# **Reinhardt:** Real-time reconfigurable hardware architecture for regular expression matching in DPI

**Taejune Park (Chonnam National University)**

Jaehyun Nam (AccuKnox)

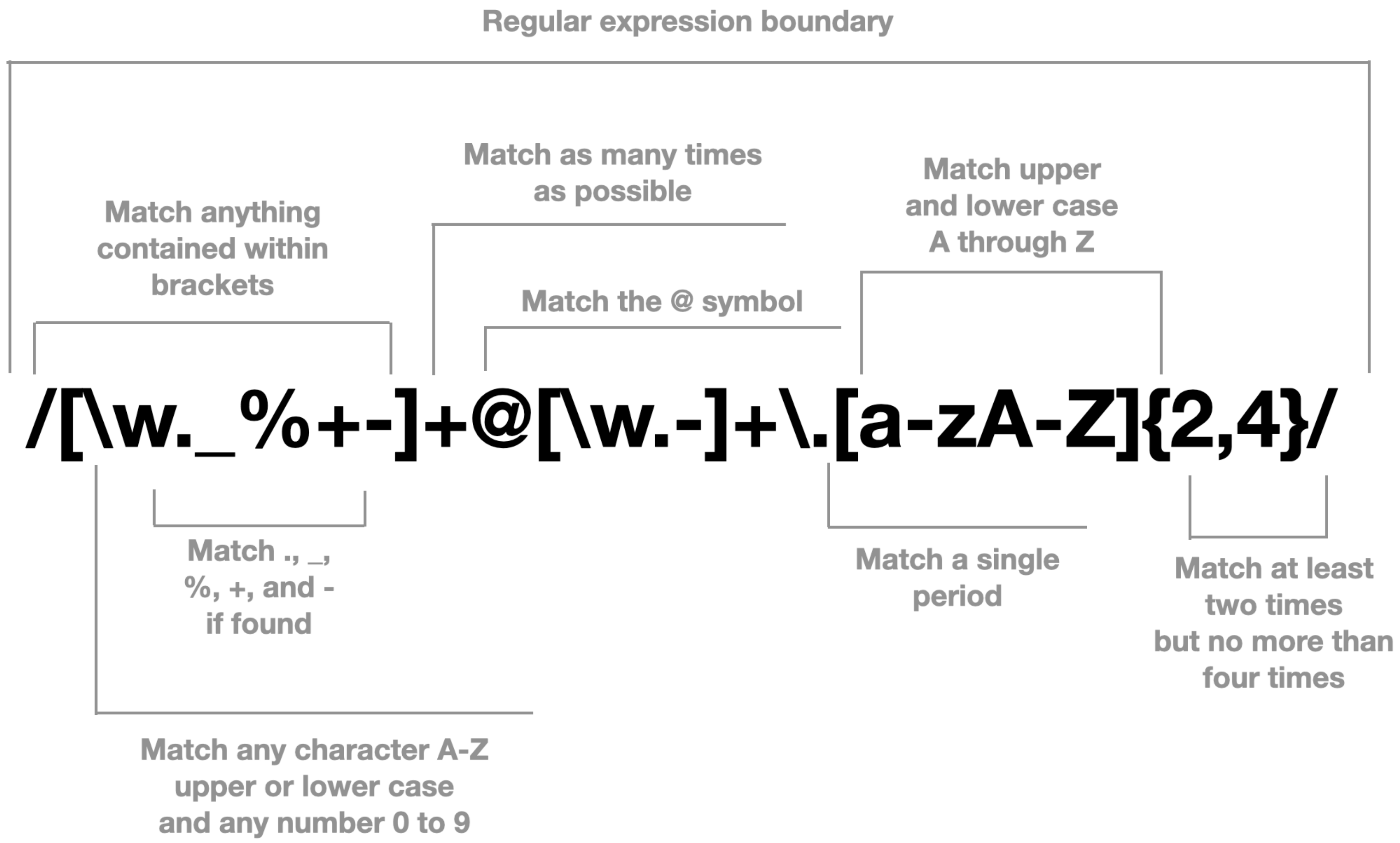
Seung Ho Na (KAIST)

Jaewoong Chung (Atto Research)

Seungwon Shin (KAIST)

# Deep Packet Inspection and regular expression

- **Regex is one of the most important features in DPI (NIDS/IPS)**
  - Inspect packet payload with specific patterns
  - Essential to handle arbitrary protocols in a modern network environment



Regular expression boundary

Match anything contained within brackets

Match as many times as possible

Match upper and lower case A through Z

Match the @ symbol

```
/[\w._%+-]+@[\w.-]+\.[a-zA-Z]{2,4}/
```

Match ., _, %, +, and - if found

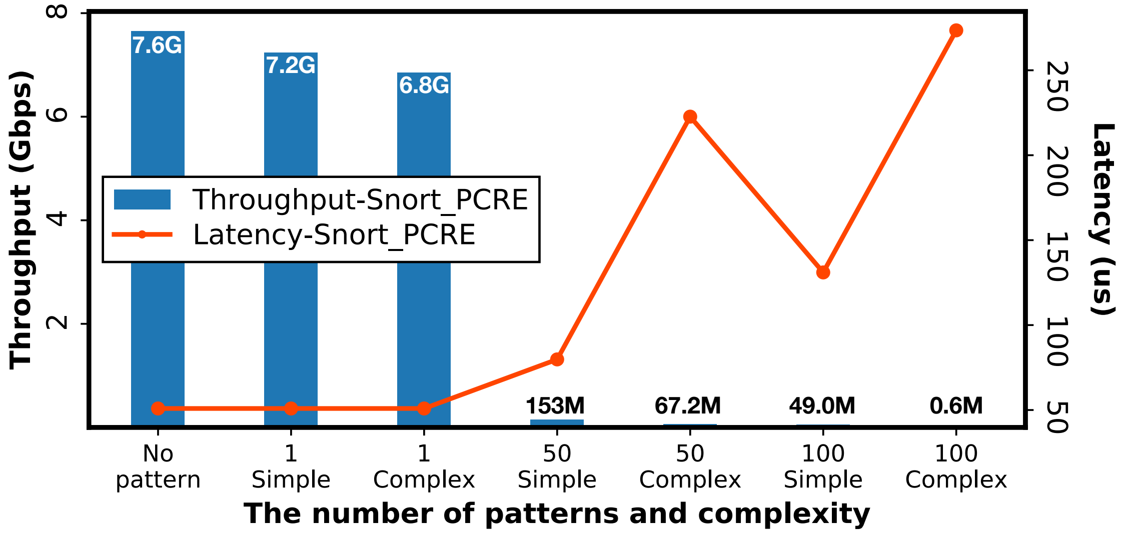
Match a single period

Match at least two times but no more than four times

Match any character A-Z upper or lower case and any number 0 to 9

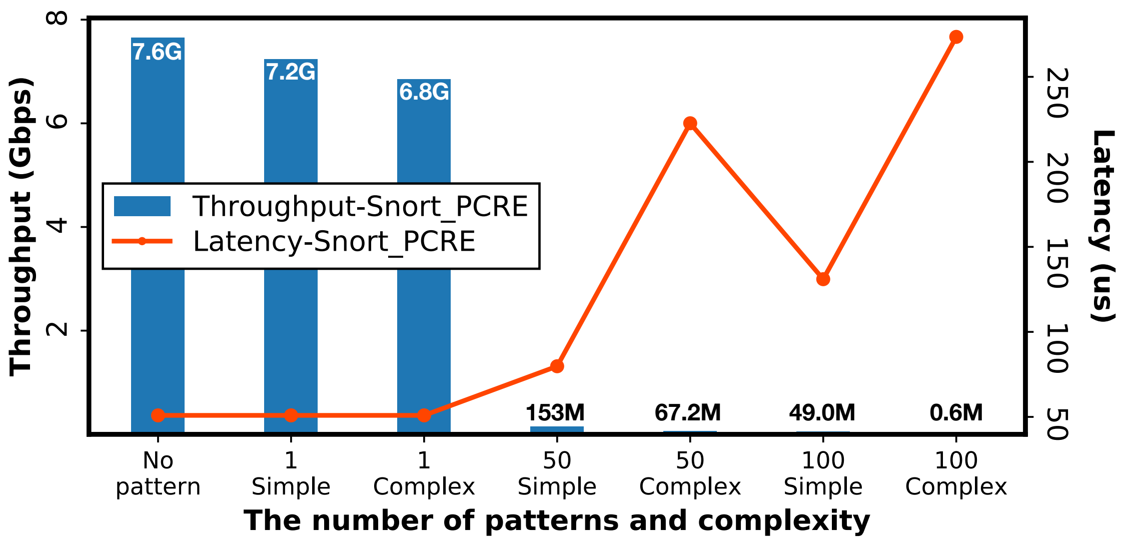
# Challenge of regex processing in DPI

- **Low-performance**
  - Major bottleneck point in both throughput and latency
  - Highly affected by the number and complexity of patterns

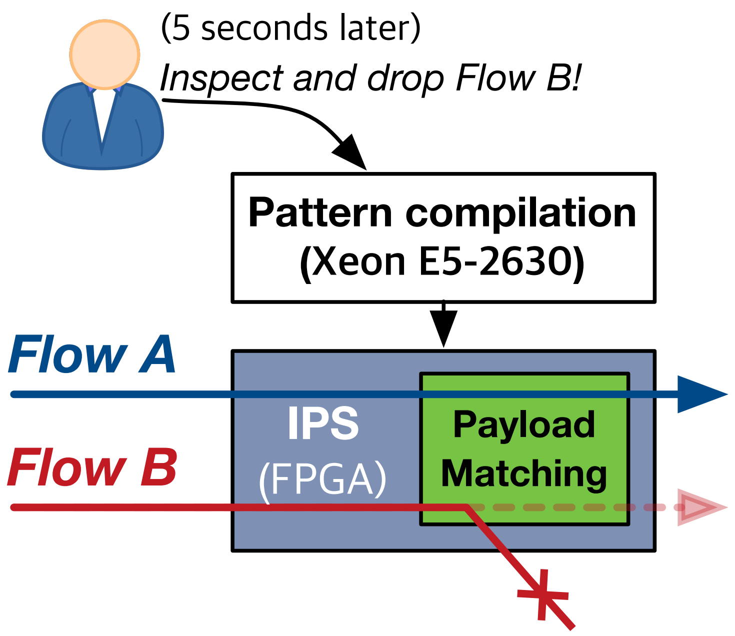


# Challenge of regex processing in DPI

- **Low-performance**
  - Major bottleneck point in both throughput and latency
  - Highly affected by the number and complexity of patterns



- **Accelerating with Programmable Hardware: FPGA**
  - **Natural parallelism of hardware**
  - ***Lack of flexibility in pattern update***
    - Long compilation time for hardware logic: *Updating policies takes at least hours*
    - Inevitable Service Interruption
    - All-or-Nothing Update Operation



(5 seconds later)
*Inspect and drop Flow B!*

**Pattern compilation (Xeon E5-2630)**

***Flow A***

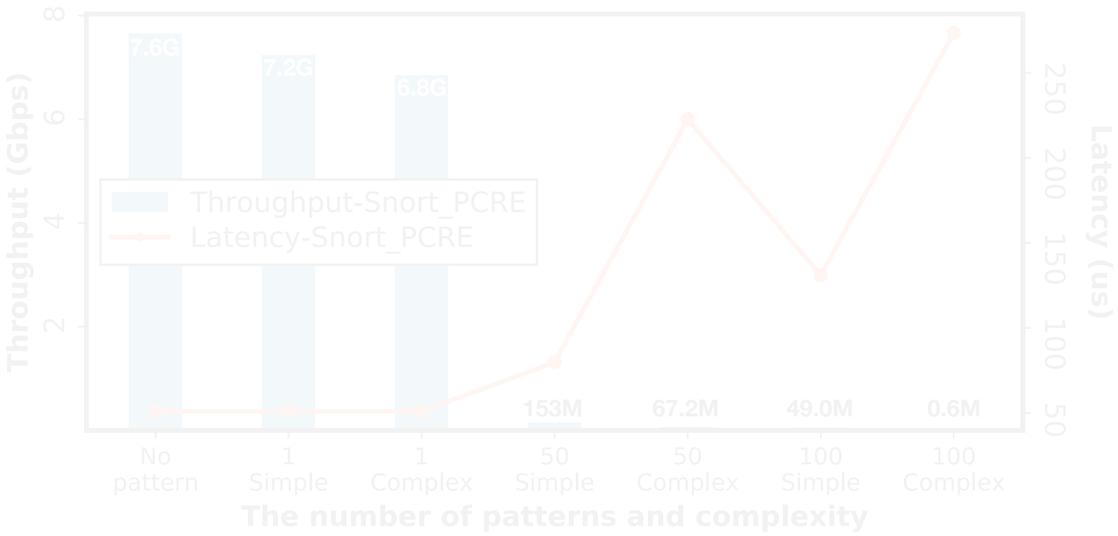
***Flow B***

**IPS** (FPGA)

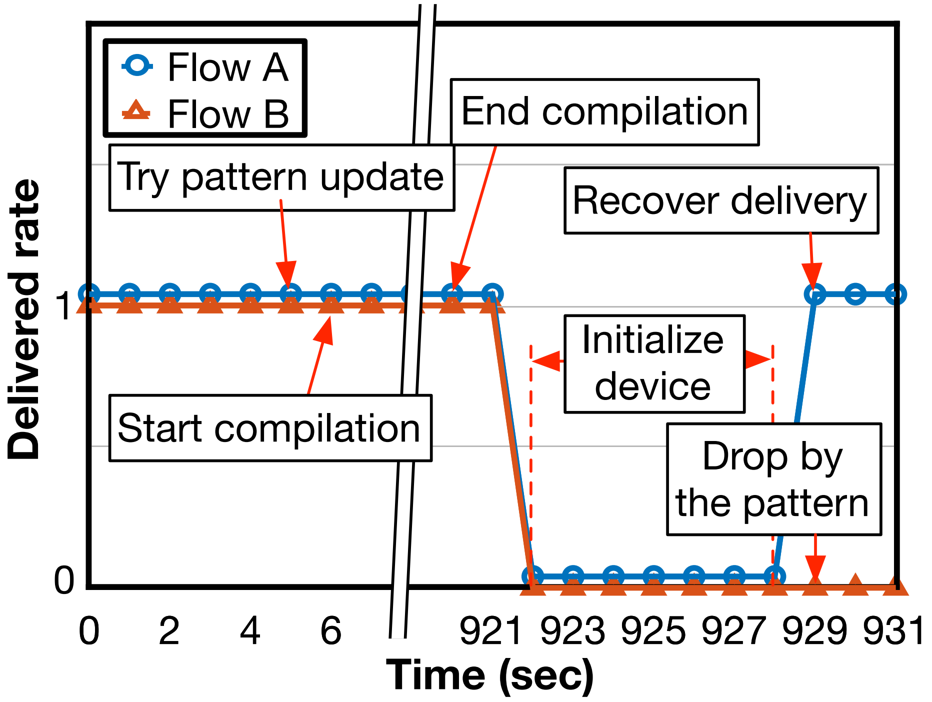
**Payload Matching**

# Challenge of regex processing in DPI

- **Low-performance**
  - Major bottleneck point in both throughput and latency
  - Highly affected by the number and complexity of patterns

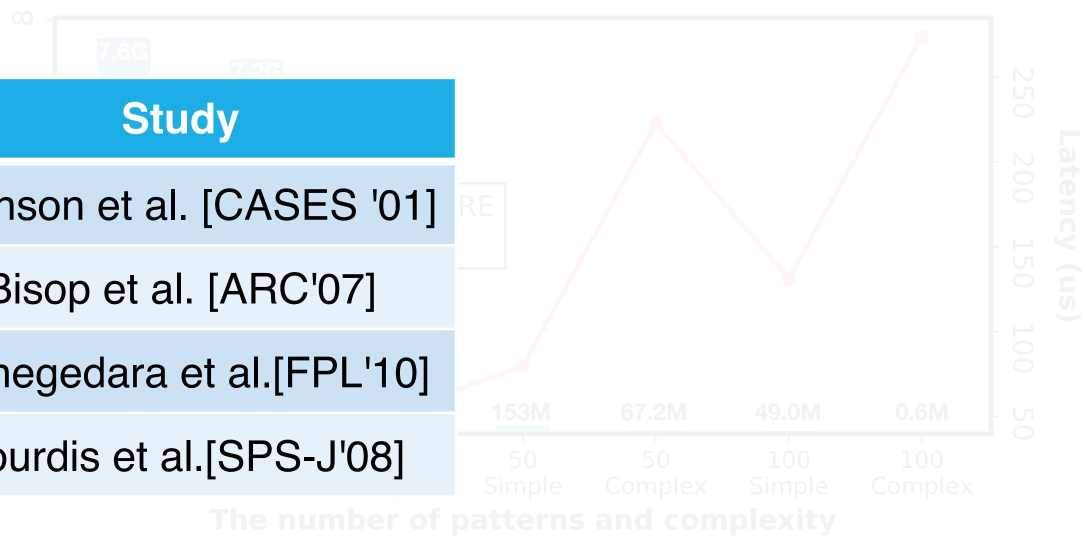


- **Accelerating with Programmable Hardware: FPGA**
  - **Natural parallelism of hardware**
  - ***Lack of flexibility in pattern update***
    - Long compilation time for hardware logic: *Updating policies takes at least hours*
    - Inevitable Service Interruption
    - All-or-Nothing Update Operation



# Challenge of regex processing in DPI

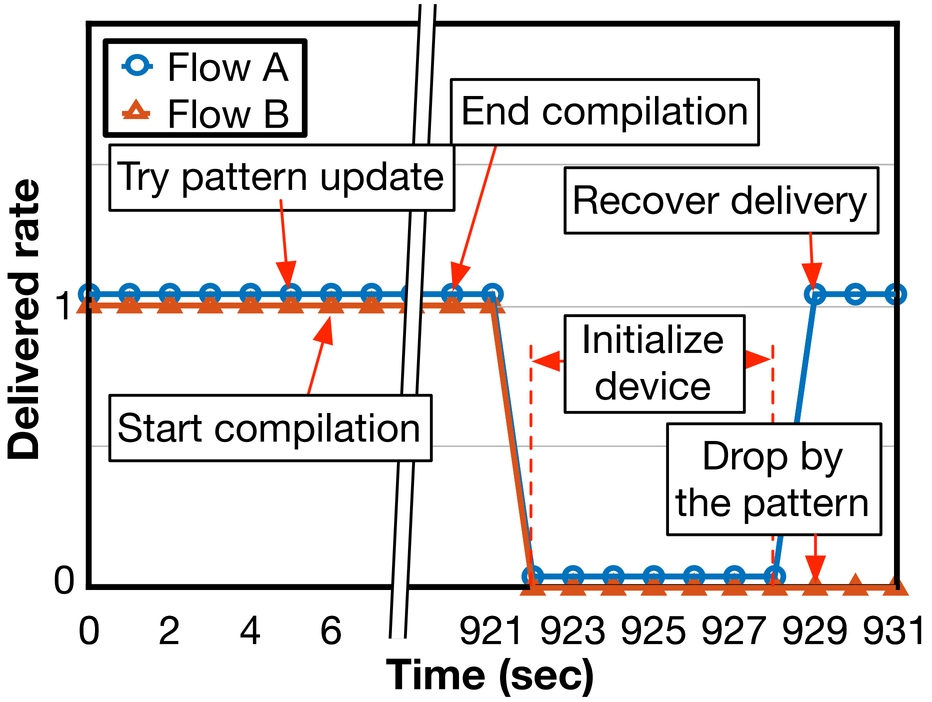
- Low-performance
  - Major bottleneck in both throughp
  - Highly affected b and complexity o



**Known update time**

| # of patterns | Update time (h:mm:ss) | Study |
|---|---|---|
| 200 | **1:38:57** | Johnson et al. [CASES '01] |
| 310 | **1:47:00** | Bisop et al. [ARC'07] |
| 760 | **1:52:00** | Ganegedara et al.[FPL'10] |
| 1,504 | **4:53:50** | Sourdis et al.[SPS-J'08] |

- **Accelerating with Programmable Hardware: FPGA**
  - **Natural parallelism of hardware**
  - ***Lack of flexibility in pattern update***
    - Long compilation time for hardware logic: *Updating policies takes at least hours*
    - Inevitable Service Interruption
    - All-or-Nothing Update Operation



# **Reinhardt:**
# Real-time reconfigurable hardware architecture for regex

- **Goal: a high-performance and programmable hardware regex matching**
  - **Supporting high-performance regex matching for DPI as well as NIDS/IPS**
    - Line-rate throughput and low-latency
  - **Enabling hardware real-time programmable**
    - Software-like programmability in updating regex patterns
    - Reinhardt host software to manage the hardware processor

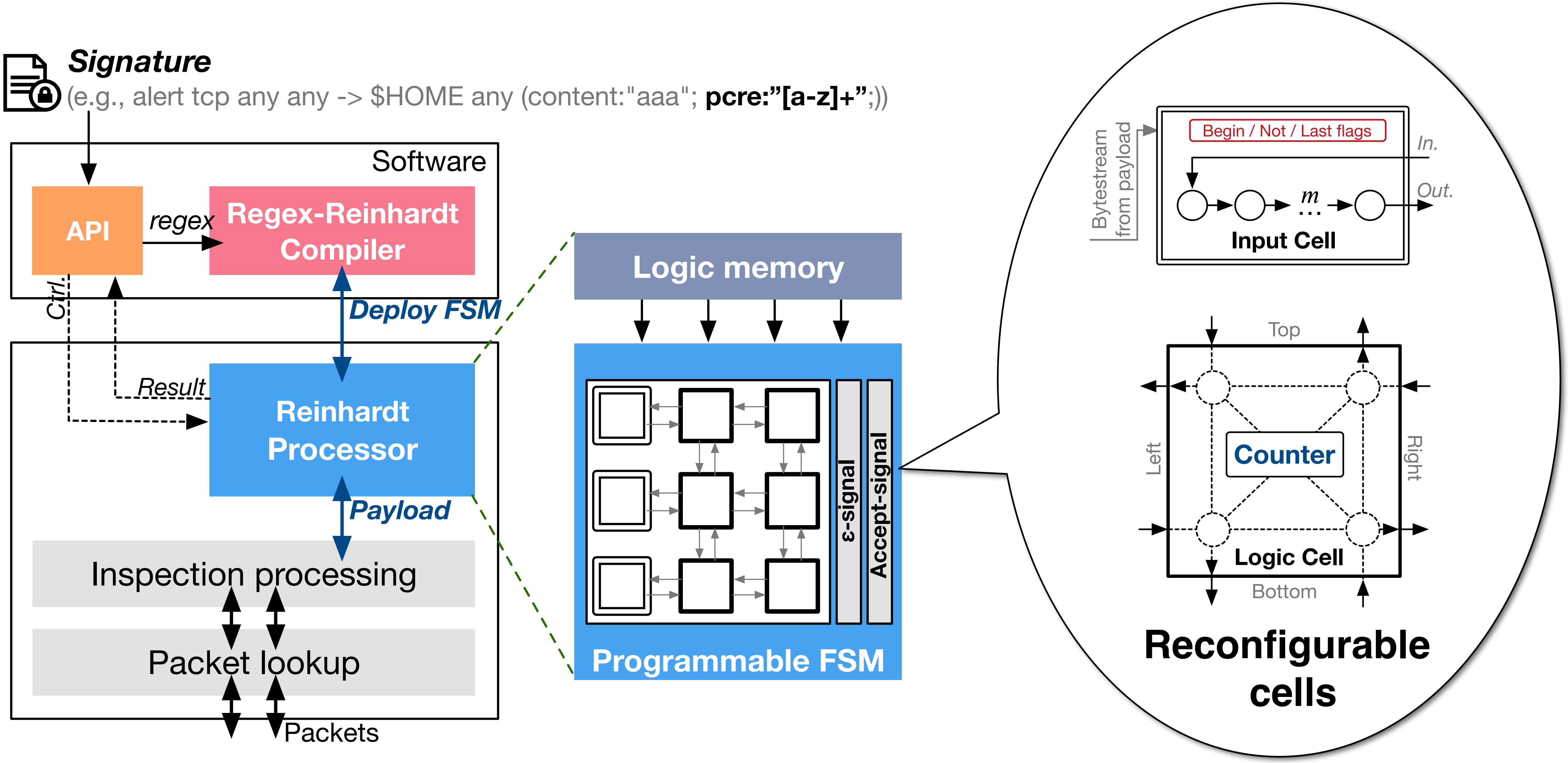
## Challenges

- **The long compilation time of hardware circuit implementation**
- **Support any arbitrary regex patterns (POSIX standard)**
- **The reasonable number of patterns**

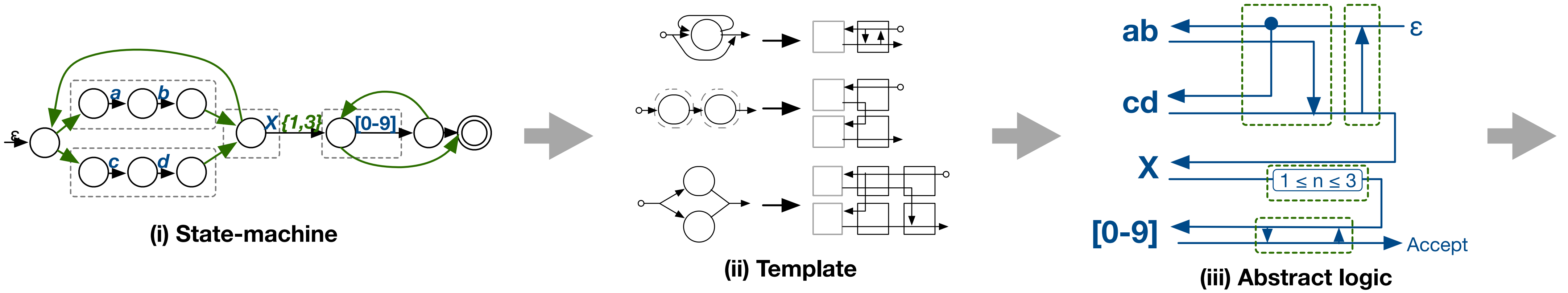
# Our approaches

- **The long compilation time of hardware circuit implementation**
  - Design a hardware circuit that generates hardware circuits in real-time
  - Provide a compiler for implementing circuits in the hardware circuit
- **Support any arbitrary regex patterns (POSIX standard)**
  - Regex expression matching begins by generating an equivalent state machine
  - Generalize how the state machines are constructed into hardware circuit, and structuralize this task through the hardware circuit
- **The reasonable number of patterns**
  - Resubmitting: Recursive processing by exploiting the high-programmability

# Reinhardt overview



# Real-time programmable payload inspection system



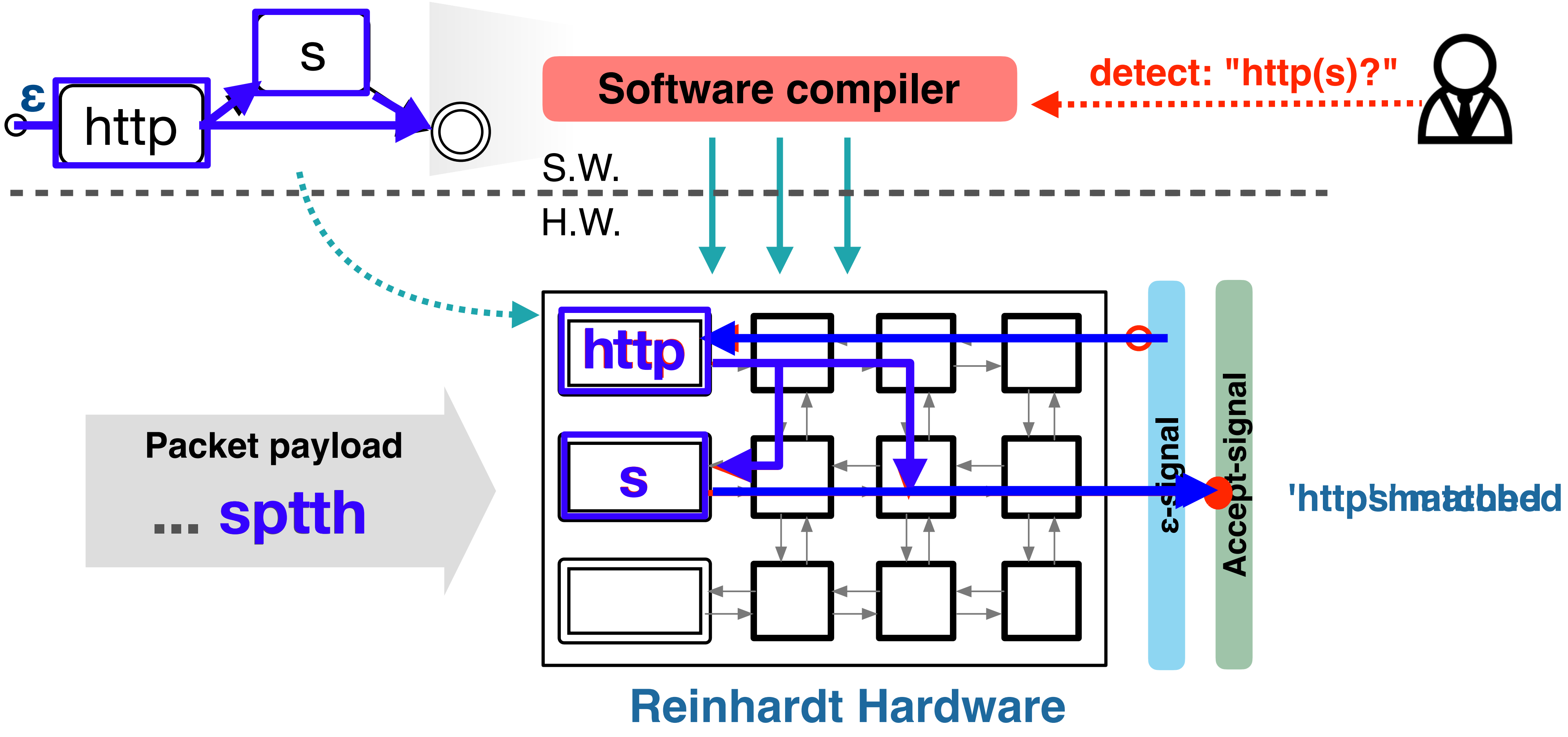
(i) State-machine

(ii) Template

(iii) Abstract logic

```
(0.0)_"ab"
(0.1)_l→r/r→bl
(0.2)_l→b/r→l/b→l
(0.3)_r→l
(1.0)_"cd"
(1.1)_t→l/l→r
…
```

(iv) Configuration opcode



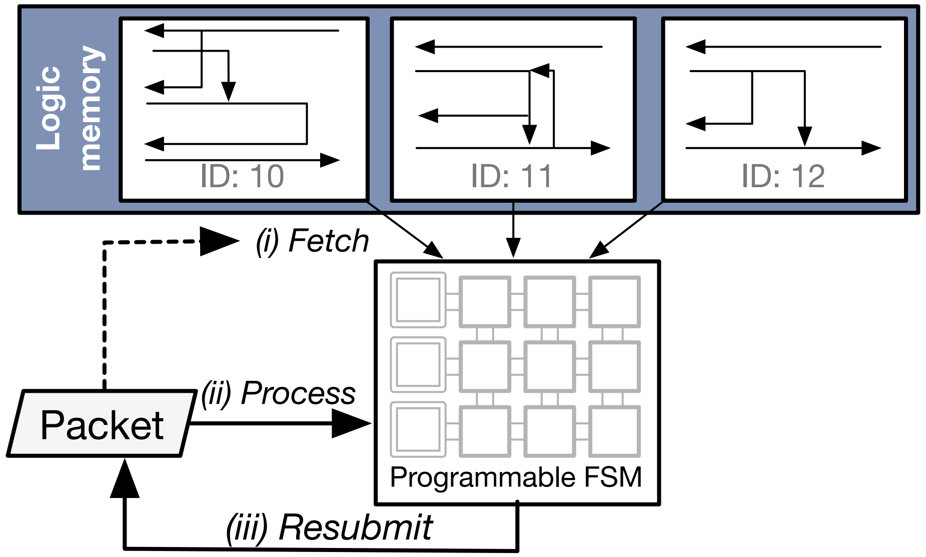
# Tradeoff: Hardware resource consumption

- **Tradeoff for the real-time programmability → Hardware resource usage**
  - Reinhardt requires 3-4 times more hardware resources per pattern than non-programmable hardware designs
  - *3-4 times fewer the number of patterns* than non-programmable circuits

# Tradeoff: Hardware resource consumption

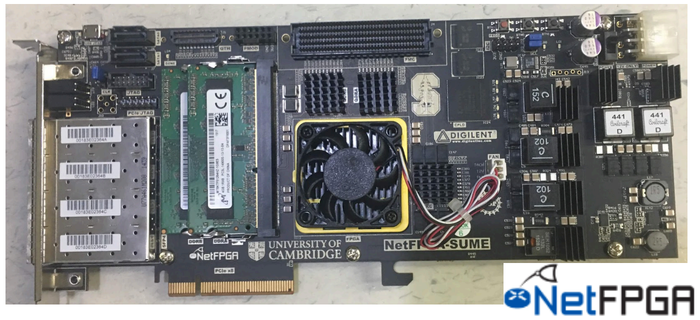
- **Tradeoff for the real-time programmability → Hardware resource usage**
  - Reinhardt requires 3-4 times more hardware resources per pattern than non-programmable hardware designs
  - *3-4 times fewer the number of patterns* than non-programmable circuits

- **Packet resubmitting, the solution from the programmability**
  - *Exploit real-time configurability of Reinhardt*
  - Inspect a packet multiple times with different regex pattern sets back-to-back
  - The number of patterns can be processed as many times as the number of resubmitting beyond the limits of hardware capacity

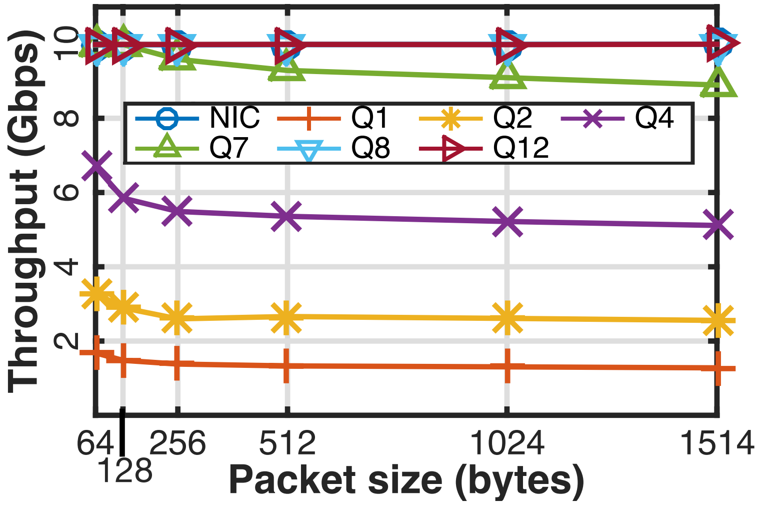


# Evaluation

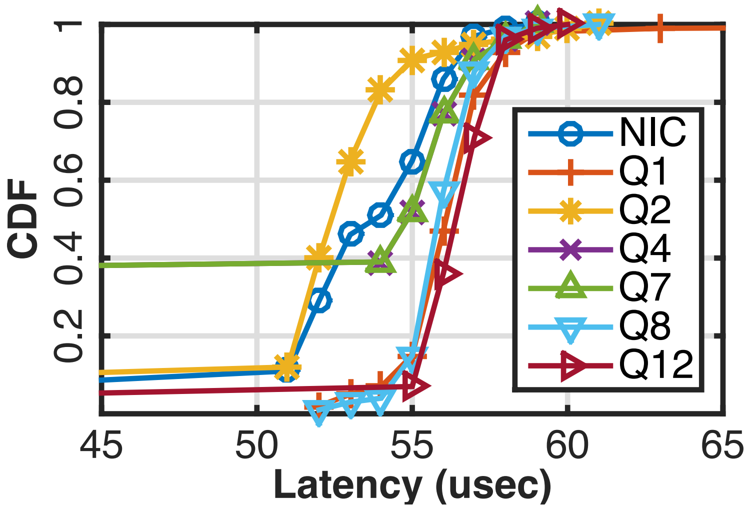
Prototype using NetFPGA-SUME



## Performance

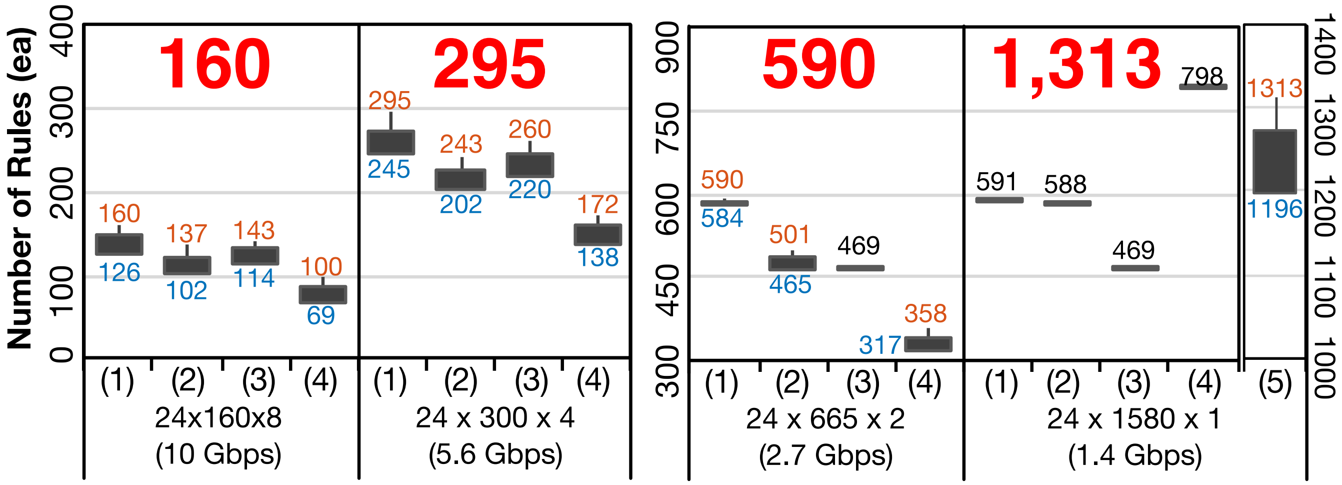


Throughput



Latency

## Pattern capacity



Pattern capacity by Reinhardt processor sizes

Pattern capacity and throughput of prior studies

| Research | # of patterns | Throughput |
|---|---|---|
| Mitra et al. [ANCS'07] | 200 | 12.9 Gbps |
| Yang et al. [ANCS'08] | 267 | 7.5 Gbps |
| MIN-MAX [TPDS-J'13] | 891 | 2.57 Gbps |
| Nakahara et al. [IEICE-TIS'12] | 1,114 | 1.6 Gbps |

# Programmability

## ‣ Pattern deployment time

- Overwhelmingly faster than prior works without service interruptions

**Reinhardt**

| Update time (sec) | # of patterns |
|---|---|
| **0.116** | ≤ 160 |
| **0.186** | ≤ 295 |
| **0.403** | ≤ 590 |
| **0.965** | ≤ 1,313 |

*Approximately 700 us per each*

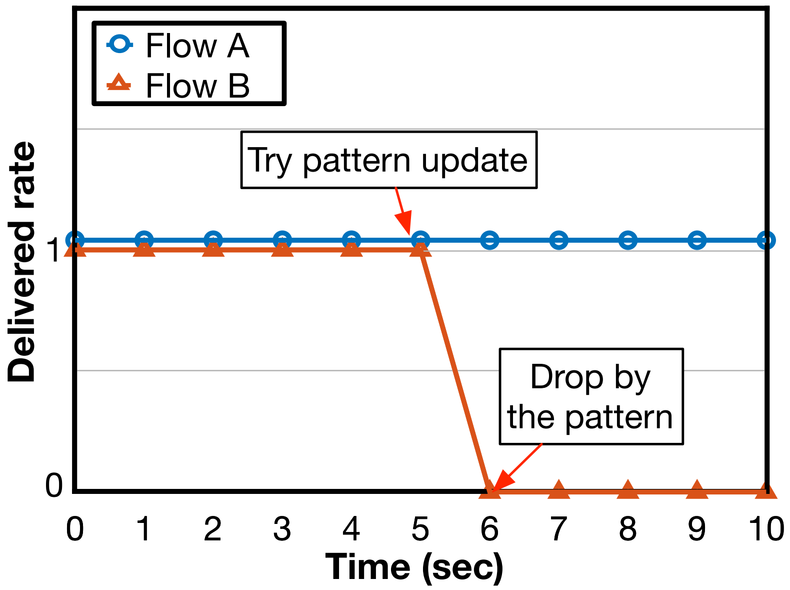
VS

**Known update time**

| # of patterns | Update time (h:mm:ss) | Study |
|---|---|---|
| 200 | **1:38:57** | Johnson et al. [CASES '01] |
| 310 | **1:47:00** | Bisop et al. [ARC'07] |
| 760 | **1:52:00** | Ganegedara et al.[FPL'10] |
| 1,504 | **4:53:50** | Sourdis et al.[SPS-J'08] |

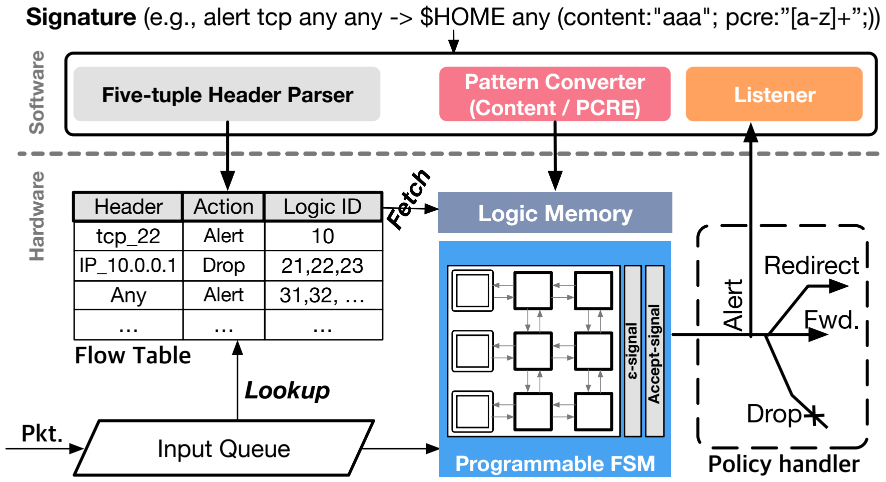
## ‣ Pattern update responsiveness

- **The new signature works instantly**
  - *Flow B is dropped immediately*
- **The device is up and running**
  - *Flow A is delivered continuously*



# Reinhardt deployment

- **Standalone NIDS/IPS device**
  - Deploying into a data plane (e.g., DPX)
  - Achieve 10 Gbps throughput and low-latency in processing DPI
  - Cover about 5,500 signatures

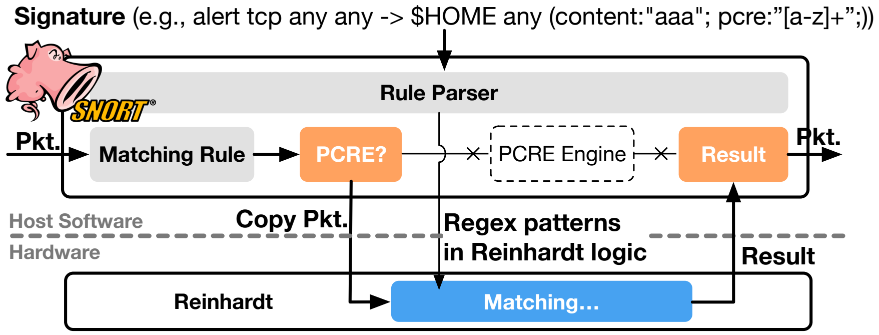


**Signature** (e.g., alert tcp any any -> $HOME any (content:"aaa"; pcre:”[a-z]+”;))

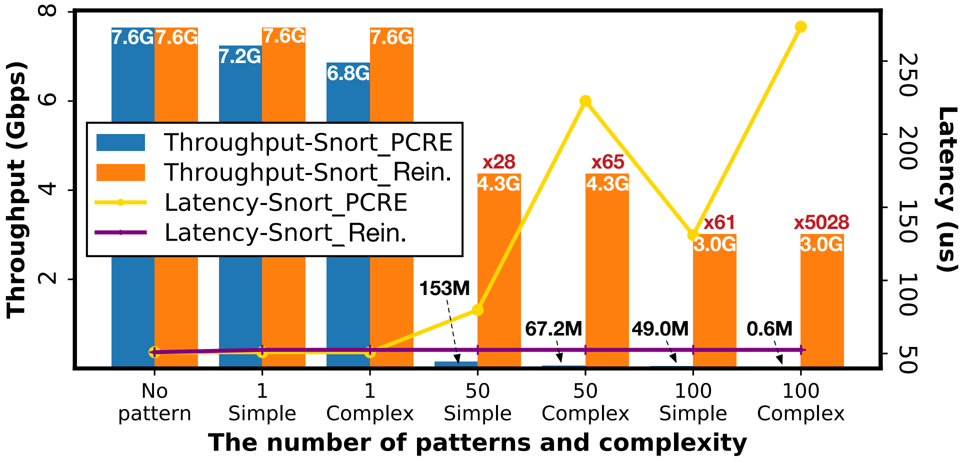
| Header | Action | Logic ID |
|---|---|---|
| tcp_22 | Alert | 10 |
| IP_10.0.0.1 | Drop | 21,22,23 |
| Any | Alert | 31,32, … |
| … | … | … |

Flow Table

- **Accelerating Snort NIDS**
  - Perform pattern matching on Reinhardt instead of the PCRE engine
  - Increase up to x65 (x5028)
    - 7.6 - 3.0 Gbps
    - *The degradations mostly come from Snort IDS itself, not Reinhardt*



**Signature** (e.g., alert tcp any any -> $HOME any (content:"aaa"; pcre:”[a-z]+”;))



# Summary

- **DPI is the key feature in security inspection**
  - Unfortunately, its pattern matching is a major bottleneck point in performance
  - Hardware acceleration? → Poor updatability → Not suitable modern environment
- **Reinhardt: Real-time reconfigurable hardware regex processor**
  - Achieve line-rate performance with low-latency
  - Enable high-programability comparable to software solutions in DPI.

**Reinhardt presents high-performance and high-programable DPI for a dynamic network environment**

# Thank you!

Contact: Taejune Park (taejune.park@jnu.ac.kr)