- XDP
  - 引言
    - 研究背景
      - 网络流量增长与数据包处理挑战
        - 通用os网络栈花在每个包的指令多
        - 专门包处理工具完全绕过内核独占cpu
      - 现有解决方案内核旁路的局限性
        - 难集成
        - 成熟模块在用户态再实现，常见工具不可用
        - 破坏内核边界
      - eXpress Data Path的提出与意义
        - 网络栈添加可编程能力
          - 兼容现有os，复用已有网络设施
          - 在eBPF指令虚拟机中，是一个受限环境
          - 内核接触包数据前就能做一些自定义包处理
          - 加载时静态校验，动态生成原生机器指令
        - 优点
          - 硬件控制权完全留在内核
          - 无需硬件特性，复用内核网络栈
          - 接口稳定，无用户态到内核态切换的昂贵开销
  - 相关工作
    - 操作系统内核数据包处理现状 — 现有内核数据包处理机制 — Kernel Bypass（内核旁路）是绕过Linux内核（TCPIP协议栈）的技术，不使用Linux内核子系统的功能，采用自己实现的相同功能的代码来处理，从用户空间直接访问和控制设备内存，避免数据从设备拷贝到内核，再从内核拷贝到用户空间。
    - 可编程数据包处理研究
      - 用户态 — DPDK 也称作内核旁路框架（kernel bypass framework）， 因为它将网络硬件的控制权从内核转移到了用户态的网络应用，完全避免了内核-用户态 之间的切换开销。
      - 内核 — 以内核模块（kernel module）方式实现包处理功能代价非常高， — 程序执行出错时可能会导致整个系统崩溃
      - 可编程硬件 — NetFPGA [32]，通过对它暴露的 API 进行编程，能够在这种基于 FPGA 的专 用设备上运行任何包处理任务。
  - 系统设计
    - eXpress Data Path架构设计
      - XDP driver hook
        - 在网络设备驱动中执行，每收到一个包，程序就执行一次
        - 在网卡收到包之后最早能处理包的位置执行
        - 流程
          - 提取包头中的信息
          - 读取或更新一些资源的元信息
          - 程序能修改包数据的任何部分，包括添加或删除包头
          - 结果
            - 丢
            - 发
            - 重定向
            - 放行，进入内核网络栈
      - eBPF virtual machine
        - BPF 使用 基于寄存器的（register-based） virtual machine 来描述 过滤动作（filtering actions）
        - 内核管理所有 BPF 程序的生命周期。
      - BPF maps — 不同 BPF 程序之间、BPF 程序与用户空间应用之间能够通信 — 程序每次执行时，初始状态都是相同的（即程序是无状态的），它们无法直接访问内核中的持久存储（BPF map）
      - eBPF verifier
        - 加载入口：bpf() 系统调用 — 确保程序不包含任何可能会破坏内核的操作
        - two-pass DAG
          - 一次深度优先搜索（depth-first search），以确保它是无环
          - 再扫描一遍，这次会遍历 DAG 的所有可能路径 — 确保程序的内存访问都是安全的
    - 性能优化策略
      - 与DPDK
        - 直接弃包（packet drop）
          - XDP 基准性能是 24Mpps/core，DPDK 是 43.5Mpps/core
          - 全局性能受限于 PCI 总线，启用 PCI descriptor compression（在 CPU cycles 和 PCI 总线带宽之间取舍）之后，能达到 115Mpps。
          - conntrack 模式达到了 1.8Mpps/core，raw 模式是 4.8Mpps/core ； 这两种模式均未达到硬件瓶颈。 最终的性能，XDP 比常规网络栈的最快方式快了 5 倍。
        - CPU使用
          - DPDK 是 busy poll 模式，因此 CPU 永远是 100%。
          - XDP 和 Linux 内核网络栈都是随流量平滑增长：前面一小段是非线性的，后面基本是线性的
        - 包转发
        - 转发吞吐 — 性能随 CPU 数量线性扩展，直到达到全局性能瓶颈。 XDP 在同网卡转发的性能远高于 DPDK 异网卡性能，原因是内存处理方式不同
        - 转发延迟 — 高 pps 场景下，XDP 的延迟已经接近 DPDK。但在低 pps 场景下，XDP 延迟比 DPDK 大的多，原因是 XDP 是基于中断的，中断处理时间（ interrupt processing time）此时占大头；而 DPDK 是轮询模式，延迟相对比较固定。
      - 差距所在
        - XDP 未做底层代码优化
        - 将内核中与测试网卡无关的 DMA 函数调用删掉，单核性能提升
      - 性能提升
        - 驱动代码
        - 删除核心 XDP 代码中的非必要操作
        - 通过批处理平摊处理开销