Netcope P4 for Intel PAC N3000
Netcope Technologies

- Primary focus on FPGA-based dataplane acceleration in P4 language
  - P4 programmability for Intel FPGAs
- Sites
  - San Jose, California
  - Brno, Czechia
- FPGA Expertise
  - Over 15 years
  - Close cooperation with Intel
Become **flexible** with Netcope P4 - paradigm shift in FPGA programming.
P4 Gains Broad Networking Industry Adoption, Joins Open Networking Foundation (ONF) and Linux Foundation (LF) to Accelerate Next Phase of Growth and Innovation

P4 continues to gain rapid adoption and support across the networking industry. P4.org currently has more than 100 member organizations spanning industry and academia and continues to grow, adding new members and developers.

“P4 and programmable forwarding planes are examples of cutting-edge technologies we are using in our new network architecture design.”

Tom Bie
VP Technology & Engineering

“Our whole networking industry stands to benefit from a language like P4 that unambiguously specifies forwarding behaviour, with dividends paid in software developer productivity, hardware interoperability, and furthering of open systems and customer choice.”

Tom Edsall
SVP and CTO
Data Center Networking

“As one of the creators of P4, Google is proud to see the rapid adoption of P4 across the networking industry.”

Amin Vahdat
Engineering Fellow, Vice President and Technical Lead for Networking

P4 Language

Domain specific language specialised in network data forwarding

- **Flexible protocol stack** - Independence of network protocols. Traffic processing described with a custom program
- **Target Independence** - Suitable for various architectures such as SW, ASICs, NPUs, FPGAs
- **Field reconfigurability** - Change of behaviour after deployment

P4 Language Benefits

- Lowering entry barriers for new player in the industry
- Easy to use
- Suitable for network architects and network engineers
Netcope P4

Netcope P4 Cloud is a web service that takes P4 code (packet processing program) as an input and generates two possible outputs:

- **Bitstream** - For a set of supported platforms customer can directly compile final bitstream, e.g. Intel PAC N3000 acceleration card.

- **Netlist** - This output enables the customer to customize the design by adding additional 3rd party IP before final bitstream compilation. Requires customer or 3rd party to supply board support package.
Netcope P4 solves network performance bottlenecks

FPGAs have the potential to be programmed to suit needs of network engineers, Netcope P4 enables this.

**Growth in the use of FPGAs**

- FPGAs becoming standard due to flexibility and performance. Difficult to program.
- FPGA-based smart NICs are becoming commodity.
- Large need by mass market for easy to use development tools.
- Netcope has partnerships with key smart NIC vendors.
- Netcope P4 brings the power of FPGAs to network engineers.

**Using Netcope to program FPGAs**

- Netcope P4 allows FPGAs to be programmed using the open standard P4 language.
- Enables network architects / admins to manage network data plane.
- Cloud-based tool to generate binary configuration from P4 to FPGA.
- Shortens the development cycle from months to days.
Netcope P4 - Product Overview

Netcope P4 unlocks new applications and use cases for FPGAs at the telco edge

- Highly efficient approach
- Ready to run on different hardware targets
- Hand-optimized RTL
- FPGA firmware
- Firmware module
- Extensible IP delivery platform
- 3rd party IP
- 256 or 512 bit data width at 200 MHz (40 or 100 Gbps)
- Single packet per clock cycle

Web GUI
User P4 code input

Netcope P4 Cloud
Manage tasks
- Tasks List
- Tasks in progress
- Completed tasks
- In progress tasks
- Ready tasks
- Exit

Product Overview
Netcope P4 unlocks new applications and use cases for FPGAs at the telco edge
Netcope P4 - Value proposition for SRv6

P4 development cycle can be leveraged to cut costs and improve speed

HDL Development Cycle (52 days, 10k+ lines of code*)

- Problem analysis: 1 day
- Code Ethernet and IPv6 parser: 10 days
- Code very simple match table: 6 days
- Code single-purpose packet editor: 8 days
- Simulation: 5 days
- Build firmware, meet timing: 5 days
- Code API: 6 days
- Performance and conformance testing in the hw: 1 day
- Bug Fixing: 10 days

<table>
<thead>
<tr>
<th>Availability</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ $ $</td>
<td>HDL developers are rare/difficult to find and more expensive</td>
</tr>
</tbody>
</table>

P4 Development Cycle (5 days, 390 lines of code*)

- Problem analysis: 1 day
- Code and test P4 code: 1 day
- Build firmware from P4: 4 hours (1 day)
- Performance and conformance testing in the hardware: 1 day
- Bug Fixing: 1 day

<table>
<thead>
<tr>
<th>Availability</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ $ $</td>
<td>There are more software developers available and for a lower cost</td>
</tr>
</tbody>
</table>

Total 5 days, >10x improvement
Extensible code, 390 lines, >80x improvement

(* 36 185 lines of VHDL generated by Netcope P4 compiler internally, not including existing library of standard modules, FIFOs etc.)
## Netcope P4 availability

<table>
<thead>
<tr>
<th>Bitstream</th>
<th>Netlists (IP core)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel PAC N3000</td>
<td>Intel FPGA Chips</td>
</tr>
<tr>
<td>Intel Arria 10</td>
<td>Arria 10</td>
</tr>
<tr>
<td>4x25G, 2x2x25G, 8x10G</td>
<td>Stratix 10</td>
</tr>
</tbody>
</table>
Intel PAC N3000

- First Intel Branded PAC card for NFV
- Intel built and validated
- 8x10G / 2x2x25G / 4x25G Network
- PCIe Gen3x16 Host
- 75W Estimated, passive cooled
- FHHL, single slot
- 8GB DDR, 144Mb QDR
- 2 x 40G Fortville NICs
**Netcope P4 - segments and use-cases**

- **Telco Gateways - vBNG and vEPC / 5GC**
  - A core of wired / wireless mobile network
  - Trend in separation of control plane and data plane
  - Data plane accelerated by FPGAs to serve more subscribers

- **NFVi - Segment Routing**
  - A source-routing architecture that seeks the right balance between distributed intelligence and centralized optimization
  - Enables high resilience, low latency, scalability and centralized traffic engineering
  - Segment routing function offload to allow CPU core utilisation by virtual functions
vBNG and vEPC on Intel PAC N3000

- Dataplane functions written in P4 language
  - Components of vBNG / vEPC dataplane are offloaded to Intel PAC N3000
    - Encapsulation / Decapsulation (GTP, PPPoE, VLAN, MPLS)
    - Packet Filtering, QoS enforcements, Flow based charging
    - Firewalling, NAT, DPI

- Generating bitstream for Intel PAC N3000
  - vBNG / vEPC components described in P4 language are automatically converted into bitstream for Intel PAC N3000 by Netcope P4 Cloud
  - No need for FPGA knowledge or FPGA integration
  - Significant shortening of development process
  - Easy customization of supported protocol stack or other features
5G Mobile Network

Packet manipulation with Netcope P4

- GTP-U encapsulation
- Packet filtering, QoS enforcement, flow-based charging
- TCP/UDP checksum
- Intel HQoS integration
- Parental control, video & web optimization
- Firewalling, NAT, DPI

The diagram covers only user plane (payload), i.e. data path, which uses NG3 and NG6 interfaces, as opposed to signalling interfaces.

Bearer = IP packet flow with a defined QoS between the gateway and the UE
vBNG pipeline: it’s open source

- Open Source implementations of vBNG in P4 already exist
  - [https://github.com/opencord/p4se](https://github.com/opencord/p4se)

- More than just vBNG
  - ~30 Match Action Tables
  - ~25 Counter arrays
  - Many #ifdefs (INT, IPv6, SPGW, VRF, ...)

- Does not cover features outside P4 language spec (QoS)
  - Set as output metadata
Results (focused at P4 vBNG on FPGA)

Features
- Input filtering (VLAN)
- BNG encap/decap (IPoE)
- ACL
- Forwarding (incl. MPLS, VLAN)

Throughput
- 100 Gbps (Intel N3000 platform)
  - 80 Gbps (incl. Intel HQoS)

What is limited by what?
FPGA logic
- Number of tables
- Ternary table size/key width

FPGA memory
- LPM/Exact table size
- Counters

Off-chip memory
- Exact table size
- Counters
- Throughput
Leveraging Intel PAC N3000 assets

- Using external on board components from P4 pipeline to create complex and high performance vBNG / vEPC solutions
  - Using External DDR4 Memories
    - To support large number of subscribers
  - Integration of Intel HQoS
    - Possibility to use 3rd Party IP cores
  - Providing access to custom search engines
    - Optimised algorithms (not covered by P4 specification)
  - Feature utilisation of Intel XL710 chip
    - DPDK, Virtual Functions, Filtering
Segment Routing (SRv6) on Intel PAC N3000

- SRv6 functions written in P4 language
  - Actions performed in SRv6 function are mapped to constructs of P4 language
    - Parsing of IPv6 extension headers (segments) covered by P4 parser
    - Encapsulation / Decapsulation covered by P4 functions for adding / removing headers
    - Packet forwarding covered by P4 match & action tables

- Generating bitstream for Intel PAC N3000
  - SRv6 functions described in P4 language are automatically converted into bitstream for Intel PAC N3000 by Netcope P4 Cloud
  - No need for FPGA knowledge or FPGA integration
  - Significant shortening of development process
Segment Routing (SRv6) on Intel PAC N3000

- Netcope P4 enables SR(v6) in dataplane
  - Netcope P4 is a framework for customization of packet forwarding plane with high level language for FPGA-based products. It targets Intel’s FPGA-accelerated network adapters.
  - Packet processing speed of FPGA (up to 100 Gbps line rate)

- The Netcope P4 consists of
  - High level language (P4) to FPGA compiler
    - Pre-build high-performance components
  - C language library (API) + software tool for configuration
  - Documentation (sample applications included)
// Rewrites destination IPv6 address
action rewrite() {
    // Rewrite the destination IPv6 address with the last IPv6 segment
    modify_field(ipv6.dstAddr, lastSeg.segVal);
    add_to_field(ipv6_ext.next_seg, -1);
}

// Only default rule with action rewrite
table tab_rewrite {
    actions {
        rewrite;
    }
}

#define PROTOCOL_IPV6 0x86dd
#define PROTOCOL_V6EXT 0x2B
header ethernet_t ethernet_0;
header ipv6_t ipv6;
// Ethernet parsing
parser parse_ethernet {
    extract(ethernet_0);
    return select(latest.etherType) {
        PROTOCOL_IPV6 : parse_ipv6;
        default : ingress;
    }
}

// IPv6 parsing
parser parse_ipv6 {
    extract(ipv6);
    return select(latest.nextHead) {
        PROTOCOL_V6EXT : parse_ext;
        default : ingress;
    }
}
SRv6 implementation (P4 vs. HDL)

HDL Development Cycle (52 days, 10k+ lines of code*)

Netcope P4 Development Cycle (5 days, 370 lines of code*)

Total 5 days, >10x improvement
Extensible code, 390 lines, >80x improvement

(* 36 185 lines of VHDL generated by Netcope P4 compiler internally, not including existing library of standard modules, FIFOs etc.)
Thank you for your attention.

Q&As

Find us.

www.netcope.com