NFQL: A Tool for Querying Network Flow Records [6]

nfql.vaibhavbajpai.com

Vaibhav Bajpai, Johannes Schauer,
Corneliu Claudiu Prodescu, Jürgen Schönwälder

{v.bajpai, j.schauer, c.prodescu, j.schoenwaelder}@jacobs-university.de

IETF 87, Berlin

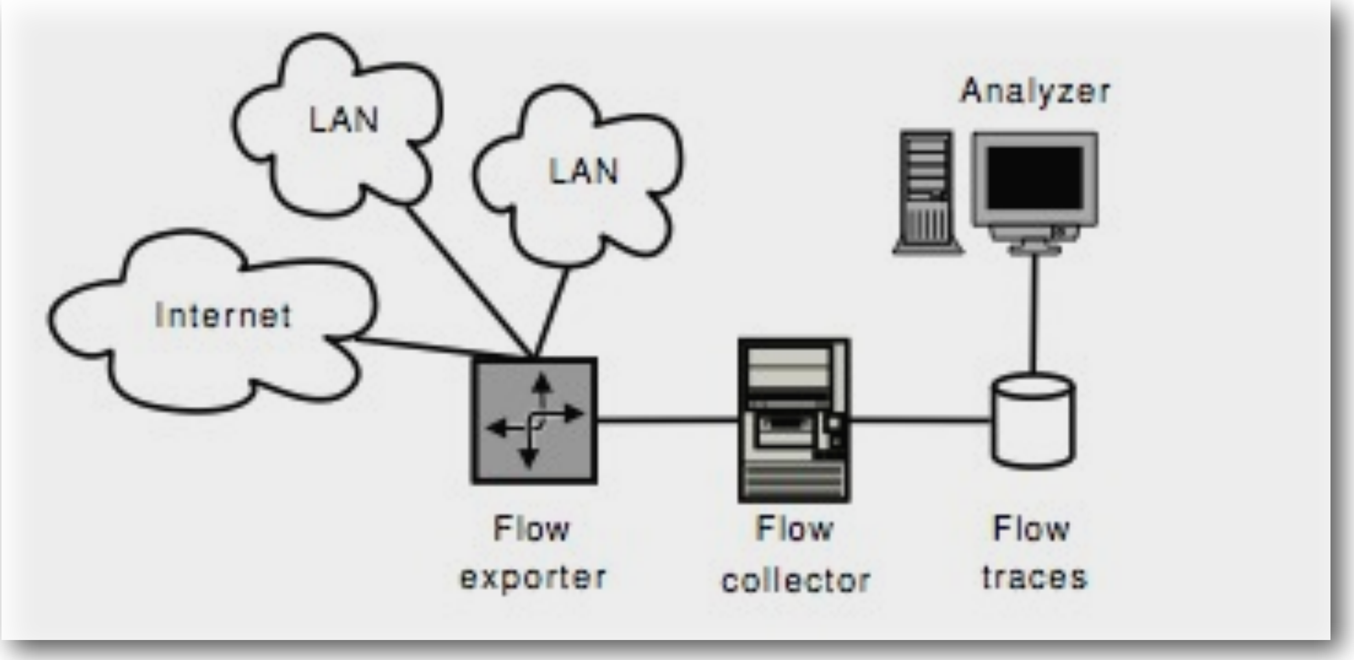
Computer Networks and Distributed Systems
Jacobs University Bremen
Bremen, Germany

July 2013

Supported by:
Flamingo Project: http://fp7-flamingo.eu

Motivation

- IP traffic flow



Flow analysis use cases:

- Survey on detection of intrusion attacks [1].
- Survey on behavior analysis of backbone traffic [2].

Version	Features
v1, {2, 3, 4}	original format with several internal releases
v5	CIDR, AS support and flow sequence numbers
v{6, 7, 8}	router-based aggregation support
v9	template-based with IPv6 and MPLS support
IPFIX	universal standard, transport-protocol agnostic

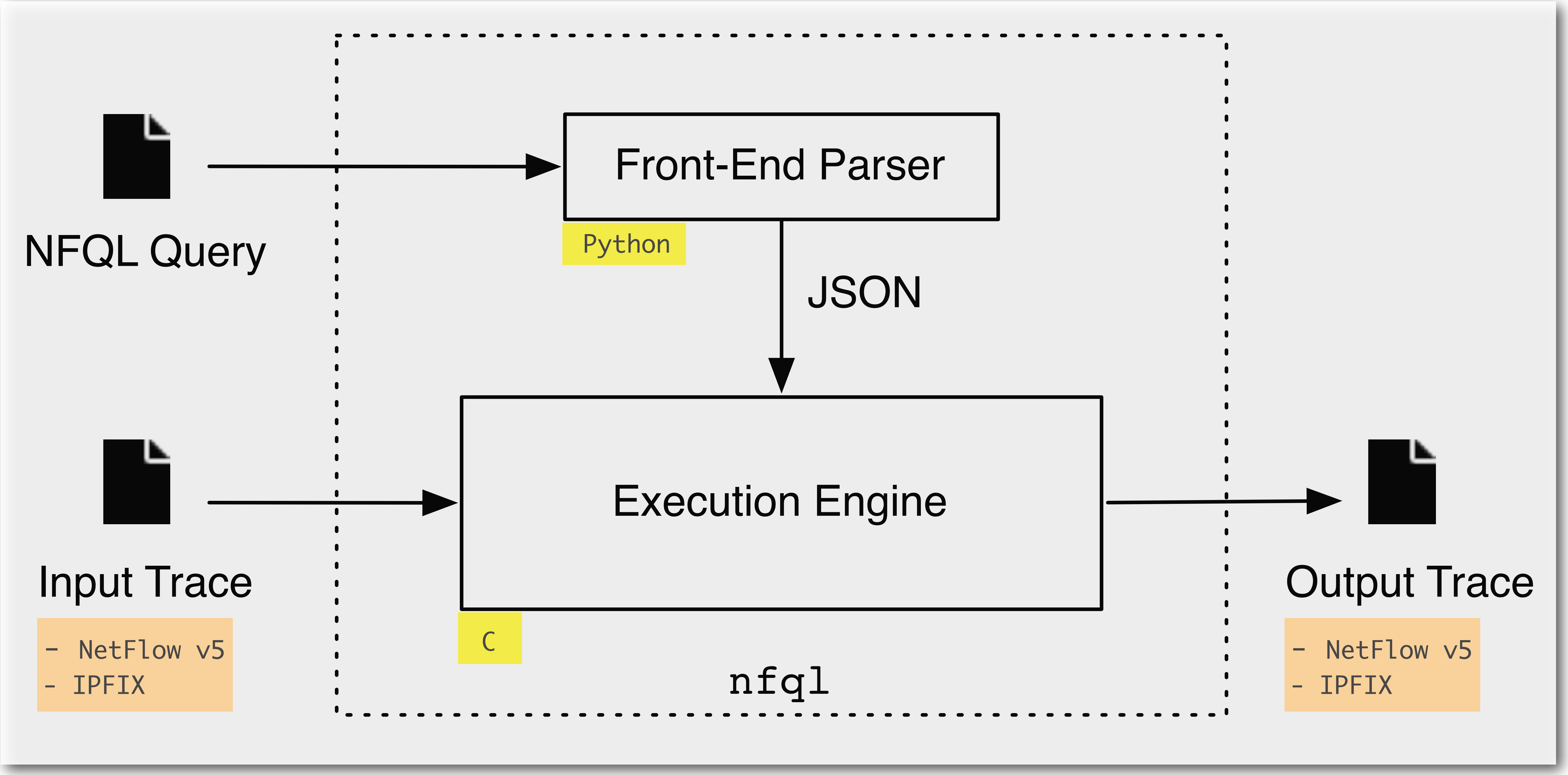
- Flow export protocols
  - Cisco NetFlow [RFC 3954]
  - IETF IPFIX [RFC 5101]

- Understanding intricate traffic patterns require sophisticated flow analysis tools.
- Current tools span a smaller use-case owing to their simplistic language designs.

Related Work

- Simple traffic analysis tools
  - ntop, FlowScan, NfSen, Stager
- Popular open-source NetFlow analysis tools
  - `flow-tools`: supports NetFlow v5
  - `nfdump`: supports NetFlow v9
- Popular open-source IPFIX analysis tools
  - `SiLK`

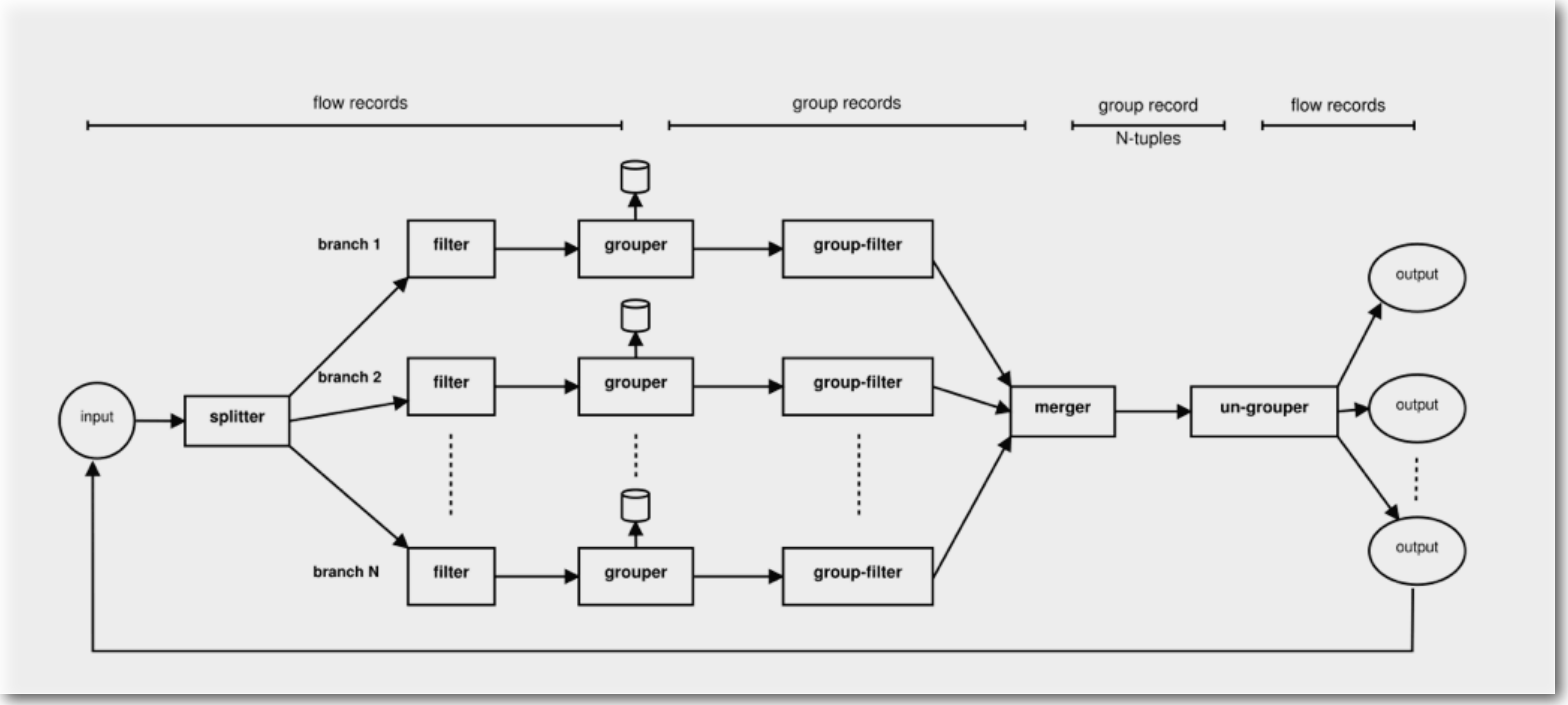
nfql Tool



nfql architecture

NFQL (Network Flow Query Language)

NFQL processing pipeline [3]



- Each branch runs in a separate thread.
- Affinity masks help delegate each branch to a separate processor core.

NFQL Domain Specific Language (DSL)

```
filter http {
    tcpDestinationPort = 80 delta 1
}
```

DSL

NFQL Parser

```
"filter": {
  "dnf-expr": [{
    "clause": [{
      "term": {
        "delta": 1,
        "offset": {
          "name": "destinationTransportPort",
          "value": 80
        },
        "op": "RULE_EQ"
      }
    }]
  }]
}
```

JSON intermediate format

The query uses IPFIX entity names and datatypes:
http://www.iana.org/assignments/ipfix/ipfix.xhtml

- JSON intermediate format
  - Each pipeline stage of the JSON query is a DNF expression.
  - JSON query can disable the pipeline stages at RUNTIME.
  - Execution engine uses `json-c` to parse the JSON query: http://oss.metaparadigm.com/json-c

NFQL DSL: Supported Features

- Possible Operations:
  - EQ, NE, GT, LT, LE, GE
- Possible Aggregations:
  - COUNT, UNION, MIN, MAX, SUM, MEDIAN,
  - MEAN, STDDEV, XOR, PROD, AND, OR, IN
- Possible Interval Operations:
  - X takes place before Y
  - X meets Y
  - X overlaps with Y
  - X starts Y
  - X during Y
  - X finishes Y
  - X is equal to Y

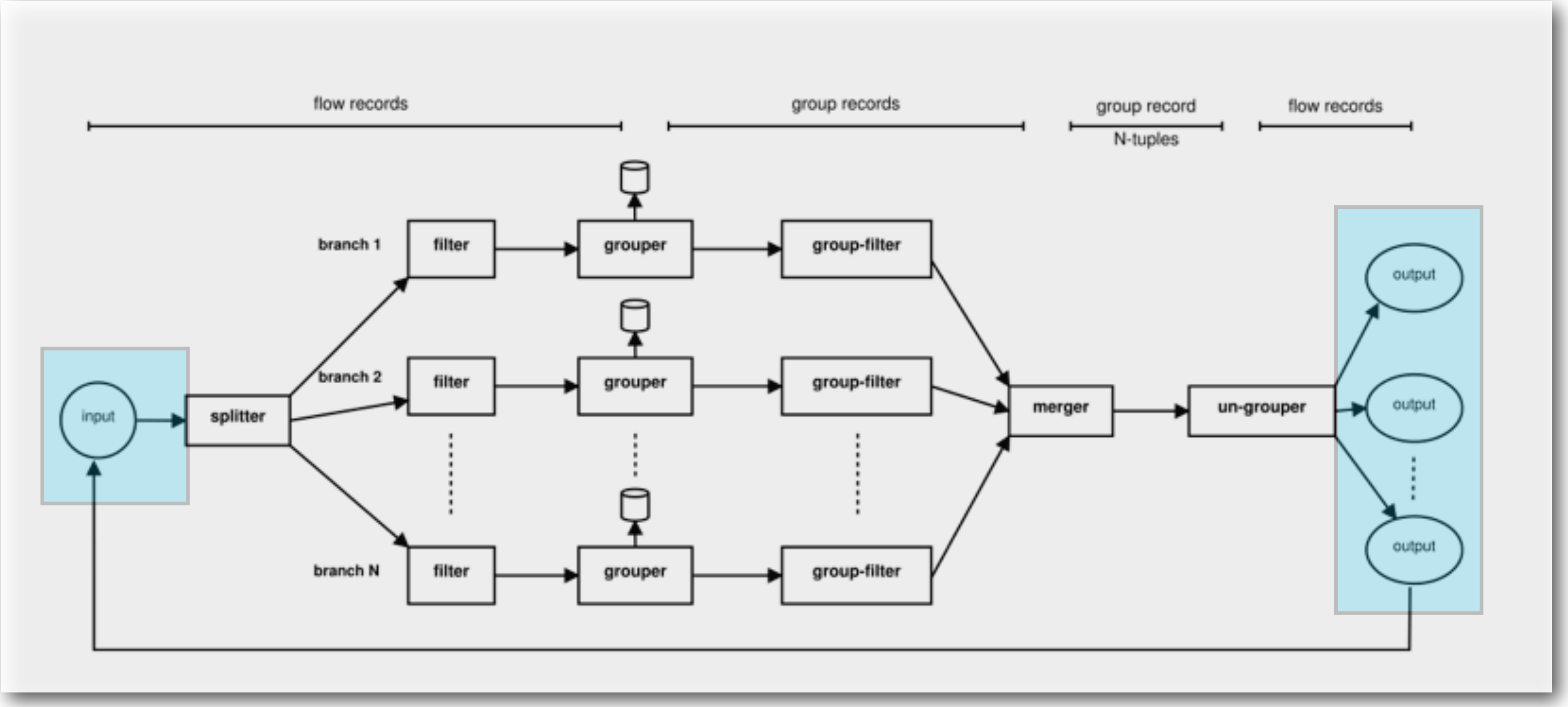
supported in SiLK

NFQL DSL: IPFIX to NetFlow v5 map

NetFlow v5	IPFIX	Comments
srcaddr	sourceIPv4Address	
dstaddr	destionationIPv4Address	
nexthop	ipNextHopIPv4Address	
input	-	missing in IPFIX?
output	-	missing in IPFIX?
dPkts	packetDeltaCount	32bit unsigned vs 64bit unsigned
dOctets	octetDeltaCount	32bit unsigned vs 64bit unsigned
dFlows	deltaFlowCount	32bit unsigned vs 64bit unsigned
First	flowStartSysUpTime	relative vs absolute time
Last	flowEndSysUpTime	relative vs absolute time
srcport	sourceTransportPort	
dstport	destinationTransportPort	
tcp_flags	tcpControlBits	
prot	protocolIdentifier	
tos	ipClassOfService	
src_as	bgpSourceAsNumber	
dst_as	bgpDestinationAsNumber	
src_mask	sourceIPv4PrefixLength	
dst_mask	destinationIPv4PrefixLength	

NFQL I/O processing

NFQL processing pipeline [3]



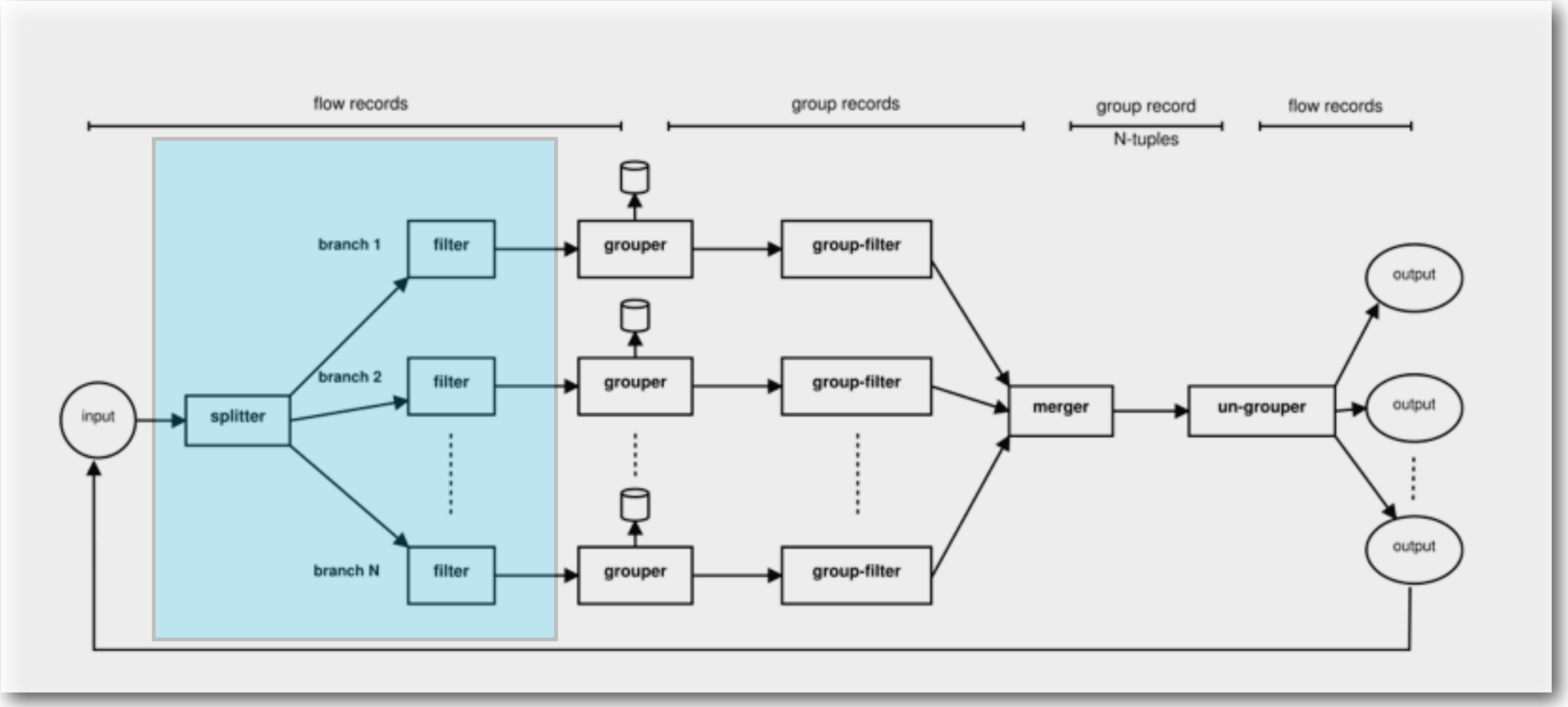
- NetFlow v5: using `flow-tools`: http://www.splintered.net/sw/flow-tools
- IPFIX: using `libfixbuf`: http://tools.netsa.cert.org/fixbuf
- Flow records are read in memory and indexed to allow retrieval in $O(1)$ time.

NFQL Example:

- <u>Problem Statement:</u>
  - Find all flow pairs representing HTTP traffic (TCP using port 80) that have exchanged more than 200 packets in both directions.

NFQL Example: Filter

NFQL processing pipeline [3]



HTTP requests:

```
branch A {
  filter f1 {
    destinationTransportPort=80
    protocolIdentifier=TCP
  }
}
```

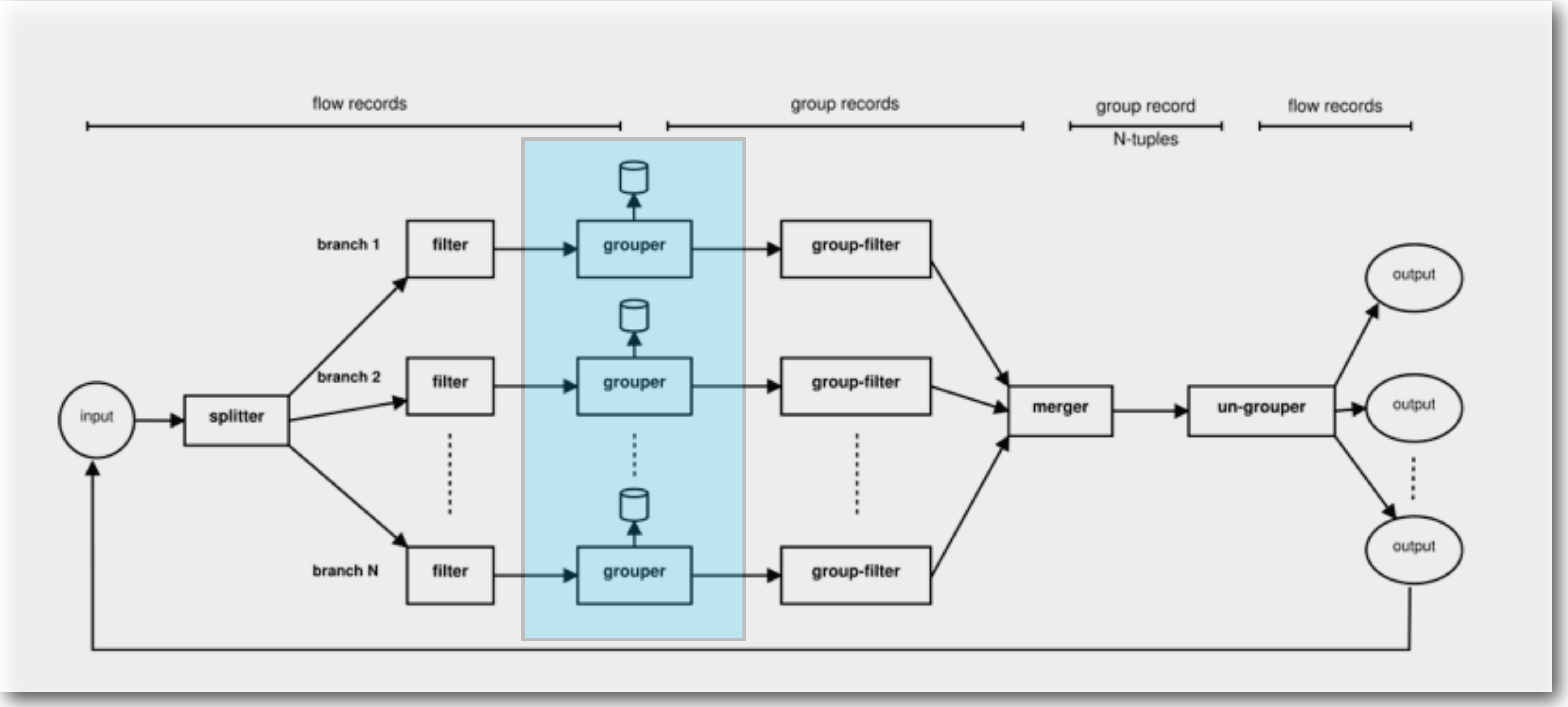
HTTP responses:

```
branch B {
  filter f1 {
    sourceTransportPort=80
    protocolIdentifier=TCP
  }
}
```

- No splitter: Using indexes to reference flows in each branch.
- Inline filter: Flows are filtered as soon as they are read in memory.

NFQL Example: Grouper

NFQL processing pipeline [3]



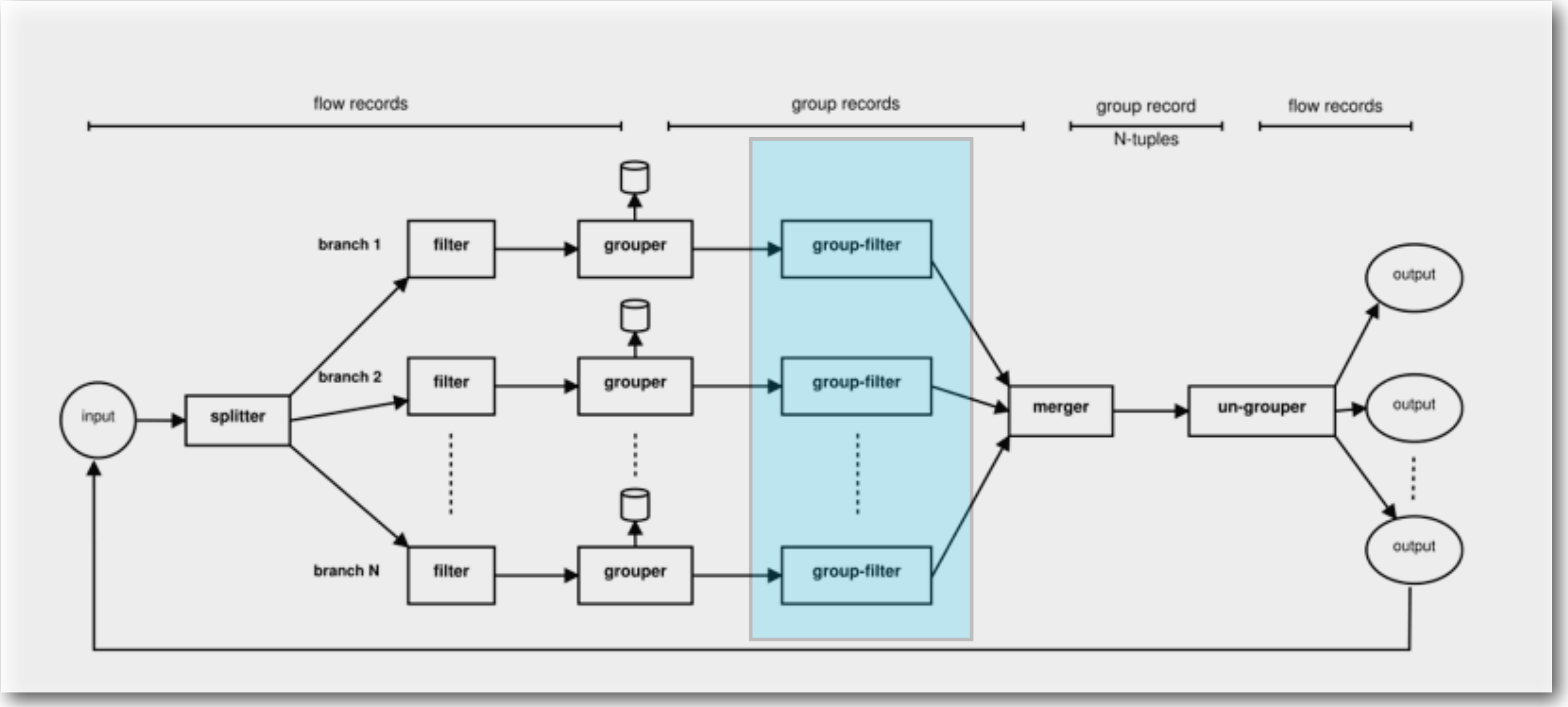
Group A and Group B

```
grouper ... {
    sourceIPv4Address =
    sourceIPv4Address
    destinationIPv4Address =
    destinationIPv4Address
    aggregation {
      sum(packetDeltaCount)
      sum(octetDeltaCount)
    }
}
```

- Flow records matching the source and destination endpoint addresses are combined.
- The number of packets and octets are aggregated together within each grouped flow.
- Faster grouper lookups: Sort on group keys and perform a nested binary search.

NFQL Example: Group Filter

NFQL processing pipeline [3]

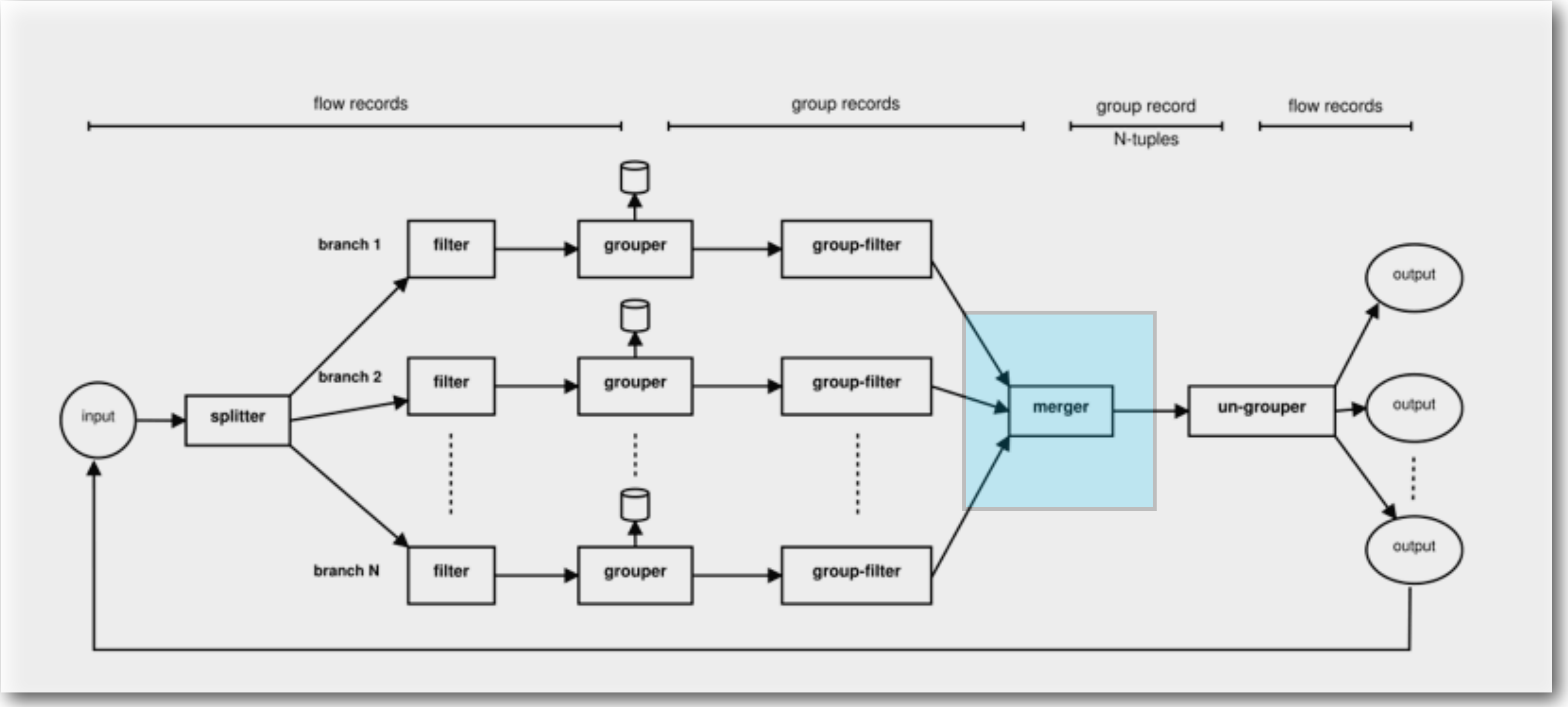


Group A and Group B

```
groupfilter ... {
  packetDeltaCount > 200
}
```

NFQL Example: Merger

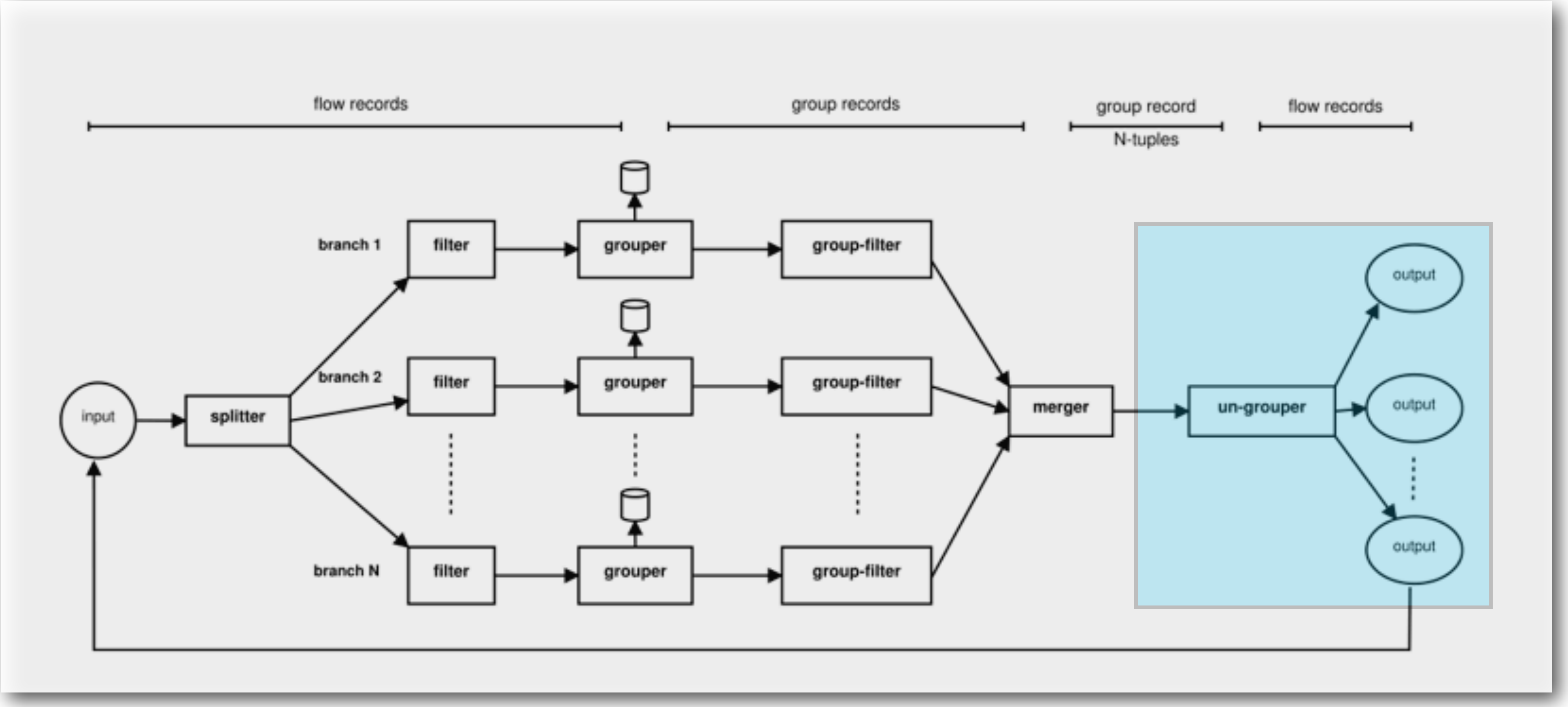
NFQL processing pipeline [3]




```
branch A { ... }

branch B { ... }

merger M {
  A.sourceIPv4Address =
  B.destinationIPv4Address

  A.destinationIPv4Address =
  B.sourceIPv4Address
}
```

- Merger merges the grouped flows from each branch to create *streams*.
- The HTTP request flow is matched with the HTTP response flow to create a HTTP session.
- Faster merger matches: Sort on merger keys to skip iterator permutations.

NFQL Example: Ungrouper



NFQL processing pipeline [3]

```
ungrouper U {

}
```

- The ungrouper unfolds the *streams* back into individual flows.
- The individual flows are written as trace files or printed on `stdout`.

nfql Tool

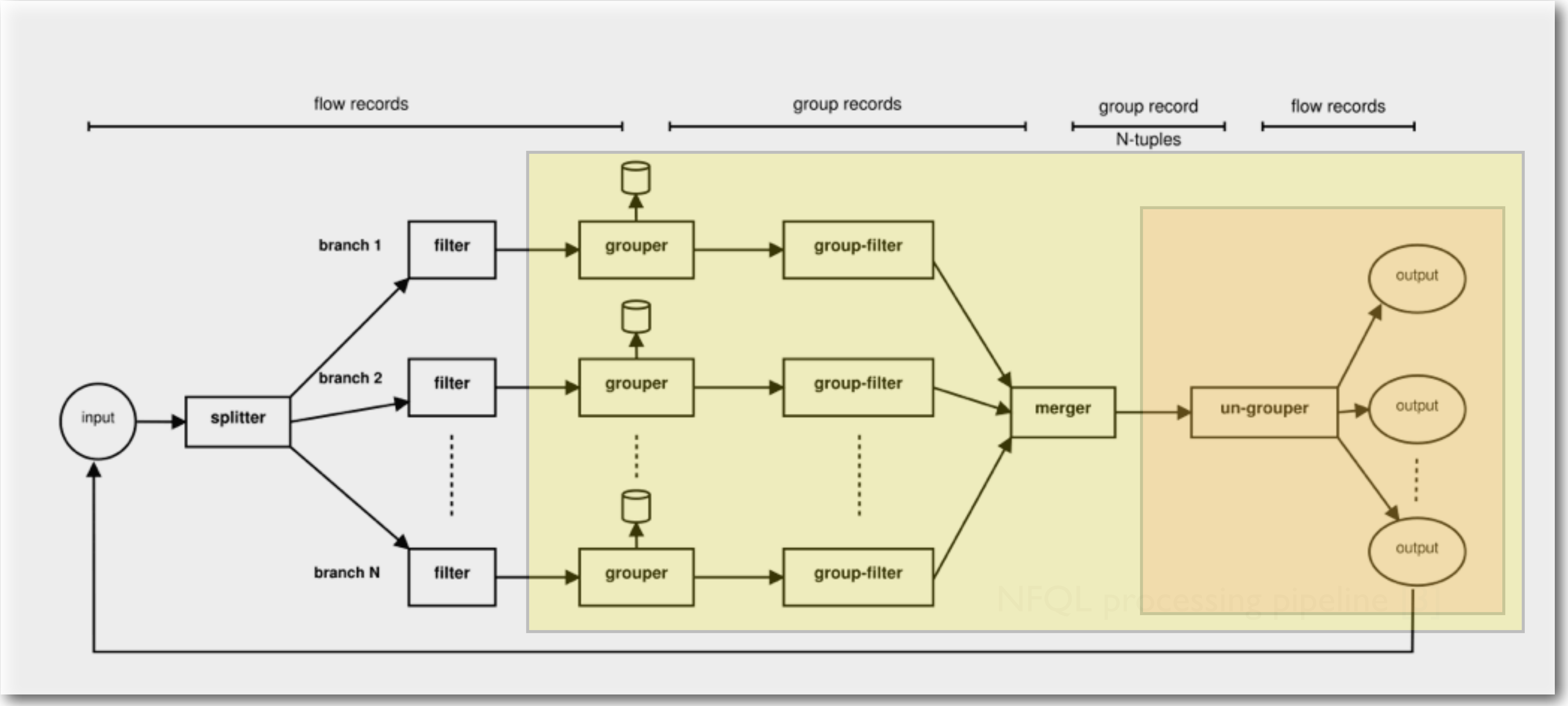
- Demo
  - Find all flow pairs representing HTTP traffic (TCP using port 80) that have exchanged more than 200 packets in both directions.

NFQL in Theory

- Features
  - Filter flows.
  - Combine flows into groups.
  - Aggregate flows on flow-keys as one grouped flow aggregate.
  - Merge grouped flows, supporting temporal relations between groups.
  - Apply absolute or relative filters when grouping or merging.
  - Unfold grouped flows back into individual flows.

Filter (worst case)	$O(n)$ *where* $n=num(flows)$
Grouper (average case)	$O(n \times lg(k)) + O(p \times n \times lg(n))$ *where* $k=num(unique(flows))$, $p=num(terms)$
Grouper aggregations (worst case)	$O(n)$
Group Filter (worst case)	$O(g)$ *where* $g=num(groups)$
Merger (worst case)	$O(g\text{\^{}}m)$ *where* $m=num(branches)$
Ungrouper (worst case)	$O(g)$

NFQL and Friends

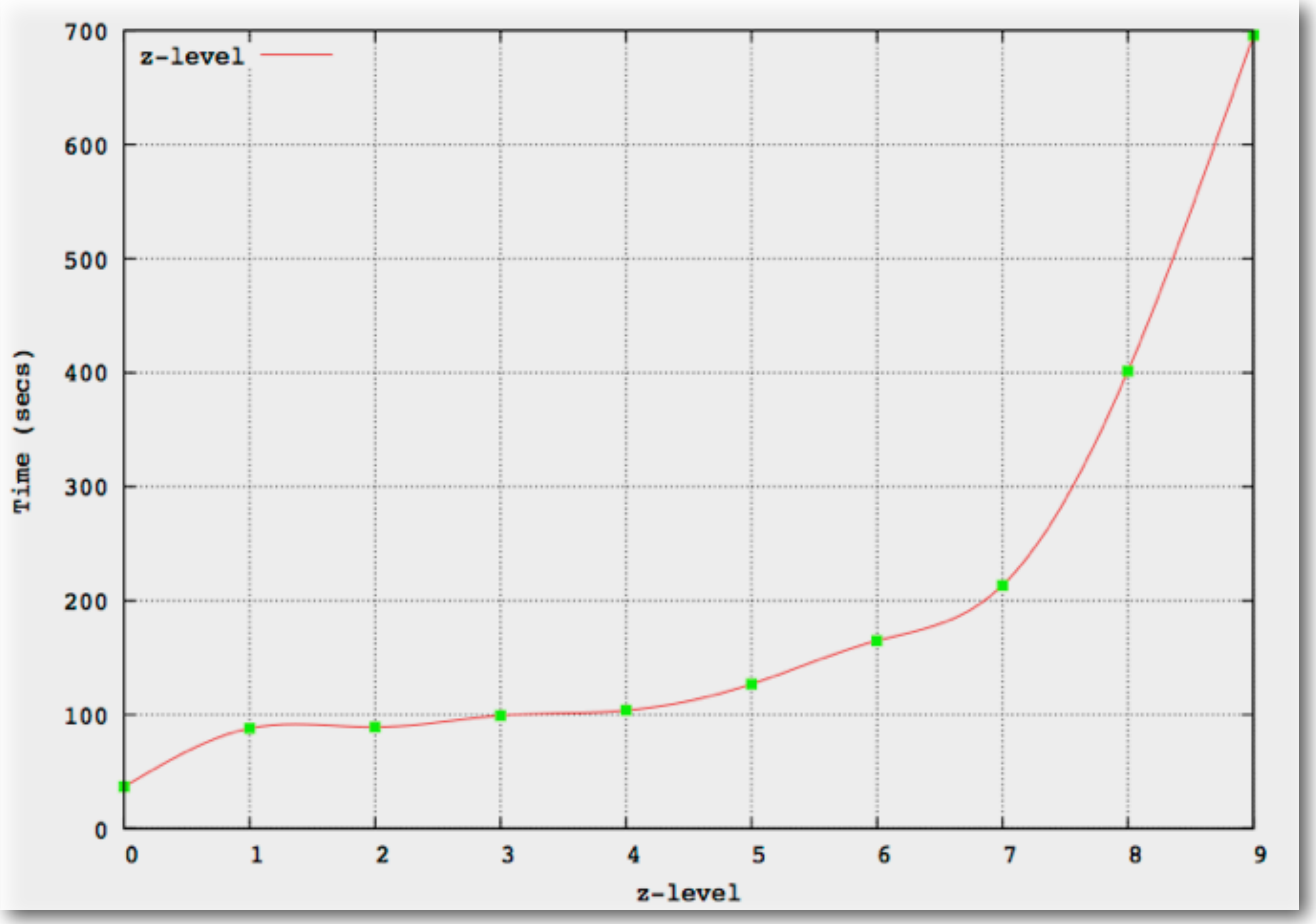


The expressiveness of the language can be seen from [4], where NFQL queries are used to identify application signatures.

not supported by {`flow-tools, nfdump`}

not supported by SiLK

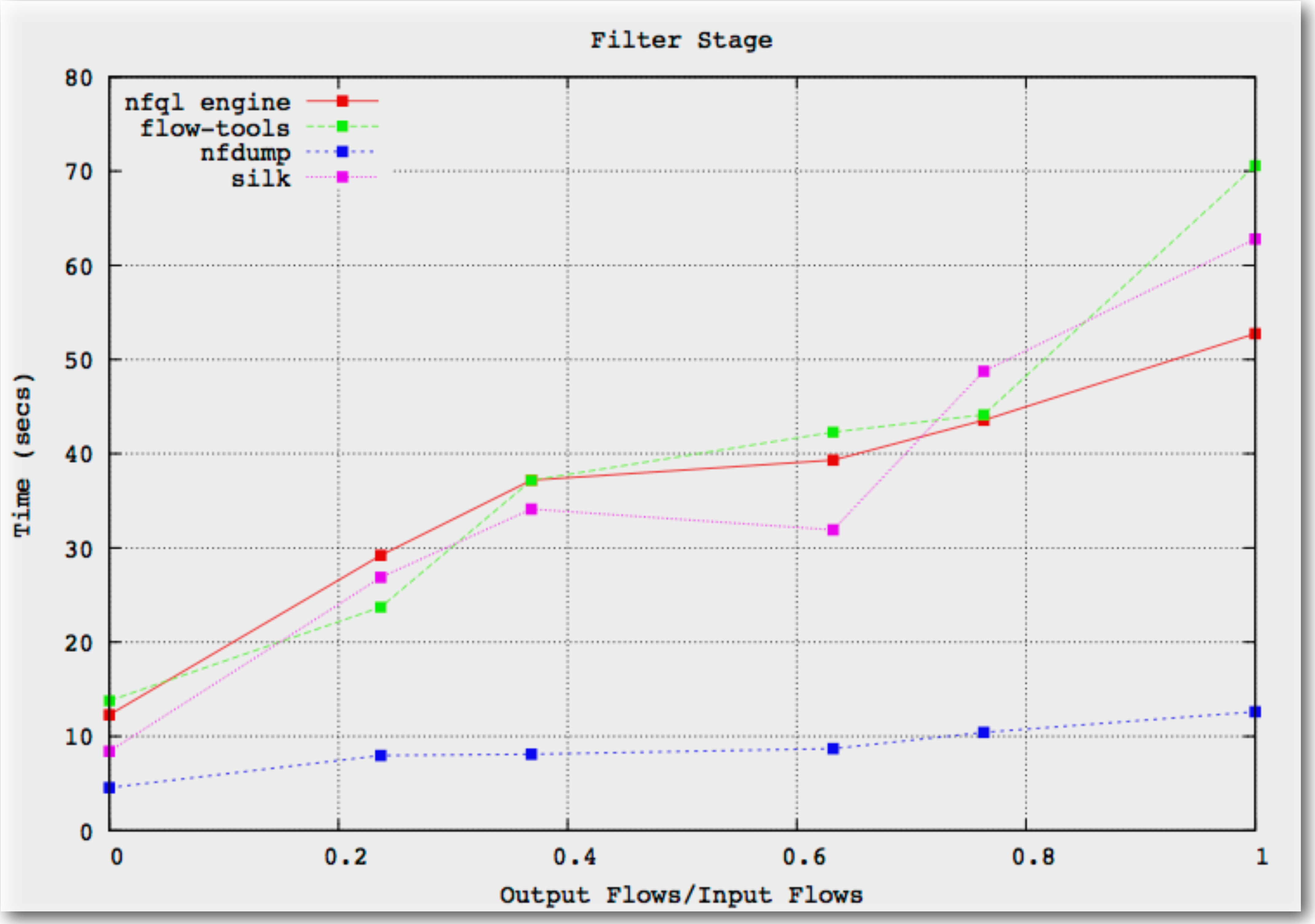
Compression Tradeoffs



- Output traces are compressed using `zlib` library. `nfdump` uses `lzo` compression.
- Compression level is configurable at RUNTIME. `nfql` uses `ZLIB_LEVEL 5` by default.
- Each compression level adds its own performance overhead when writing output traces to files.

- Additional Features
  - Each pipeline stage results can be written out as `flow-tools` files.
  - Capability to read multiple input traces from `stdin`: `$ flow-cat $TRACES | nfql $QUERY`

Performance Evaluations



- Used first 20M flows from Trace 7 in the SimpleWeb repository [5].
- Input trace was compressed `ZLIB_LEVEL 5`.
- Ran on a machine with 24 cores, 2.5 GHz clock speed and 18 MiB of physical memory.
- `nfdump` uses `lzo` compression to trade output trace size with RUNTIME speed.

- Stressing the rest of the pipeline stages, please refer to [6].

Conclusion

- NFQL' richer language capabilities allow sophisticated flow queries.
- `nfql` can process such complex queries in minutes.
- `nfql` has comparable execution times when processing real-world traces.
- Evaluation queries developed as part of this research can become input towards a generic benchmarking suite for flow-processing tools.

nfql.vaibhavbajpai.com

References

[1] A. Sperotto, et al., An overview of IP flow-based intrusion detection, IEEE Communication Surveys and Tutorials, 2010.

[2] A. Callado, et al., A survey on Internet traffic identification, IEEE Communication Surveys and Tutorials, 2009.

[3] V. Marinov, et al., Design of a stream-based IP Flow Record Query Language, Distributed Systems: Operations & Management, 2009

[4] V. Perelman, et al., Flow Signatures of Popular Applications, Symposium on Integrated Network Management, 2011

[5] R. Barbosa, et al., Simpleweb/University of Twente Traffic Traces Data Repository, http://www.simpleweb.org/wiki/Traces [Last Accessed: May 25, 2013]

References

[6] V. Bajpai, et al., NFQL: A Tool for Querying Network Flow Records, IEEE/IFIP International Symposium on Integrated Network Management, 2013.