Flightplan: Dataplane Disaggregation and Placement for P4 programs

Nik Sultana
Illinois Tech
http://www.cs.iit.edu/~nsultana1

COIN Interim // 10th Feb 2022
<table>
<thead>
<tr>
<th>Name</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shivani Burad</td>
<td>Heena Nagda</td>
</tr>
<tr>
<td>Anirudh Chelluri</td>
<td>Rakesh Nagda</td>
</tr>
<tr>
<td>André DeHon</td>
<td>Isaac Pedisich</td>
</tr>
<tr>
<td>Hans Giesen</td>
<td>Alexander Poylisher</td>
</tr>
<tr>
<td>Zhaoyang Han</td>
<td>Nishanth Prabhu</td>
</tr>
<tr>
<td>Latha A. Kant</td>
<td>Lei Shi</td>
</tr>
<tr>
<td>Boon Thau Loo</td>
<td>Nishanth Shyamkumar</td>
</tr>
<tr>
<td>Tony McAuley</td>
<td>John Sonchack</td>
</tr>
</tbody>
</table>
Dataplane Programmability
& In-Network Programming
Dataplane Programmability & In-Network Programming

(P4 Language Consortium)

(DPDK (Linux Foundation))

(Netronome)

(Netcope)
Dataplane Programmability
Dataplane Programmability

Current paradigm: dataplane program → one dataplane (device)
Dataplane Programmability

Current paradigm: dataplane program $\rightarrow$ one dataplane (device)

- Conceptually simple.
- Familiar from programming other devices.
- Mismatch with Software-Defined Networking (the overarching vision).
Current paradigm: dataplane program → one dataplane

Mismatch with Software-Defined Networking

- **Target’s resources are dedicated to the program.**
  Inefficient use of individual target resources.

- **Program must entirely execute on a single target.**
  Unnecessary constraints of program functionality.

- **We’re meant to be “programming the network”.”**
  Programmability is scoped too conservatively.
**Current paradigm:** dataplane program \(\rightarrow\) one dataplane

**New paradigm:** dataplane program \(\rightarrow\) suitable mix of dataplanes
Current paradigm: dataplane program → one dataplane

New paradigm: dataplane program → suitable mix of dataplanes

- Resource-based program decomposition.
- Split into set of 1-1 dataplane programs.
Current paradigm: dataplane program → one dataplane

New paradigm: dataplane program → suitable mix of dataplanes

- Resource-based program decomposition.
- Split into set of 1-1 dataplane programs.

Uses existing vendor toolchains, language & hardware.
Example: “Crosspod” Program
Example: “Crosspod” Program
Hand-over + monitoring
**Current paradigm:** dataplane program ➔ one dataplane

**New paradigm:** Dataplane Disaggregation

- dataplane program ➔ suitable mix of dataplanes

- Resource-based program decomposition.
- Split into set of 1-to-1 dataplane programs.
Dataplane Disaggregation

( ≠ Server Disaggregation)
( ≠ Switch Disaggregation)

Virtual Dataplane $\rightarrow$ Set of Physical Dataplanes

“One big switch” $\rightarrow$ “One big programmable switch”
Current paradigm: dataplane program → one dataplane

**How to implement?**

Automated support needed

**Dataplane Disaggregation:**
 dataplane program → suitable mix of dataplanes

- Resource-based program decomposition.
- Split into set of 1-1 dataplane programs.

✅ Uses existing vendor toolchains, language & hardware.

❌ If manual: laborious, error-prone, hard to change.
Current paradigm: dataplane program → one dataplane

Dataplane Disaggregation:
 dataplane program → suitable mix of dataplanes

- Analyze program’s use of resources.
- Split into set of 1-1 dataplane programs.
Current paradigm: dataplane program → one dataplane

Dataplane Disaggregation:

- Analyze program’s use of resources.
- Split into set of 1-1 dataplane programs.
- Exploit heterogeneous resources.
- Meet resource & performance objectives.
Current paradigm: dataplane program → one dataplane

Dataplane Disaggregation:
 dataplane program → suitable mix of dataplanes

- Exploit heterogeneous resources.
- Meet resource & performance objectives.

Placement

- Analyze program’s use of resources.
- Split into set of 1-1 dataplane programs.
Current paradigm: dataplane program → one dataplane

Dataplane Disaggregation:
dataplane program → suitable mix of dataplanes

- Exploit heterogeneous resources.
- Meet resource & performance objectives.
- Place
- Hand-over control between dataplanes.
- Synchronize state.
- Detect and handle faults.

- Analyze program’s use of resources.
- Split into set of 1-1 dataplane programs.
Current paradigm: dataplane program → one dataplane

Dataplane Disaggregation:
 dataplane program → suitable mix of dataplanes

- Exploit heterogeneous resources.
- Meet resource & performance objectives.

- Analyze program’s use of resources.
- Split into set of 1-1 dataplane programs.

- Hand-over control between dataplanes.
- Synchronize state.
- Detect and handle faults.

- No language or hardware changes.
- Changing programs, topology, and hardware.
- Ingest various data: program, topology, resource information, constraints, objectives.
- Scoping network programming and operation.
- Consider and explain multiple possible solutions.
- Multiple programs in same network.
- Diagnosis and debugging.

Placement

Exploit heterogeneous resources.
Meet resource & performance objectives.
Flightplan
Dataplane Disaggregation:
dataplane program \rightarrow \text{suitable mix of dataplanes}

- Analyze program’s use of resources.
- Split into set of 1-1 dataplane programs.
- Exploit heterogeneous resources.
- Meet resource & performance objectives.
- Hand-over control between dataplanes.
- Synchronize state.
- Detect and handle faults.

Placement

- No language or hardware changes.
- Changing programs, topology, and hardware.
- Ingest various data: program, topology, resource information, constraints, objectives.
- Scoping network programming and operation.
- Consider and explain multiple possible solutions.
- Multiple programs in same network.
- Diagnosis and debugging.

Code, tests, scripts, data, documentation: https://flightplan.cis.upenn.edu/
Flightplan
Flightplan
Flightplan
Idea from symbolic AI: rule-based search.

**Flightplan**
Flightplan
Example program (Crosspod)

```
bit<1> compressed_link = 0;
bit<1> run_fec_egress = 0;
...
Flyto(Compress);
// If heading out on a multiplexed link, then header compress.
egress_compression.apply(meta.egress_spec, compressed_link);
if (compressed_link == 1) { // If compressed, then...
    header_compress(forward);
    if (forward == 0) { // Drop if not forward.
        return;
    }
}
Flyto(FEC_Encode);
check_run_FEC_egress.apply();
// If heading out on a lossy link, then FEC encode.
if (run_fec_egress == 1) {
    ...
    classification.apply(hdr, proto_and_port); // Sets hdr.fec.isValid()
    if (hdr.fec.isValid()) {
        encoder_params.apply(hdr.fec.traffic_class, k, h);
        update_fec_state(hdr.fec.traffic_class, k, h,
                          hdr.fec.block_index, hdr.fec.packet_index);
        hdr.fec.orig_ether_type = hdr.eth.type;
        FEC_ENCODE(hdr.fec, k, h);
        ...
```
Abstract program

```c
flyto(Compress);
// If heading out on a multiplexed link, then header compress.
egress_compression.apply(meta.egress_spec, compressed_link);
if (compressed_link == 1) {
    header_compress(forward);
    if (forward == 0) {
        drop();
        return;
    }
}
flyto(FEC_Encode);
```

(Rule generation is fully automatic)
Flightplan
Flightplan
Abstract Resource Semantics

(Specific device + platform)

CPU Rate < $2 \times 10^8$
PacketSize > 1000
header_compress

Lat. $\Rightarrow$ Lat. + $7.4 \times 10^{-3}$
Rate $\Rightarrow$ Rate $\times \frac{189.9}{194.75}$
once Power $\Rightarrow$ Power + 150 W
once Cost $\Rightarrow$ Cost + 5

FPGA Rate < $9.5 \times 10^9$
PacketSize > 100
header_compress

Lat. $\Rightarrow$ Lat. + $6.44 \times 10^{-6}$
Rate $\Rightarrow$ Rate $\times \frac{9.15}{9.3}$
$\langle$LUTs$\rangle$ $\Rightarrow$ LUTs + 24.4%
$\langle$BRAMs$\rangle$ $\Rightarrow$ BRAMs + 54.4%
$\langle$FF$\rangle$ $\Rightarrow$ FF + 15.8%
once Power $\Rightarrow$ Power + 30 W
once Cost $\Rightarrow$ Cost + 2
In-Network Program Examples

- Layer 2+ FEC (Forward Error Correction)
- Traffic compression
- In-network caching of key-value requests
Flightplan
Runtime: Fault Detection + Handling

Two mechanisms:

- In-dataplane: +ve and -ve Acks.
- Strobes from control program.
Evaluation

• **Simulation:**
  - Scale of the network (featuring various programs)
  - Overhead
  - Disaggregation (different programs split in different ways)
  - Fail-over

• **Test-bed:**
  - Throughput, latency, power, resource utilization
  - Plan comparisons for hardware alternatives
  - Single-feature evaluation
Fig 7: Multiple Programs vs Runtimes vs Splits in same network (Simulation)

<table>
<thead>
<tr>
<th>Runtimes</th>
<th>Functions/features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full</td>
<td>ALVp4</td>
</tr>
<tr>
<td>Headerless</td>
<td>firewall.p4</td>
</tr>
<tr>
<td></td>
<td>KV cache</td>
</tr>
<tr>
<td></td>
<td>Header compress/decompress</td>
</tr>
<tr>
<td></td>
<td>FEC encode/decode</td>
</tr>
</tbody>
</table>
Fig 7: Multiple Programs vs Runtimes vs Splits in same network (Simulation)
Fig 7: Multiple Programs vs Runtimes vs Splits in same network (Simulation)
Flightplan demo

MSc students: Heena Nagda (GATech), Rakesh Nagda (Penn)

Other features: graphs, multimedia cues (e.g., icons, packet structure), ...

https://flightplan.cis.upenn.edu/demo

Ack: Haoxian Chen, Max Demoulin, Joel Hypolite, Pardis Pashakhanloo, Lei Shi, Nishanth Shyamkumar, Caleb Stanford, Ke Zhong