Programmable Dataplane for Next Generation Networks

NETLAB NETWORKED SYSTEMS RESEARCH LABORATORY
University of Glasgow | School of Computing Science

Simon Jouet, Richard Cziva, Dimitrios Pezaros
simon.jouet@glasgow.ac.uk richard.cziva@glasgow.ac.uk dimitrios.pezaros@glasgow.ac.uk
https://netlab.dcs.gla.ac.uk

INTRODUCTION

Today's network programmability mainly provided by OpenFlow:

- Controller is logically-centralized
- Abstraction over a fix set of dataplane functions (packet matching, counters)
- Simplifies Network Management, Evolution and Innovation
    - Lots of work in Routing, Traffic Engineering, Quality of Service enforcement (QoS) or Network Virtualization have leveraged OpenFlow.

OpenFlow is the *defacto* Implementation of Software Defined Networking but only marginally covers the SDN concept:

- Programmability is very limited to a fixed set of features
    - supported protocols, statistics, actions …
- Dataplane still remains a blackbox
    - Upcoming and custom network protocols (SCTP, QUIC, VxLAN …)
    - Custom Routing, Queuing, Load-Balancing, QoS enforcement
    - Custom Statistics and Notifications for telemetry

APPROACH

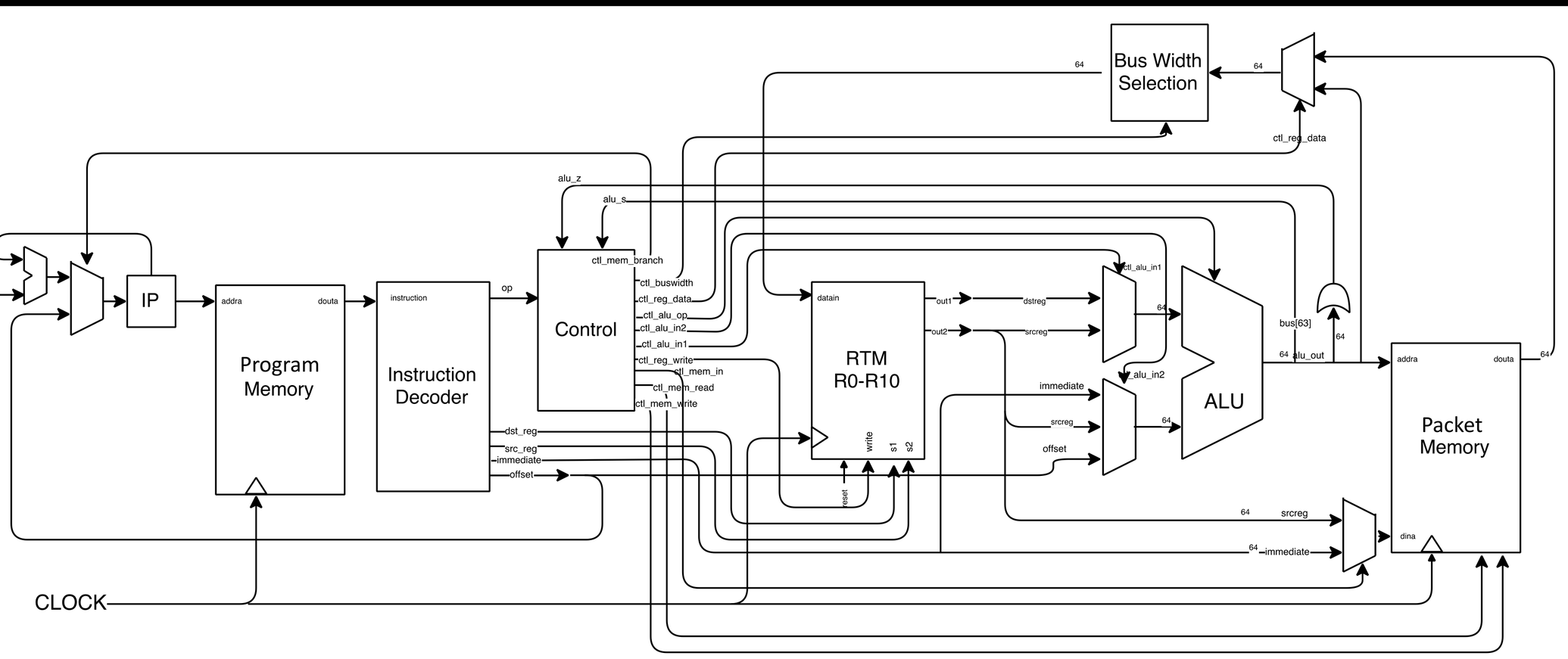
Platform and Protocol-Independent Instruction set:

- The (e)BPF instruction set has been widely used for packet processing, filtering, and classification
    - Linux Kernel, FreeBSD, TCPdump/LibPCAP, Wireshark …
- Platform-independent RISC-like instruction set designed specifically to be protocol independent and used for real-time packet processing with fixed time constraints by disabling loops.
- Language support for hashtable lookup, adding support for TCAM an LPM table lookup is under discussion.

Move away from the typical match-action pipeline:

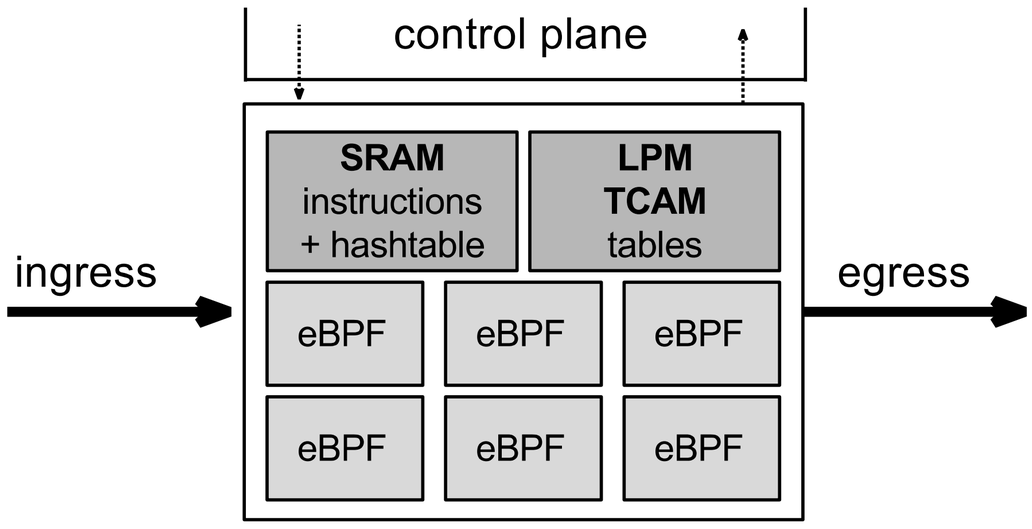
- Replace linear, multi-stage match-action pipeline with many instances of eBPF execution units; high programmability at high aggregate throughput
- Use a higher-level programming language to express dataplane behaviour
    - P4 to eBPF compiler
    - Restricted C to eBPF compiler

eBPF CORE DESIGN



Verilog Implementation of an eBPF execution unit

eBPF SWITCH DESIGN



Verilog Implementation of an eBPF execution unit

Performance limited by the clock speed

- 64bits bus, 1 clock cycle per instruction (except jump and memory read)
- Estimate FPGA clock speed ~250MHz, hence 4ns per instruction
- Complex BPF program of 100 instructions takes 400ns to execute
- A single core can handle 1.28Gbps of 64B back-to-back frames and 30Gbps MSS-sized packets.

FUTURE WORK

Example use-cases:

- Line-rate anomaly detection, SYN/FIN ratio, EWMA
- Telemetry, alerts when buffer occupancy or counters reach a threshold

NetFPGA 10G Implementation:

- Evaluate space and speed constraints of a single eBPF core on the NetFPGA with MAC/PHY IP, ingress and egress queues and TCAM/SRAM memory
- eBPF requires random access to packet memory, added complexity over reference router implementation using AXI4 stream
- Many-eBPF core implementation with input queue arbiter into a single packet queue, load sharing against many-cores and queue to egress.
- Consistent read/write from multiple cores to the shared SRAM and TCAM

REFERENCES

[1] S. Jouet, R. Cziva, and D. Pezaros, "Arbitrary Packet Matching in OpenFlow," in IEEE HPSR '15
[2] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming Protocol-independent Packet Processors," SIGCOMM '14
[3] IOVisor, "BPF-to-P4 compiler frontend," github.com/iovisor/.