P4CEP: Towards In-Network Complex Event Processing

Thomas Kohler, Ruben Mayer, Frank Dürr, Marius Maass, Sukanya Bhowmik, and Kurt Rothermel

August 20th, 2018
Motivation – In-Network Complex Event Processing

Status quo of latency-critical Complex Event Processing (CEP):
- Operators implemented off-path in software (middlebox model)
- Inherent distribution of sources/sinks; overlay graph of operators
In-Network Complex Event Processing:

- Implement CEP within reconfigurable data plane hardware
- Using a uniform language for Data Plane Programming

**Motivation** – In-Network Complex Event Processing

Temperature sensor

Smoke detector

60°C

smoke is detected

in-situ processing on reconfigurable hardware

Fire!

Extinguisher system

no additional RTTs (latency), high throughput
Contributions

• Concepts for in-network implementation of Complex Event Processing (P4CEP)
  ◦ Rule specification language
  ◦ Compiler from rule specification to P4

• Proof-of-concept implementation of P4CEP compiler
  ◦ For programmable NICs (Netronome NFP) and bmv2
  ◦ Publicly available at http://goo.gl/MEdPvv

• Discuss experience and limitations of Data Plane Programming for stateful packet processing

• Evaluation on a programmable NIC (NFP)

• Roadmap towards a distributed in-network CEP
Complex Event Processing

- CEP operator: processes streams of incoming events \( S_i \) to detect complex events \( C_i \)

- Event specification language
  - Specifies conditions (expressions) for complex events
    - \( P_i \): predicates on values (numerical and logical operators)
    - \( O_i \): logical operators for combination of input streams (AND, OR, ...)
  - **Example**: \( \text{temperature} > 50 \text{ AND smoke_detected} \Rightarrow \text{Fire!} \)

\[
\begin{align*}
S_{i\_Temp} & : \ldots 20°C \ 18°C \ 30°C \ 35°C \ 42°C \ 55°C \ 49°C \ 63°C \ 65°C \rightarrow t \\
S_{i\_Smoke} & : \ldots \text{false} \ \text{false} \ \text{false} \ \text{false} \ \text{false} \ \text{true} \ \text{true} \ \text{true} \ \text{true} \ \text{true}
\end{align*}
\]
Complex Event Processing

- Conditions on history of events
  - Infinite input sequence is split into windows
  - Window operators: aggregation functions ($F_n$) over a window

  \[
  S_{i,\text{Temp}}: \ldots \ 20^\circ\text{C} \ 18^\circ\text{C} \ 30^\circ\text{C} \ 35^\circ\text{C} \ 42^\circ\text{C} \ 55^\circ\text{C} \ 49^\circ\text{C} \ 63^\circ\text{C} \ 65^\circ\text{C} \\
  F_n: \text{avg/min/max}
  \]

  \[
  S_{i,\text{Smoke}}: \ldots \ false \ false \ false \ false \ false \ true \ true \ true \ true \ true
  \]

  \[
  F_n: \text{count}
  \]

- Requirements on processing
  - Memory for storing (limited) event history → stateful processing
  - Processing logic for evaluation of expressions and window operators
P4CEP – System Model

Data Plane

end-system

CEP end-system
(source)

monitor

up-date

P4CEP-TARGET

P4CEP-TARGET

packets

to

complex
events

CEP end-system
(sink)

Control Plane

P4CEP Runtime Component

P4 Table Entries

State transitions

Research Group
Distributed Systems
P4CEP – Pipeline Processing

- Classification of ingress packets or events
  - Events are encoded in packet headers, leveraging P4’s flexible parser
- Co-NF processing: forwarding, other non-CEP network functions
- Sequential CEP processing (for each complex event to detect)
  1. Window operations (persisting value, window evaluation)
  2. State machine execution (pattern detection)
Rule Specification Language

- Sole input to P4CEP compiler
- Consists of
  - Definition of windows
  - Definition of complex events to detect
- Example:

```plaintext
window sample_wnd {
    size 4
    value ipv4.totalLen
}
complex_event sample_evt {
    value sum(ipv4.totalLen)
    strategy skip-till-next-match
    pattern ([ipv4.totalLen > 500] && [tcp.dstPort == 80]) ;
    ([sum(sample_wnd) > 6000] || [ipv4.protocol == 17])
}
```
Window Operators

- Supported aggregation functions ($F_n$)
  - max, min, sum, count
  - average (future work)

- Implementation
  - Ring-buffer (event values) and index-pointer stored in P4 registers
  - Register access protected by confinement in critical section
    - Preventing inconsistency effects (e.g., lost updates)
    - NFP: pre-processor pragma or C mutex library
    - P4_16: atomic control flow block
  - Evaluating aggregation functions
    - Un-rolling the iteration over the window
    - Transient metadata fields storing aggregate value, index variable, value

Definition:

```
window sample_wnd {
  size 4
  value ipv4.totalLen }
```
Complex Event Definition

Definition:
```
complex_event sampleEvt {
  value sum(ipv4.totalLen)
  strategy skip-till-next-match
  pattern ... }
```

- **Elements**
  - return value ← static expression, field reference, window aggregate
  - transition strategy ← \{skip-till-next-match, strict\}
  - pattern: P4 expression (simple or compound predicate)

- **Implementation**
  - Deterministic Finite State Machine
Pattern Detection Engine – FSM Representation

```
pattern ([ipv4.totalLen > 500] && [tcp.dstPort == 80]) ;
([sum(sample_wnd) > 6000] || [ipv4.protocol == 17])
```

- **Pattern definition**
  - Pattern of basic events (input symbol \( x \in \Sigma \))
  - Predicates \( P_x \) on field references, window aggregates
  - Composition of predicates using logical operators seq., conj., disj.

\[
C = (\Sigma, S, s_0, \delta, F)
\]
Pattern Detection Engine – Transition Table Entries

pattern ([ipv4.totalLen > 500] && [tcp.dstPort == 80]) ; ([sum(sample_wnd) > 6000] || [ipv4.protocol == 17])

• FSM transition
  ◦ Metadata fields storing current state \((q \in S)\), matched predicate \((P_x \rightarrow x)\)
  ◦ Lookup in transition table \(\delta\), persisting new state / handle complex event

<table>
<thead>
<tr>
<th>Keys</th>
<th>Values</th>
<th>Next State</th>
<th>Accept. State</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>Match (predicate ID)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>totalLen &gt; 500</td>
<td>1</td>
<td>false</td>
</tr>
<tr>
<td>0</td>
<td>dstPort == 80</td>
<td>2</td>
<td>false</td>
</tr>
<tr>
<td>1</td>
<td>dstPort == 80</td>
<td>3</td>
<td>false</td>
</tr>
<tr>
<td>2</td>
<td>totalLen &gt; 500</td>
<td>3</td>
<td>false</td>
</tr>
<tr>
<td>3</td>
<td>sum &gt; 6000</td>
<td>4</td>
<td>true</td>
</tr>
<tr>
<td>3</td>
<td>protocol == 17</td>
<td>4</td>
<td>true</td>
</tr>
</tbody>
</table>
Encountered Limitations

- **Target-dependent**
  - Synchronization of state memory access
    - additional latency

- **Language-dependent (P4)**
  - Registers cannot directly be referenced by arithmetic operators or as table keys
    - indirection over transient meta data field
  - No floating point arithmetic, no division operator
    - fixed-point arithmetic
  - No loop-construct (not even bounded loops)
    - requires manual loop-unrolling
Evaluation – Methodology

- 1 CEP pattern of 2 basic events, sum over window of varying size \( n \)
- Acquired metrics: target’s processing latency and throughput
Evaluation – Results

- **NFP-C:**
  - $9.8 \mu s \leq l_p \leq 29.5 \mu s$
  - $56\% \geq B_p \geq 16\%$
  - $l_p$ scales linearly with window size $n$ ($n \leq 1000$)

- **bmv2:**
  - $512 \mu s \leq l_p \leq 10,000 \mu s$
  - $B_p \approx 0.05\%$
Conclusion

- Introduced to Complex Event Processing and requirements on processing

- Presented our in-network implementation of Complex Event Processing (P4CEP)

- Discussed encountered limitations of Data Plane Programming for stateful packet processing

- Shown P4CEP’s practicability on a programmable NIC target
  - Microsecond / million messages per second scales
Future Work – Roadmap to Distributed In-Network CEP

- **Placement of operators**
  - According to complexity of processing
    - End-systems: smart-NIC HW, SW (kernel), SW (user space)
  - Disaggregation of event detection (replication / partitioning of events)

- **Pre-processing** (content-based in-network filtering of events)
Thanks for your attention

Any Questions?

Contact & further information:

https://goo.gl/tYWSgW
Related Work

• In-network Computing (Use Cases)
  ◦ Dang et al., *NetPaxos: Consensus at Network Speed*, SOSR ’15
  ◦ Liu et al., *IncBricks: Toward In-Network Computation with an In-Network Cache*, ASPLOS ’17
  ◦ Sapio et al., *In-Network Computation is a Dumb Idea Whose Time Has Come*, HotNets-XVI, 2017

• High-level Network Programming Languages (Network-centric)
  ◦ Arashloo et al., *SNAP: Stateful Network-Wide Abstractions for Packet Processing*, SIGCOMM’16
  ◦ McClurg et al., *Event-driven Network Programming* (“Stateful NetKat”), PLDI ’16

• „In-network“ State Machine Implementation (OpenFlow-based)
In-Network Computing – Background

• Data Plane Programming
  ◦ Hardware- and protocol-agnostic language (P4)
  ◦ .. defining forwarding behavior of reconfigurable data plane hardware
  ◦ Key elements (programmable)
    ▪ Parser / deparser \(\rightarrow\) semantics of packet header
    ▪ Match-action engine \(\rightarrow\) semantics of packet processing
  ◦ P4 was not designed for general-purpose computing

• In-Network Computing
  ◦ Offloading of application functionality from end-systems to data plane
    ▪ Leverage performance of specialized forwarding hardware
  ◦ Typical targets: programmable NICs, FPGAs, programmable data-center switches (based on ASICs or FPGAs)
In-Network Computing – Challenges for Stateful Packet Processing

• Target-dependent limitations
  ◦ Consistency of state data
    ▪ Synchronization of access  ▪ Atomic operations
  ◦ Line-rate enforcing → limited processing (pipeline steps)
  ◦ Limited memory for control logic and state (SRAM, TCAM)

• Language-dependent limitations (P4)
  ◦ No floats, no loops, missing arithmetic operators, etc.
    → P4 not designed for general-purpose computing (not Turing-complete)

• Increasing processing capabilities (extensibility)
  ◦ Increase expressiveness by leveraging „extern functions“ mechanism
  ◦ Interface: target-dependent (P4_{14}) or standardized (P4_{16} primitive)

"Packet Transactions ..." Sivaraman et al., SIGCOMM’16
Event Encoding & Packet Classification

- Events are encoded in packet headers
  - Leverage P4’s flexible parser / deparser
  - Basic events $\rightarrow$ P4 parser
    - Fields: event type and values
    - Pattern matching based on predicates over these header fields
  - Returned complex events $\rightarrow$ P4 deparser

- Classification of ingress packets to discriminate...
  - Basic events $\rightarrow$ CEP Engine
  - Non-CEP traffic, complex events $\rightarrow$ Co-NF
Evaluation – Results

- **NFP-C:**
  - $9.8 \mu s \leq l_p \leq 29.5 \mu s$
  - $56\% \geq B_p \geq 16\%$
  - $l_p$ scales linearly with window size $n$ ($n \leq 1000$)

- **bmv2:**
  - $512 \mu s \leq l_p \leq 10,000 \mu s$
  - $B_p \approx 0.05\%$
Evaluation – Additional Results

• Baselines performance (parsing L2-L5, smallest-packet size)
  ◦ NFC: \( l_p: 6.8 \text{ µs}; \; B_p: \text{line-rate} \)
  ◦ bmv2: \( l_p: 475 \text{ µs}; \; B_p: 0.08\% (12 \text{ Kpps}) \)

• Scalability
  ◦ Number of expressions on NFC \( n \leq 20 \) \( \rightarrow \) constant \( l_p \) and \( B_p \)
  ◦ Predicate complexity \( |P_x| \leq 8 \) on NFC \( \rightarrow \) constant \( l_p \) and \( B_p \)
  ◦ Number of complex events to detect on NFC/-C: \( \leq 4 \)
  ◦ Number of pattern interleavings on NFC/-C: \( \leq 5 \)

• Apache Flink performance
  ◦ \( l_p: 232 \text{ µs} \)
  ◦ \( B_p: \sim 750 \text{ Kpps} \) (1 node, CPU-dependent, external measurement)
P4CEP Compiler – Code Generation

Rule Spec. → Parser → AST → Machine Builder → AST Transform → Control Data Generator

Control Data Generator → Control Data

Control Data Generator → AST Transforms

AST Transforms → P4 Code

Code Generator

Table entries

JSON

Control table

Registers