PackOS: A Microkernel Based on IPv6

Master's thesis by John Stracke

Introduction

- The core of any microkernel is its IPC.
- Most microkernels use some sort of RPC.
- PackOS uses IPv6 instead.

Why would you do that?

- Reuse the existing IP-based protocols.
- Communicate with outside world.
- Simplify the kernel.
- IPv6 instead of IPv4 because of address space.
  - Each process needs an address.

Benefits

- All kernel calls are O(1).
  - No outstanding kernel operations.
  - No kernel stacks, which means cheap threads.
- Simplifies process migration.
  - All resources identified by IPv6 address.
  - Copy the memory space, use Mobile IP to deliver.
  - Notify process to start using local resources.

Prior work

- The main influence on PackOS was L4.
- L4 has RPC-based IPC.
  - Partially zero-copy.
  - Structured messages.
  - Clans & Chiefs.
  - Overcomplicated.

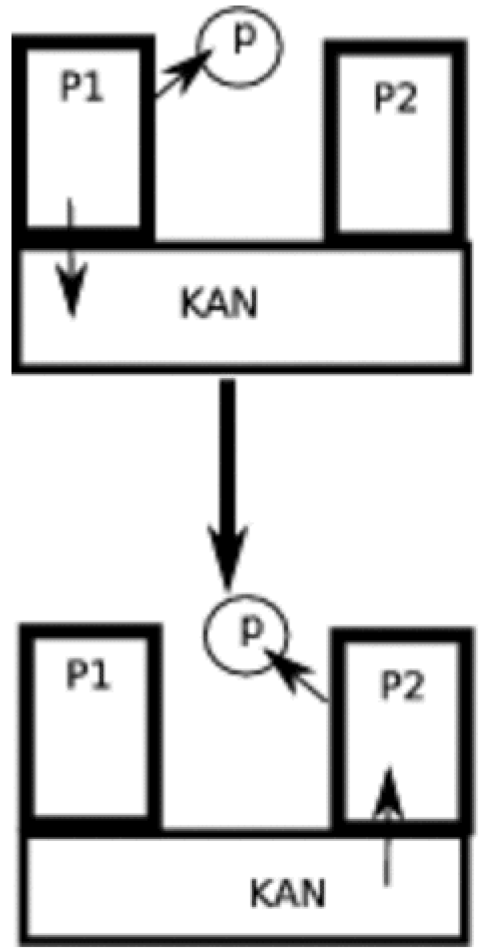
What PackOS learned from L4.

- IPC performance is vital.
  - Slow IPC means large-grain components.
  - Large-grain components limit flexibility.
- L4 features PackOS adopted:
  - Zero-copy.
  - User-space process management.
  - User-space drivers.

PackOS's innovation: the KAN

- Kernel Area Network: a virtual link layer.
- Asynchronous IPC.
- Packets are pages.
  - Mapped out of sender's space, into recipient's.
  - Zero-copy.
- Every process has at least one KAN address.
- Network interfaces are routers.

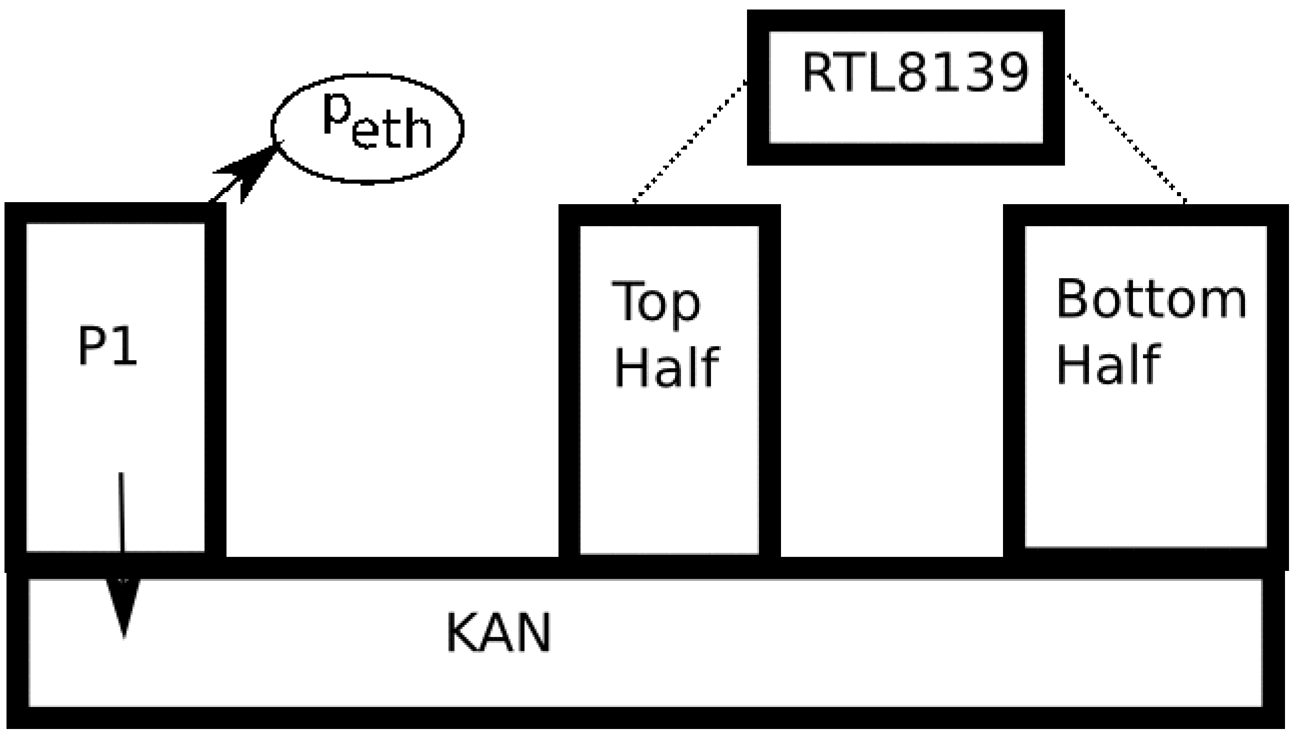
Zero-copy IPC



The KAN (continued)

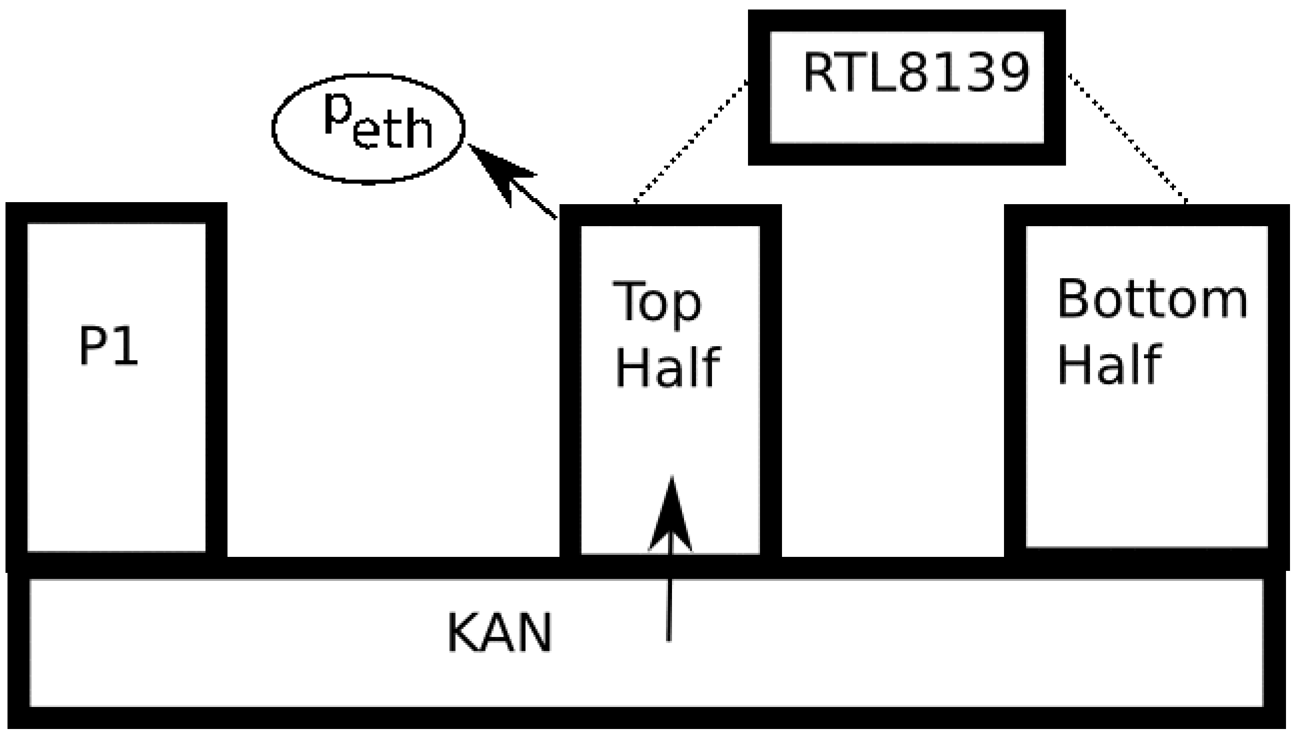
- Finds unusual uses.
- Interrupt handling via IPC.
  - User space driver requests interrupt notification.
  - On each interrupt, a KAN packet is delivered to the bottom half.
  - Bottom half manages the hardware, sends packet to top half; kernel clears interrupts.
  - Top half talks to other processes.

Sending an Ethernet packet: 1



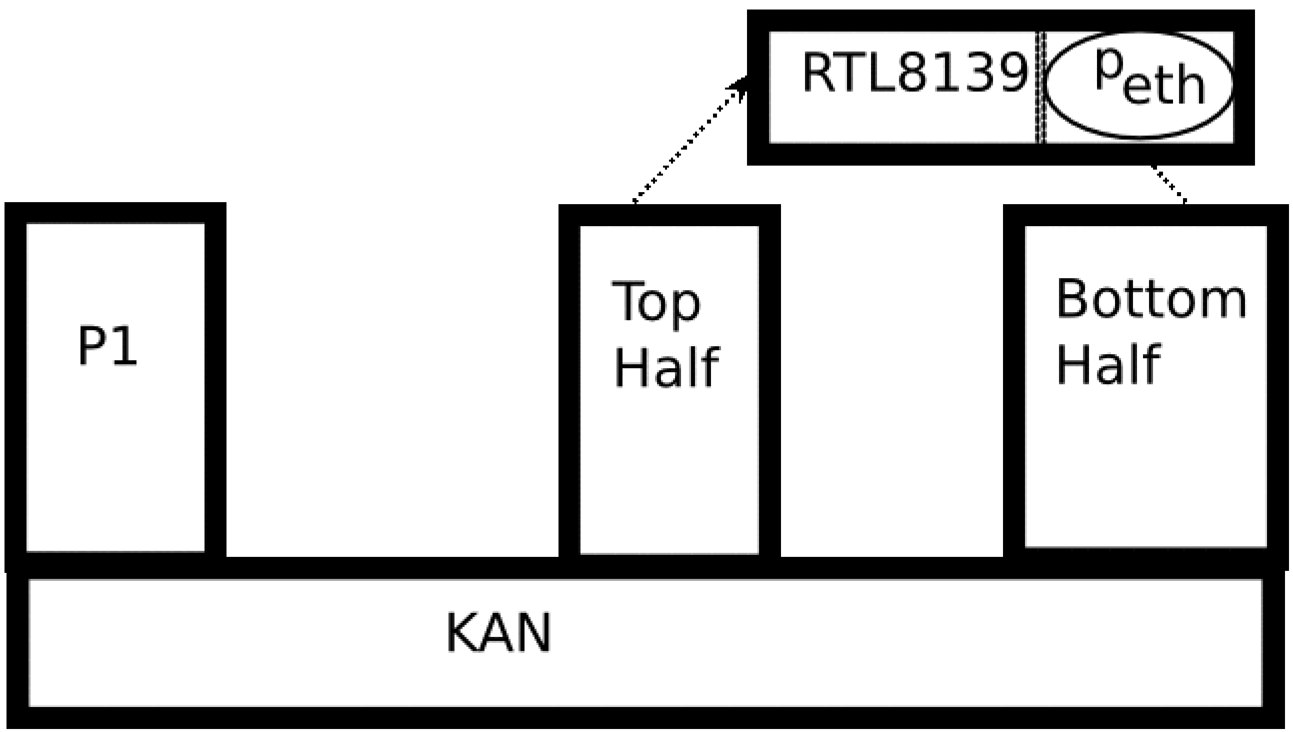
P1 prepares a packet to send over Ethernet.

Sending an Ethernet packet: 2



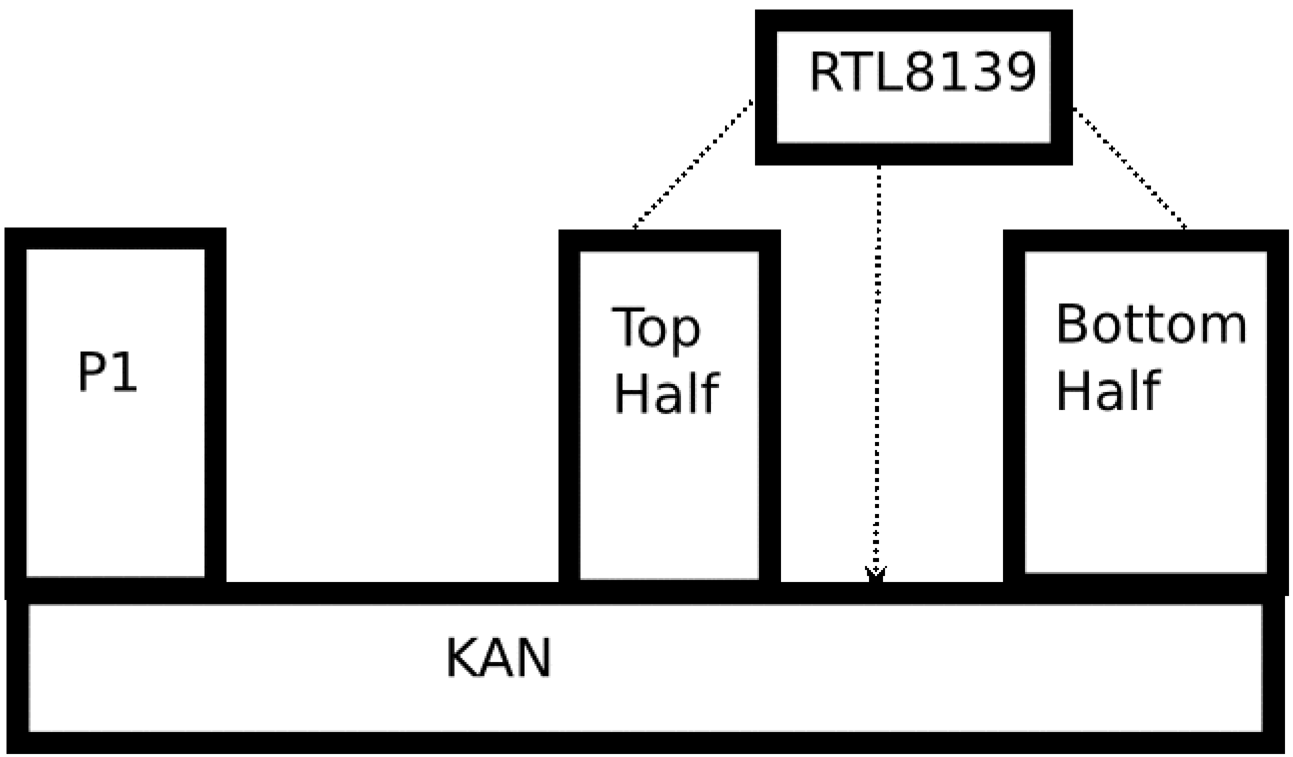
The top half receives the packet.

Sending an Ethernet packet: 3



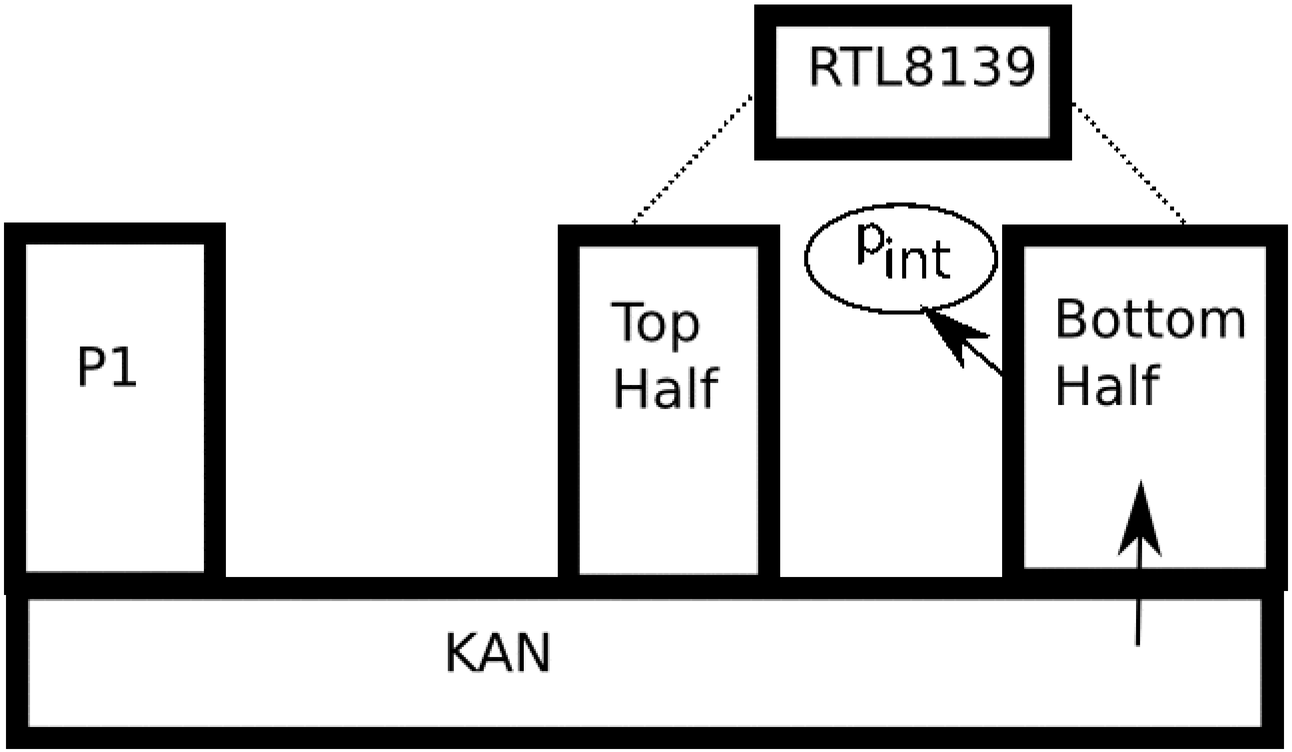
The top half copies the packet into the NIC.

Sending an Ethernet packet: 4



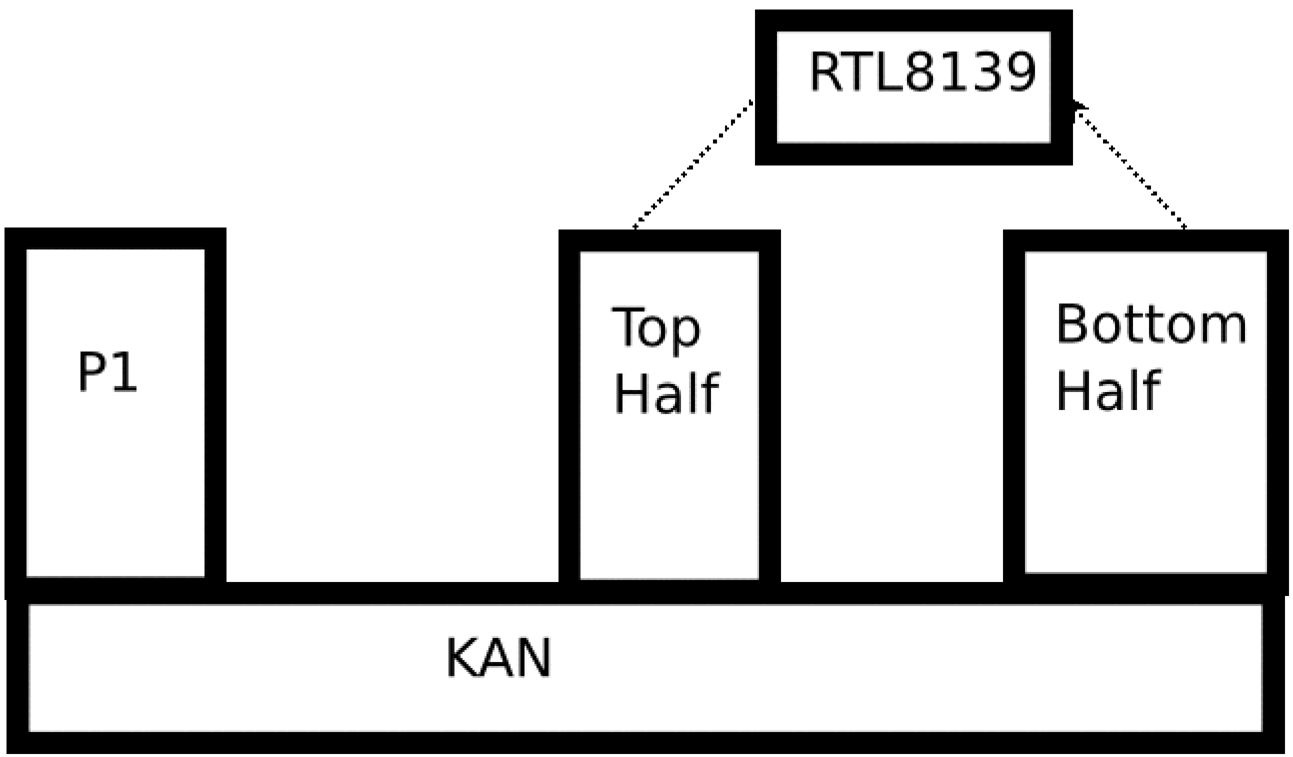
When the packet has been sent,
the NIC raises an interrupt.

Sending an Ethernet packet: 5



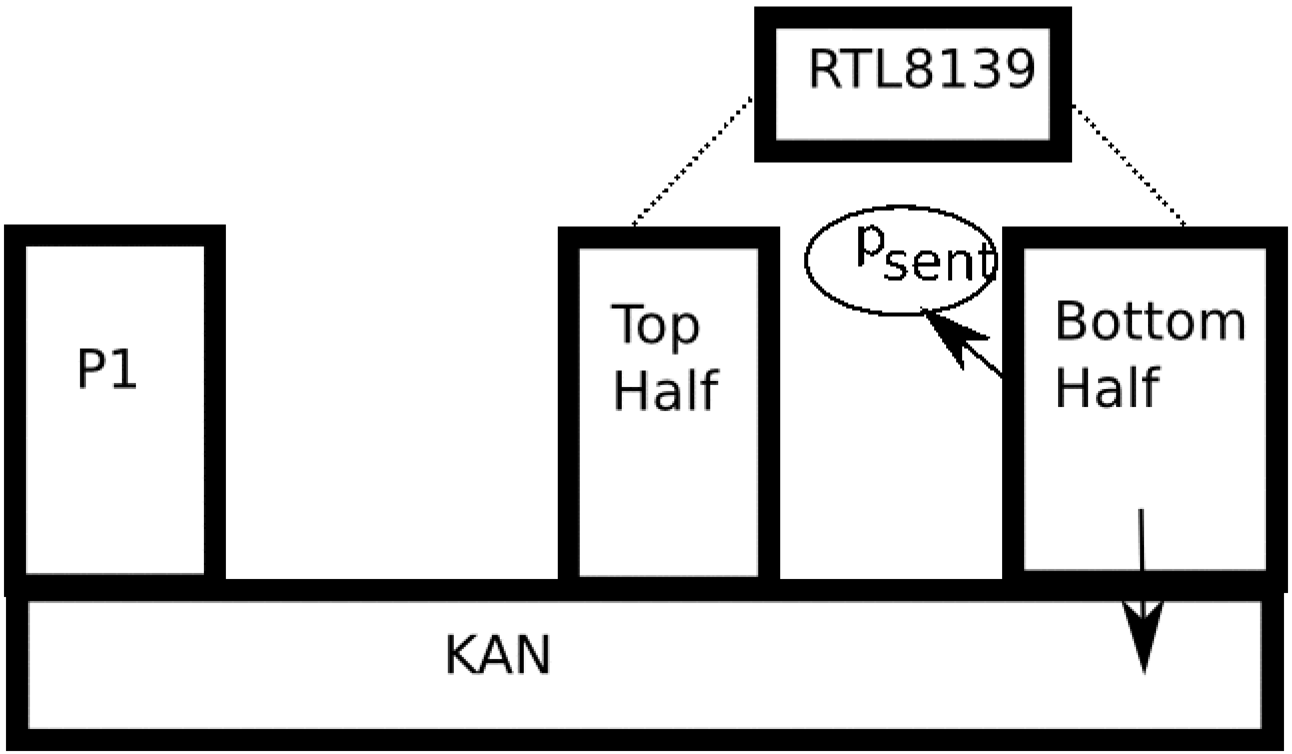
The kernel sends the bottom half an interrupt packet.

Sending an Ethernet packet: 6



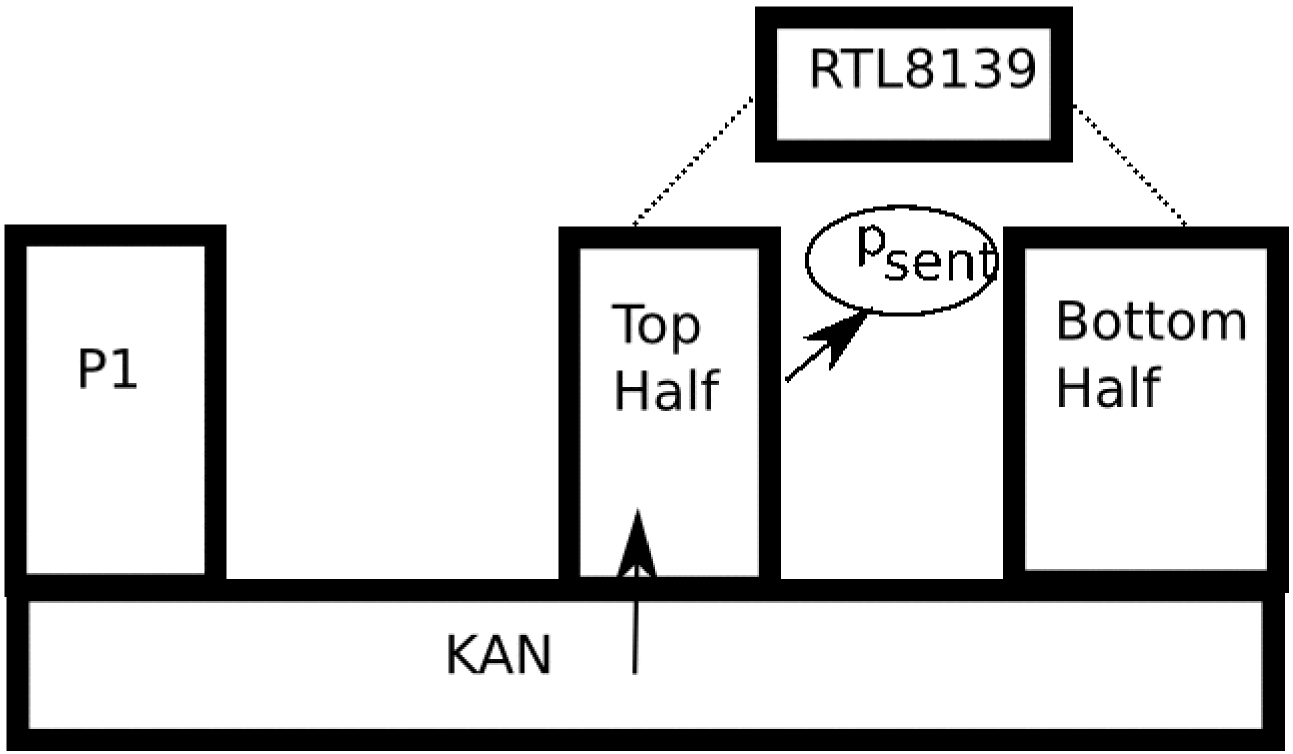
The bottom half updates the NIC data structures.

Sending an Ethernet packet: 7



The bottom half sends a packet to notify the top half.

Sending an Ethernet packet: 8



The top half updates its internal data structures.

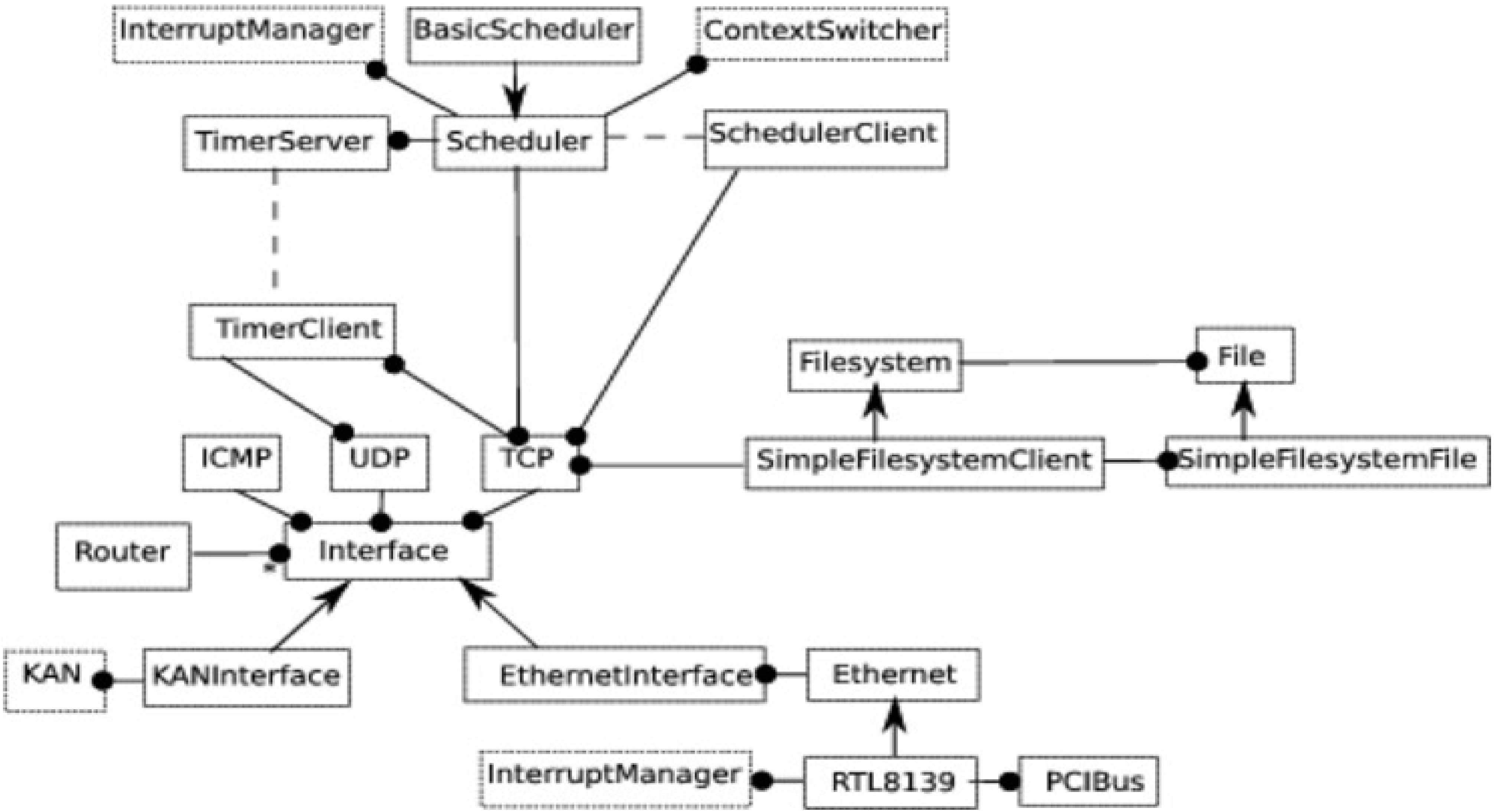
The prototype

- Started as a semester project in 91.516, running under Linux, in user space.
- Later ported to x86 PC hardware.
  - Started in real mode, moved to protected kernel.
- Adding memory protection among processes uncovered flaws in user space code.
  - Accidental use of globals spanning processes.

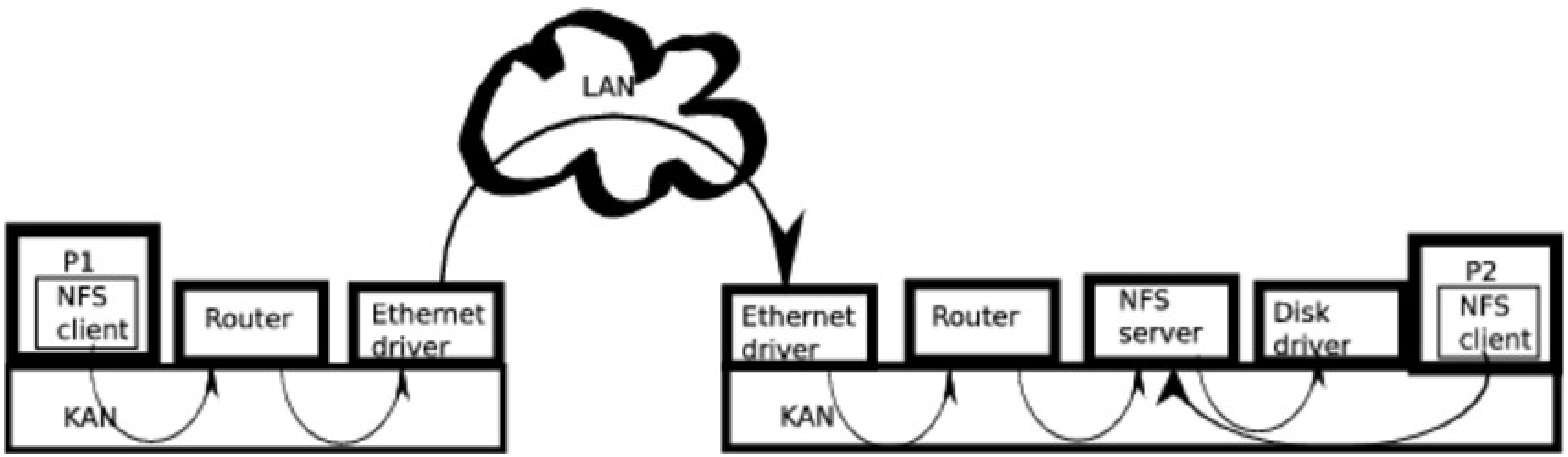
The prototype (2)

- Solving the prototype's flaws proved impractical.
- A new design is needed.
- Learned lessons from the prototype.
- These lessons will inform the new design.

The prototype: classes

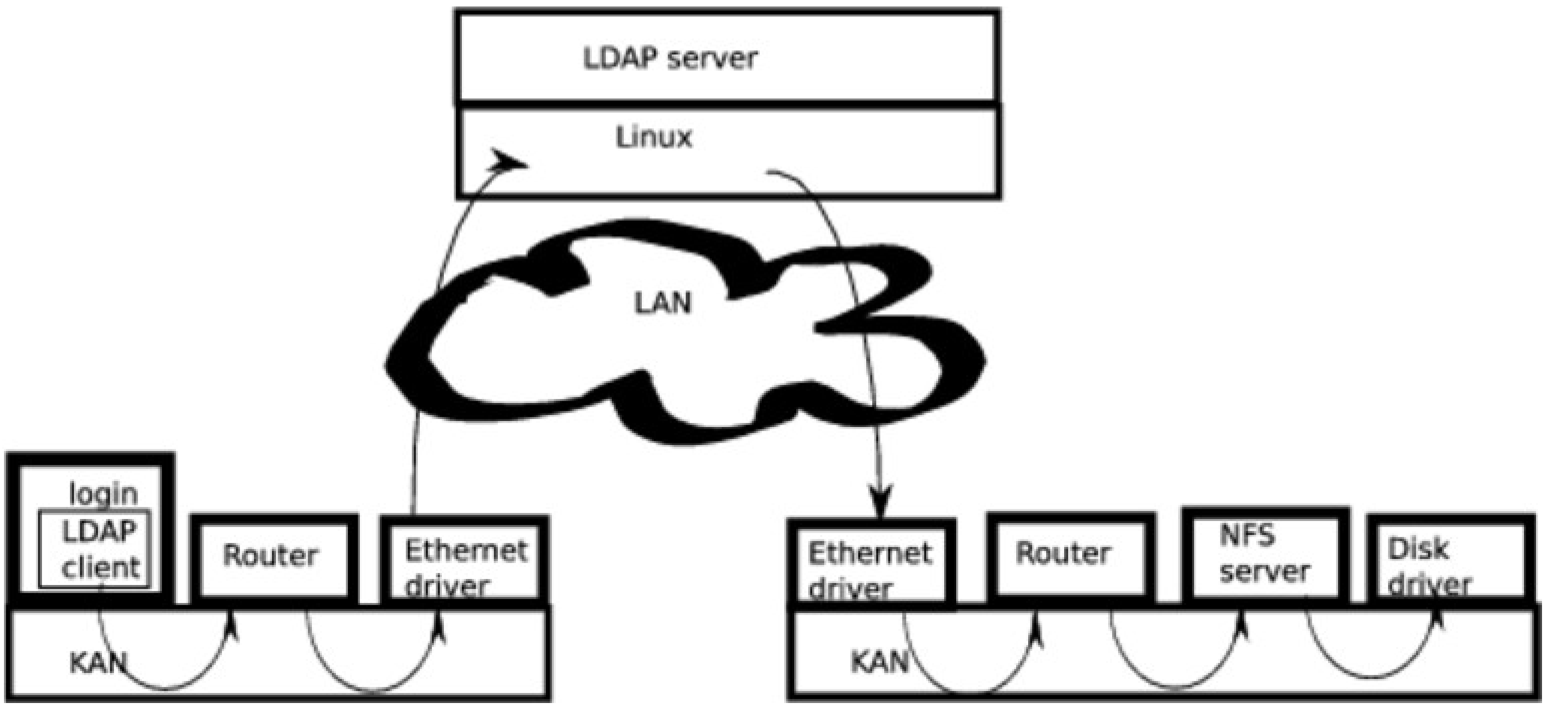


Interaction example



Two PackOS systems. The right-hand system is running an NFS server, which is being used by processes P1 and P2. Arrows show client-to-server direction.

Heterogeneous example



A heterogeneous network including two PackOS systems and one Linux system. The left-hand PackOS system is using the Linux system's LDAP server for authentication; the LDAP server is using the right-hand PackOS system's NFS server for its configuration files.

Lessons learned

- Include memory protection from the start.
- Separate user and kernel binaries.
- PackOS needs threads.
- DMA is dangerous.
- The kernel should include a clock.

Include memory protection from the start.

- Original user-space prototype could not have memory protection.
- All processes were into the same binary.
- Certain crucial libraries had state crossing process boundaries.
- Once memory protection was added, large amounts of user-space code needed to be rewritten.

Separate user and kernel binaries.

- In the prototype, all is in the same executable.
- Reasonable for original user-space implementation.
- A problem in protected mode: no compile-time separation between kernel code and user code.
- In the new design, PackOS should have separate binaries from the start.
  - Bootstrap in kernel, ELF in user space.

PackOS needs threads.

- In the prototype, all processes are single-threaded.
- Event-driven loop.
- Unreasonably difficult to work with.
- New design will permit multithreaded processes.

DMA is dangerous.

- A design goal: keep device drivers from touching anything but their assigned hardware.
- Not possible with most DMA-capable hardware: DMA bypasses the MMU.
- Long-standing problem, much research behind it. Requires new hardware designs.
- New design can't solve it, but should be aware of the problem.

DMA is dangerous.

- “I don't have any solution, but I certainly admire the problem.” — Ashleigh Brilliant

The kernel should include a clock.

- In general, interrupts are handled in user space.
- For the clock interrupt, this turns out to be prohibitively expensive.
- In the prototype, the scheduler handles clock interrupts.
- But other code (esp. TCP) needs ticks.

The kernel should include a clock.

- User-space library for asking the scheduler for ticks.
- Much too slow, though.
- Solution: put ticks into kernel, with reference-counted packets.
  - Reference-counted packets needed for multicast anyway.

Result: New design

- Full details of the new design are in my thesis.
- A summary of the interesting decisions:
  - Threads.
  - IPv6 interface objects.
  - Per-process filesystems.
  - Service discovery.
    - Requires multicast.

New design: Threads

- Kernel provides context switching and packet delivery.
- Many-to-many relationship between contexts and KAN endpoints will permit threading.
- Useful for implementing TCP: a separate thread can handle all TCP traffic and deliver results to other threads in same process.

New design: Threads (2)

- Will require in-process synchronization primitives.
- Don't want to add them to the kernel; don't want to incur latency of round trip to a lock server.
- Can be implemented via atomic operations, plus the ability to yield to another thread.

New design: IPv6 interface objects

- An interface is an object to send and receive packets.
- Subclasses present in the prototype: KAN interface, Ethernet interface.
- Most processes have just one interface, for the KAN.
- Routers have two or more interfaces.

New design: Per-process filesystems.

- Most filesystems accessed over the KAN.
  - Each process has its own filesystem clients.
- No reason all the processes have to access the same file servers.
- Similar to Plan 9.
  - Possibly at a finer grain, though: different code in same process might access different file servers.

New design: Service discovery.

- Based on DNS SRV records.
- Want to find a local server that offers filesystem X? Ask for corresponding SRV record.
- Probably via multicast DNS (aka zeroconf, Rendezvous).
  - Prototype doesn't have multicast, so...

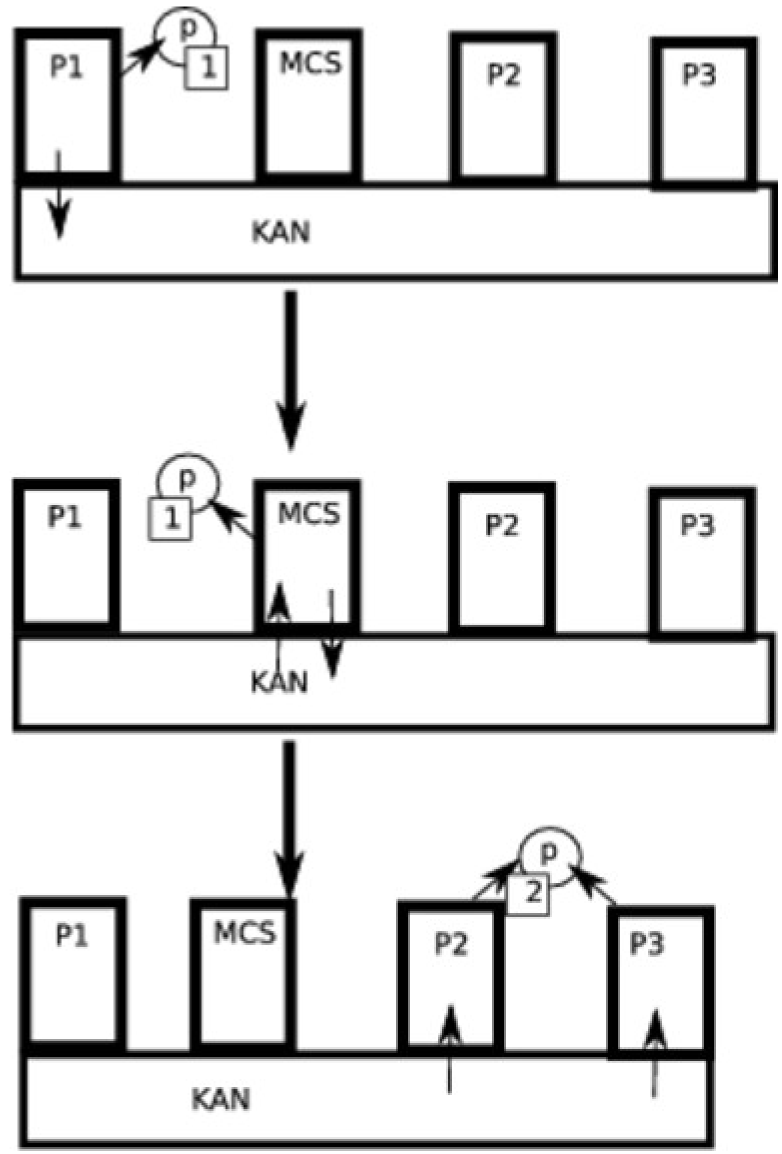
New design: Multicast

- Various possibilities.
- Most of them involve giving up the O(1) guarantee and/or zero-copy networking.
- Two remaining options:
  - Multicast server.
  - Multicast KAN endpoints.
- Both require reference counting on the packets.

New design: Multicast server

- Processes would talk to the multicast server, asking to join and leave multicast groups.
- To send a multicast packet, send it to the server.
  - Server address at link layer, group address at IP layer.
- Server forwards.
- Disadvantages:
  - Latency.
  - Single point of failure.

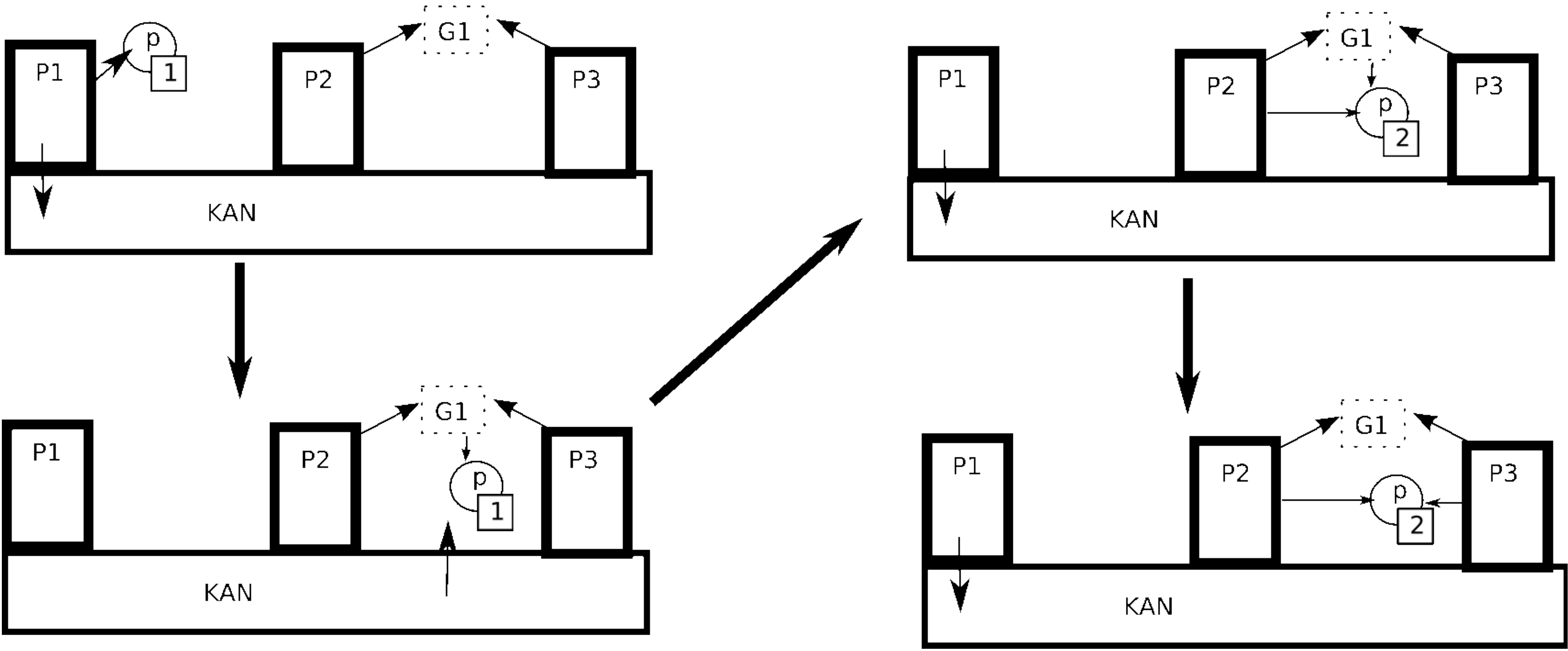
New design: Multicast server



New design: Multicast KAN endpoints

- Requires kernel support.
- Join/leave groups by asking the kernel.
- Any group with members has a KAN endpoint.
  - Circular buffer of packets for the group.
- If endpoint X is a member of group G, then receiving on X checks G's queue first.

Multicast KAN endpoints



Conclusion

- The prototype was a limited success.
- Functioning OS:
  - TCP/IPv6
  - Ethernet
  - HTTP server
- Provided plenty of experience for version 2.

Future work

- Process migration.
- POSIX support.
- Hardware support.
- Performance comparisons.
- Flexibility exploration.