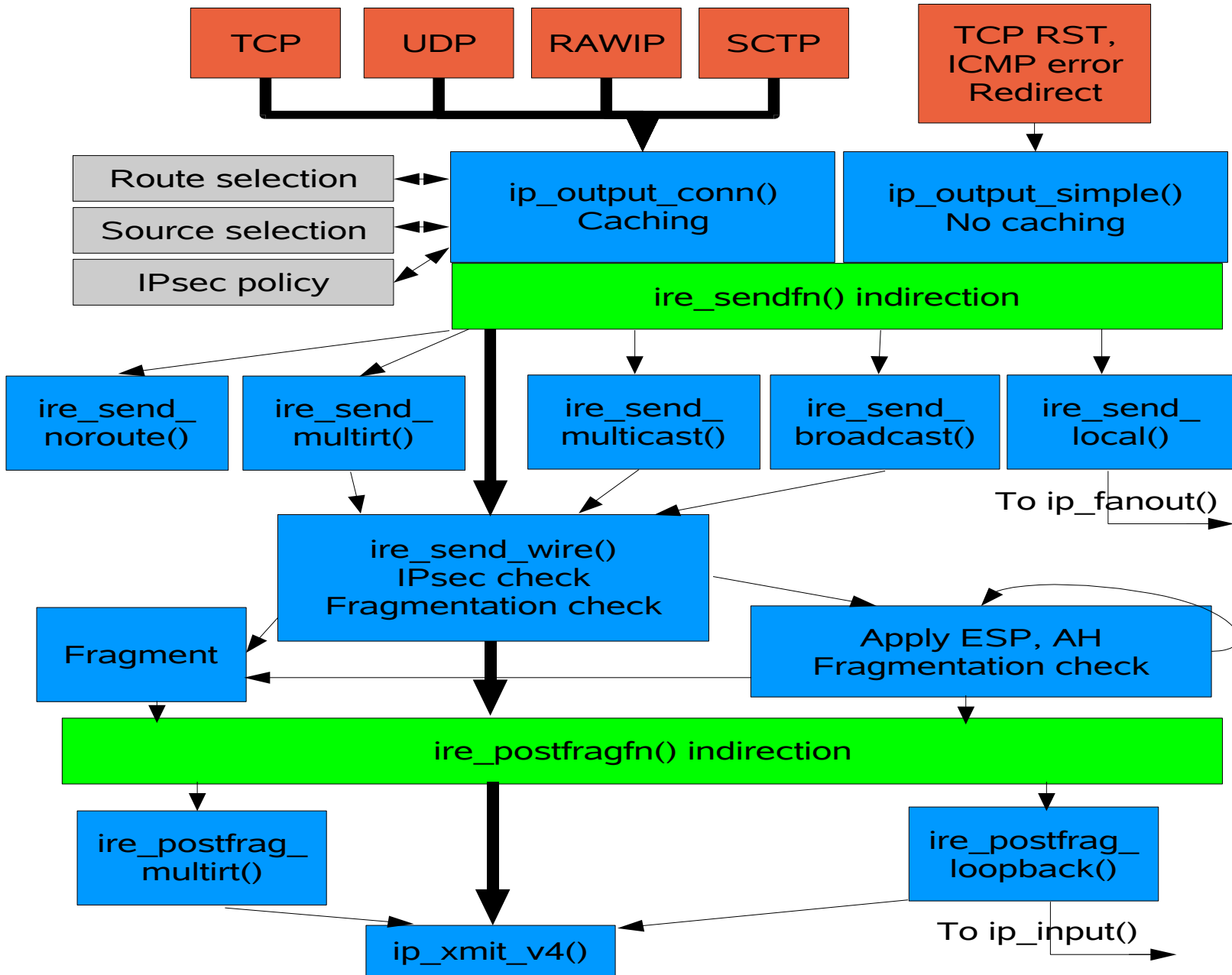
- First get all transports to call ip_output – no putnext from above
 - > This is the IP Plumbing evolution – code reviewed and soon to be putback into Nevada
 - > Allows us to handle tcp, sctp, udp, icmp uniformly
- No more ip_newroute, IRE_CACHE, ire_src_addr, ire_ipif
- Remove (all? most?) IRE_MARKs
- No ire_cachetable – put IRE_LOCAL, BROADCAST etc in forwarding table
- Simplify conn_t to not cache ill and ipif pointers

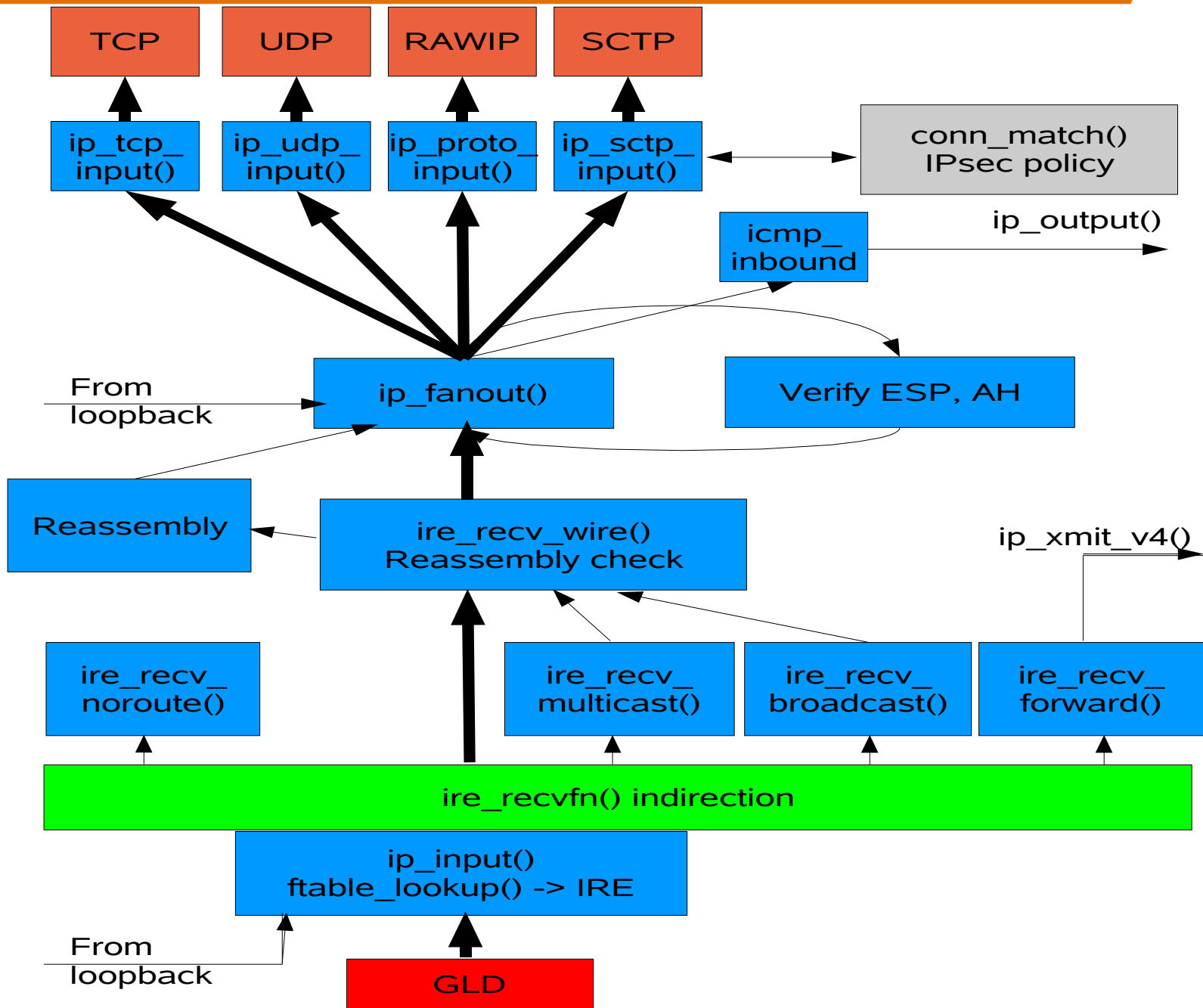
What do we need to add

- A new **optional** Destination Cache Entry
 - > Independent of routing; one per destination IP address
 - > Contains Path MTU, transport metrics, IP ID assignment
 - > Hook for Mobile IP, SHIM6 and other protocols
- New way to handle invalidation of cached IRE/DCE
 - > Avoids global walks (better real-time behavior)
 - > Implies single comparison in the data path
- Data structures for IP transmit and receive attributes
 - > Replaces M_CTL in datapaths and ip6i_t
 - > Passed as an extra argument to ip_output (transmit) and conn_recv (input)
- Re-written data paths in IP
 - > Using function pointers to factor out different behavior for different types of IREs

Example: connected transmit path

- When TCP, UDP etc connects it uses IP to
 - > Determine an IP source address
 - > Cache an IRE, DCE, NCE (NCE could be in ND_INITIAL – ARP done much later)
 - > Latch the IPsec policy
- Datapath just verifies that the cached information is current
 - > Source address still valid
 - > Route hasn't changed
 - > DCE/Path MTU unchanged
 - > NCE is still valid





What are we NOT changing

- Routing sockets
 - > Once Clearview IPMP reduces use of IPIF_CHANGING then rts can call a synchronous ip_rts_request.
- IP bind and friends?
 - > Once less IPIF_CHANGING then all ip_bind operations can be synchronous; could have a set of function calls to validate a source address, pick a source address. No more T_BIND_REQ to IP.
 - > Hopefully Volo will do this
- Make ill/ipif binding be dynamic
 - > IP_ADDMULTI with ifaddr = INADDR_ANY should join on the currently best interface, and change as NWAM changes the set of interfaces
 - > Likewise for IP_MULTICAST_IF

What are we NOT changing

- ECMP (equal cost multipath routing)
 - > Made easier by no IRE_CACHE
 - > Different conn_t's can cache different IREs
 - > Need some spreading algorithm for forwarded packets
- Two-step FIB for BGP routers
 - > Idea would be to have kernel handle indirect routes to BGP nexthops by looking up the BGP nexthop in the ftable to find the IGP nexthop/route
 - > Can cache the result in ire_nexthop
 - > The cache consistency mechanism is designed to be able to handle the two-step lookup and cache

How can you help?

- Hard to coordinate multiple people **deleting** code
 - > Quite different than **writing** code projects!
 - > Once we have udp using the new datapaths it will be easier to spread the work; apply to TCP, SCTP etc
- Please participate in design review
- Think about what odd behavior you've might have seen that we need to preserve
- Help figure out how to test this sufficiently
 - > Can we do basic block code coverage? kcov?

Questions for the Audience

- Making it easier to develop a network device driver
 - > GLDv3 should help
- Send/receive raw Ethernet packets
 - > Some new API to talk to pseudo-hardware at GLDv3?
 - > With Crossbow classification and resource controls?
- Allow new socket providers
 - > Pluggable sockets with Volo project
- Put (part of) a socket application in the kernel
 - > Kernel sockets in Volo
 - > What performance do you need? Select()?
- Hooks for IP packet processing
 - > Pfhooks in already. Extend to wider use than IP Filter?
- Kernel support for proxies (splice() system call?)

More information

- Volo
<http://opensolaris.org/os/project/volo/>
- Clearview
<http://opensolaris.org/os/project/clearview>
- Crossbow
<http://opensolaris.org/os/project/crossbow>
- Brussels
<http://opensolaris.org/os/project/brussels>
- TCP/IP plumbing evolution
<http://cr.grommit.com/~nordmark/ipd0/ip-plumbing-design.p>

Kernel TCP/IP evolution

Making the code understandable

Lower latency

Kernel Networking APIs

erik.nordmark@sun.com