d-GUARD: THWARTING DENIAL-OF-SERVICE ATTACKS VIA HARDWARE MONITORING OF INFORMATION FLOW USING LANGUAGE SEMANTICS IN EMBEDDED SYSTEMS

GARETT CUNNIGHAM, HARSHA CHENJI, DAVID JUEDES, AND AVINASH KARANTH

School of Electrical Engineering and Computer Science
Ohio University, Athens, OH 45701
Email: juedes@ohio.edu;
karanth@ohio.edu;
chenji@ohio.edu

29th Asia and South Pacific Design Automation Conference (ASP-DAC 2024)
Incheon, South Korea
January 22-25, 2024

OUTLINE

- Motivation
- d-GUARD
  - Language Abstractions
  - Denial-of-Service Prevention Policies
- Results
- Conclusions & Future Work

MOTIVATION (1/2)

- Embedded systems security is compromised when malicious code exploits pre-existing software flaws
  - Code injection attacks, return-oriented programming, buffer overflows
- Enforcing security policies in software can be expensive, so several recent work have enforced secure policies in hardware
  - PHMon, GARUDA[1], PUMP, Aker, DAGGER[2] and others
- However, several of the prior work are static and cannot support dynamically reconfigurable policies

1. Seaghan Sefton, Taiman Siddiqui, Nathaniel St. Armour, Gordon Stewart and Avinash Karanth Kodi, "GARUDA: Designing Energy-Efficient Hardware Monitors from High-Level Policies for Secure Information Flow," **IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems (TCAD),** vol. 37, no. 11, pp. 2509-2518, Nov 2018.
2. Garrett Cunningham, David Juedes, Gordon Stewart, Harsha Chenji and Avinash Karanth, "DAGGER: Exploiting Language Semantics for Program Security in Embedded Systems," *24th International Symposium on Quality Electronic Design (ISQED)*, San Francisco, CA, April 5-7, 2023.

MOTIVATION (2/2)

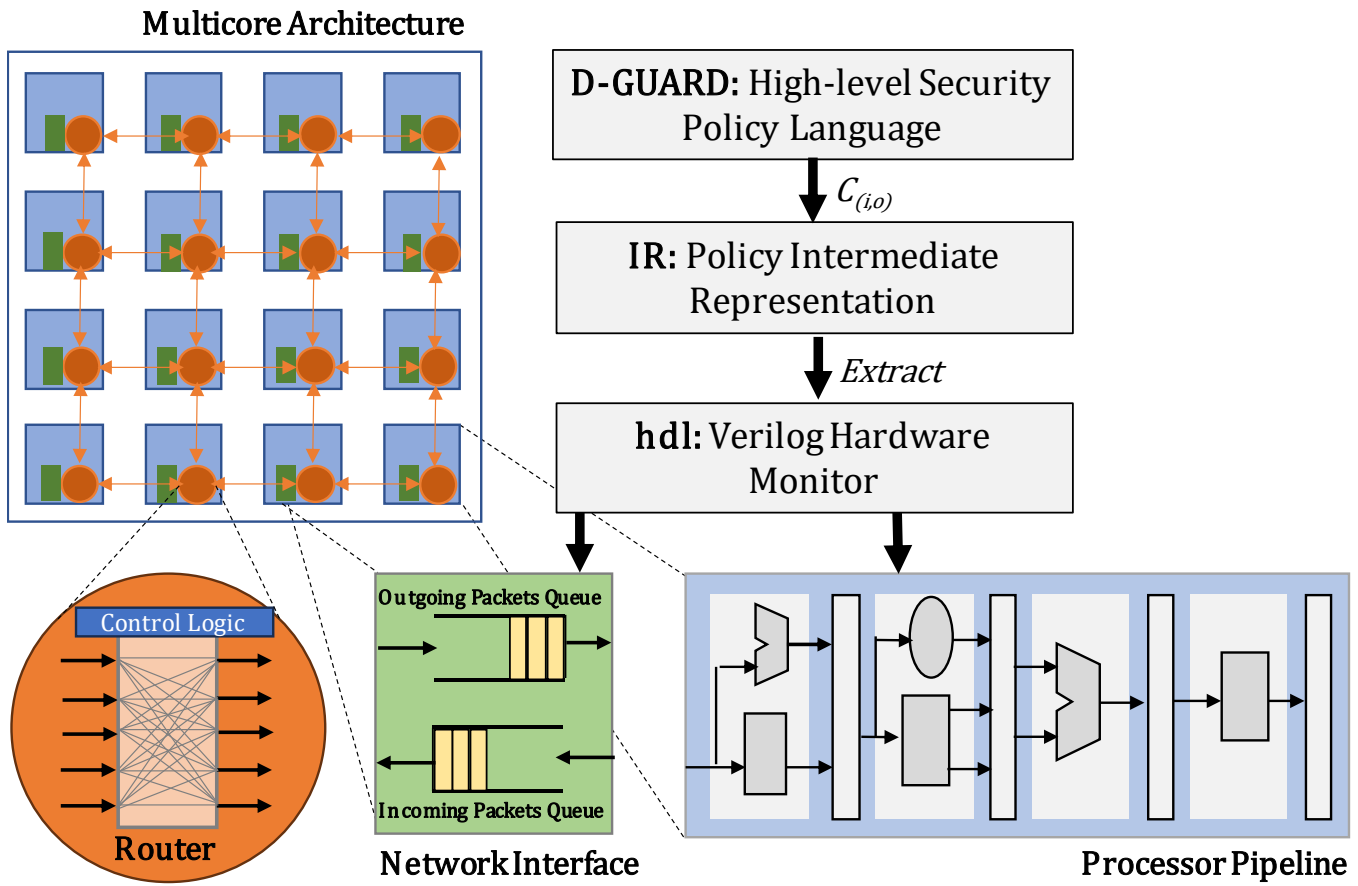
- Moreover, prior hardware monitors target only the processor core and pipelines
- Prior work have considered attacks on interconnection network (glue that connects all system components) such as Network-on-Chip (NoC)
- Attacks such as Denial-of-Service (DoS) are thwarted by rate controlling mechanisms[1]

No prior work has addressed both processor core AND Network-on-Chips (NoCs)

1. S. Charles, Y. Lyu, and P. Mishra, "Real-time Detection and Localization of Distributed DoS Attacks in NoC-based SoCs," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 39, no. 12, pp. 4510–4523, 2020.

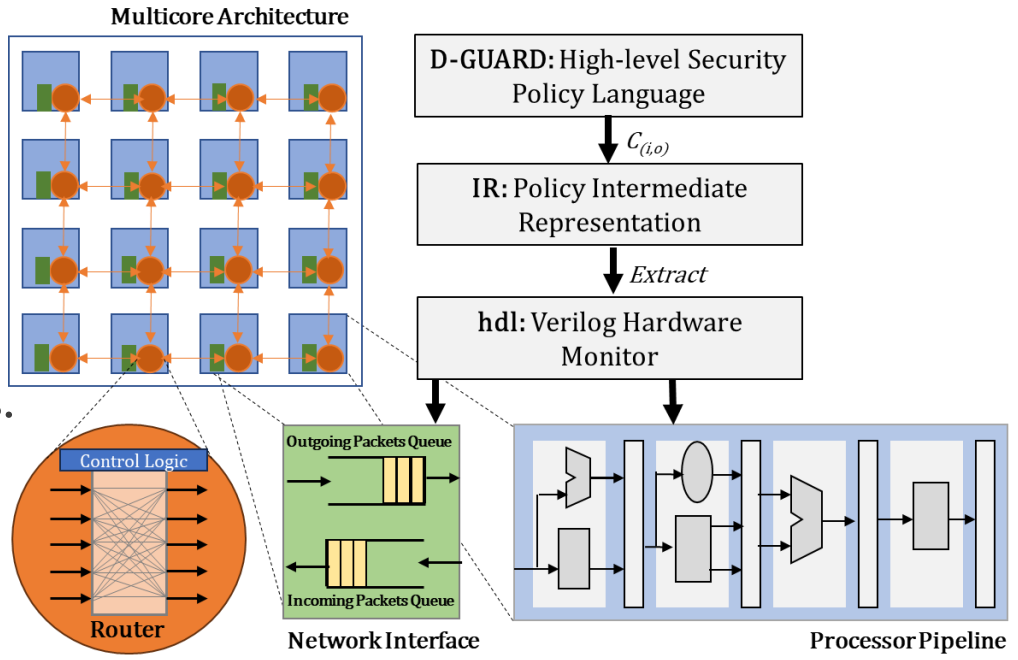
D-GUARD: OUR APPROACH

- A high-level programming language for both processor pipelines AND NoC architecture
- Design monitors in high level language that can be compiled/synthesized to Verilog

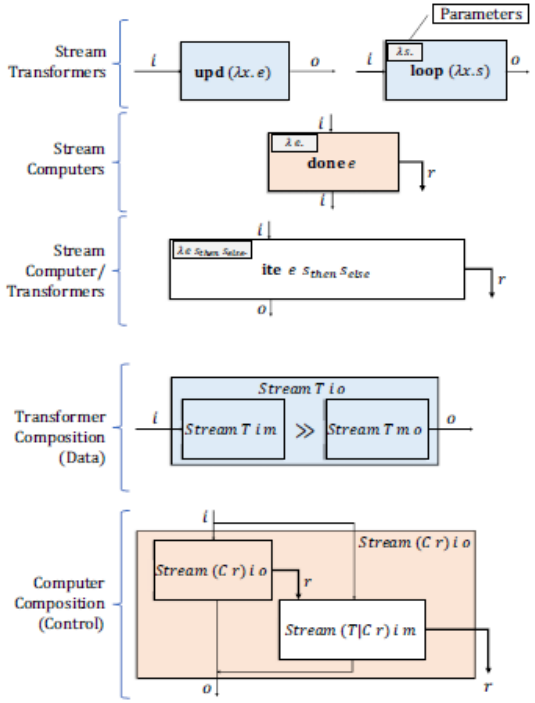


D-GUARD - LANGUAGE

- Monitors are programmed at a high level and compiled to synthesizable Verilog.
- Coq as a carrier language:
  - Leveraged for writing functions and handling computations.
  - Encourages formal verification of policies.
- Monitors as data bit-streams:
  - Monitors continuously take in data and transform it.
  - D-GUARD semantics support dynamically reconfiguring such streams.



D-GUARD LANGUAGE



$\vdash \text{upd } (\lambda x.\ e) : \text{Stream T } i\ o,$
$\quad if\ (x : i) \vdash_e e : o$

$\vdash \text{loop } (\lambda x.s) : \text{Stream T } i\ o,$
$\quad if\ (x : r) \vdash s : \text{Stream } (\text{C } r)\ i\ o$

$\vdash \text{done } e : \text{Stream } (\text{C } r)\ i\ i,$
$\quad if\ \vdash_e e : r$

$\vdash \text{ite } e\ s_{then}\ s_{else} : \text{Stream t } i\ o,$
$\quad if \begin{cases} \vdash s_{then} : \text{Stream } t_1\ i\ o \\ \vdash s_{else} : \text{Stream } t_2\ i\ o \\ t = t_1 \oplus t_2 \end{cases}$

$\vdash s_1 \gg s_2 : \text{Stream } t\ i\ o,$
$\quad if \begin{cases} \vdash s_1 : \text{Stream } t_1\ i\ m \\ \vdash s_2 : \text{Stream } t_2\ m\ o \\ t = t_1 \oplus t_2 \end{cases}$

$\vdash x \leftarrow s_1;\ s_2 : \text{Stream } t\ i\ o,$
$\quad if \begin{cases} \vdash s_1 : \text{Stream } (\text{C } r)\ i\ o \\ (x : r) \vdash s_2 : \text{Stream } t_2\ i\ o \\ t = (\text{C } r) \oplus t_2 \end{cases}$

- Dynamic streams:
  - Dynamic streams allow "halting" conditions for policies, after which a new policy can take control.
- Stream staging, `x <- s1; s2`:
  - Enforces the policy `s1` until it halts and returns a value to `x`. Then reconfigures to `s2`.
- Stream looping, `loop(λx. s)`:
  - Effectively staging a stream with itself. Enforces stream `s` until it halts, stores the return value in `x`, and reiterates `s`.

DENIAL-OF-SERVICE POLICY FOR NOC ARCHITECTURES

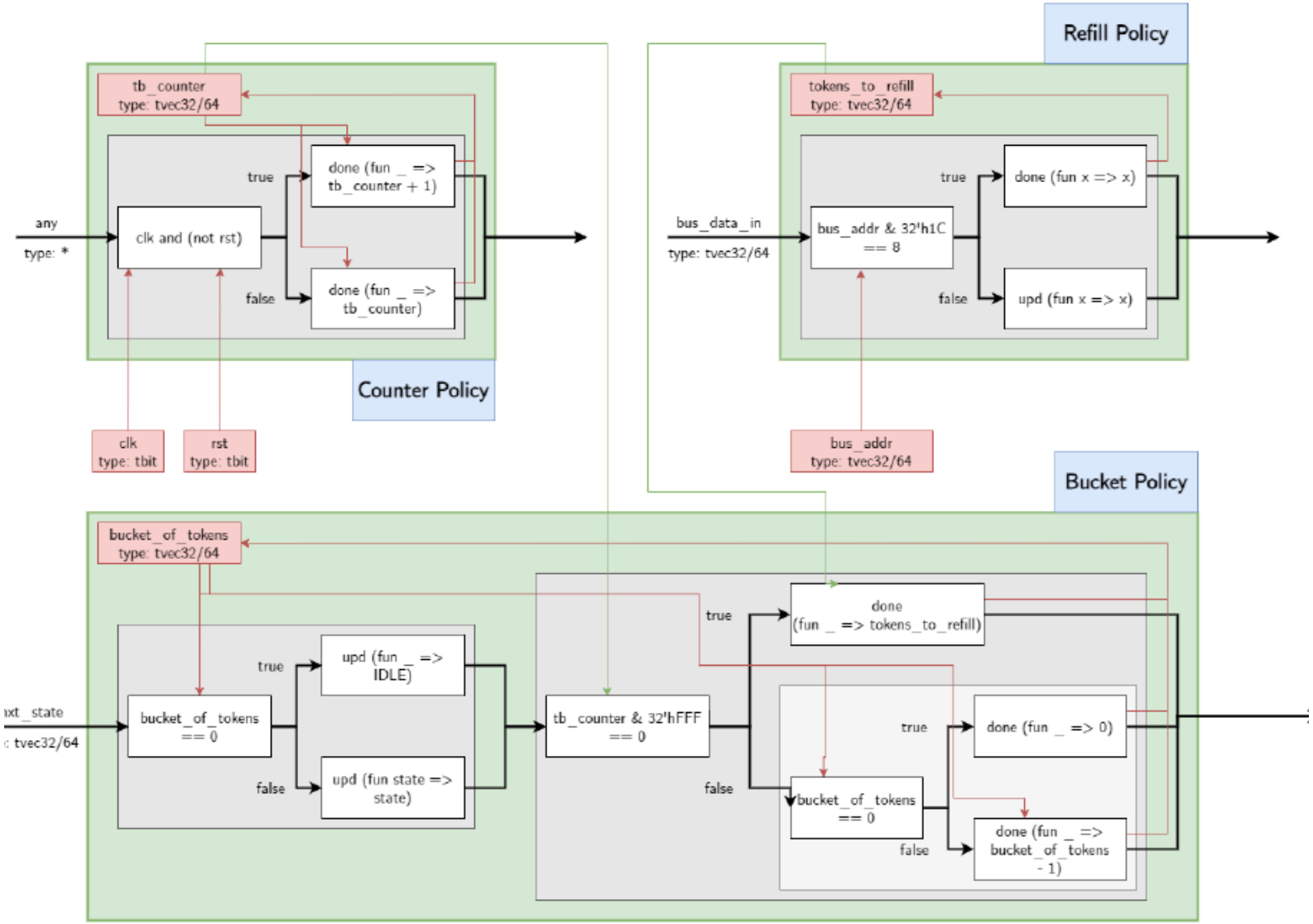
- To thwart DoS attack, we throttle excessive packets into the network using a token-bucket policy
  - These policies are expressed as bucket, counter and refill policies

```
Definition bucket
           (tb_counter tokens_to_refill : tvec32)
           : stream T tbit tbit ≜
     loop (λ bucket_to_tokens : tvec32 ⇒
           ite (bucket_of_tokens == 0)
                 then (upd (λ _ ⇒ IDLE))
                 else (upd (λ x ⇒ x))
           >> ite ((tb_counter & 32'h00000FFF) == 0)
                 then (done (λ _ ⇒ tokens_to_refill))
                 else (ite (bucket_of_tokens == 0)
                       then (done (λ _ ⇒ 0))
                       else (done
                             (λ _ ⇒ bucket_of_tokens - 1)))
```

```
Definition counter (clk rst : tbit)
           : stream T tbit tbit ≜
     loop (λ tb_counter : tvec32 ⇒
     ite (clk and (not rst))
           then (done (λ _ ⇒ tb_counter + 1))
           else (done (λ _ ⇒ tb_counter)))
```

```
Definition refill (bus_addr : tvec32)
           : stream T tvec32 tvec32 ≜
     loop (λ tokens_to_refill ⇒
     ite ((bus_addr & 32'h0000001C) == 8)
           then (done (λ x ⇒ x))
           else (upd (λ x ⇒ x)))
```

DOS PREVENTION POLICY DIAGRAM



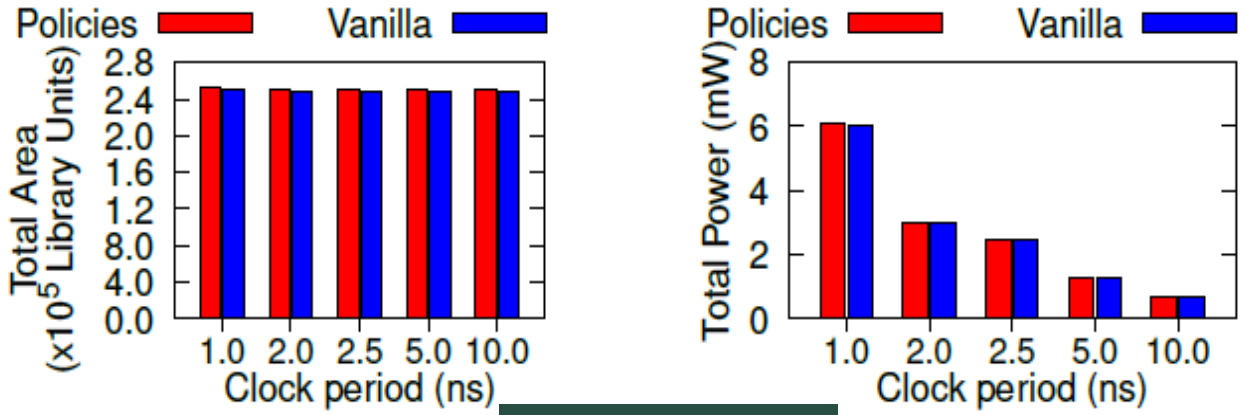
PROCESSOR AND NOC POLICIES IMPLEMENTED

	Policy	Behavior	Location
Pipeline	`leak`	Allow only writes to a fixed memory location, preventing all reads.	ID / EX
	`sjsfi`	Combination of `secjmp` (Fig. 3) and SFI policy that forces addresses into a fixed, safe range.	ID / EX
	`shadow`	On function calls, push return addresses onto a 32-deep stack. On return, check the proposed address against the stack, triggering a violation in the case of mismatch.	ID / EX / MEM
	`taint`	Taint memory addresses as write-only. If a read instruction accesses a tainted memory address, a violation is triggered.	ID / EX / MEM
Network	`bucket`	Maintain a bucket of tokens and halt packet transmission if the bucket reaches 0. Decrement tokens from the bucket as packets are processed, then determine refills based on data from `counter` and `refill`.	Network Interface (NI)
	`counter`	Count clock cycles if reset bit is not active.	CPU Clock
	`refill`	Read bus data and save incoming values specifying the number of tokens to refill the bucket with.	Wishbone Bus

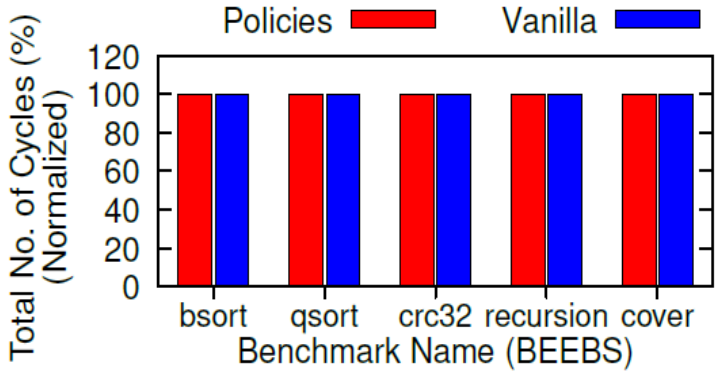
SECURE POLICIES

- We implemented the following secure policies in our work –
  - Data Leak, Secure jump and software fault isolation (SJSFI), Shadow stack, and Taint Tracking
  - NoC policies of bucket, counter and refill
- We used OptimSoC where we synthesized the policies through Synopsys Design Compiler with mflogen framework using 14 nm educational library as well as the freePDK-based Nangate 45 nm library
- We consider baseline design (*Vanilla*) of a single core that implements 32-bit OpenRISC 1000 ISA with 32 MB of RAM
- Secure *Policies* were implemented in the pipeline and synthesized on BEEBS benchmark suite (bsort, qsort, crc32, recursion and cover)

RESULTS - PROCESSOR



Synopsys 14nm

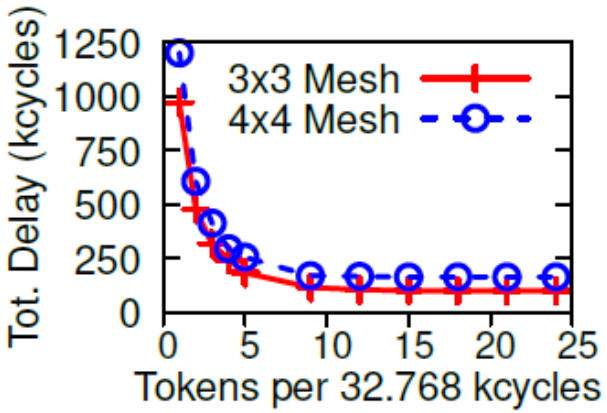


- Results show 1% overhead for policies across libraries for clock speeds ranging from 1 GHz – 100 MHz
- Power consumption scales with higher clock periods

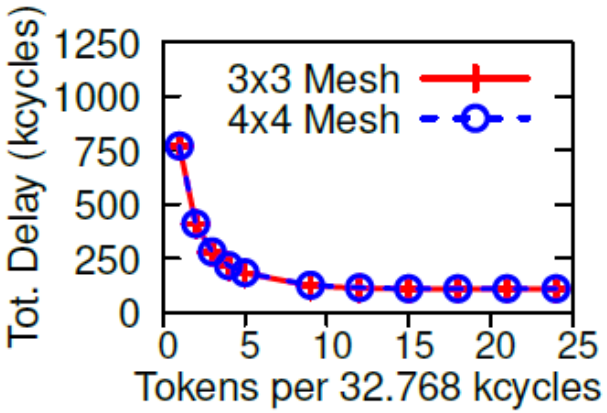
- No extra clock cycles for implementing the policies, no processing overhead

RESULTS - NOC

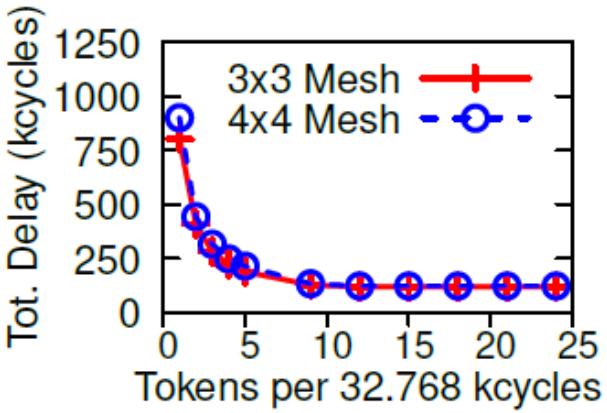
- We measure total delay for a predetermined number of packets to reach the destination for 4 synthetic traffic patterns for 3 x 3 and 4 x 4 mesh
  - Pattern 0 – the destination for every tile is top-left tile (hot-spot traffic)
  - Pattern 1 – tile *i* sends traffic to tile (*i*+*1*) (neighbor)
  - Pattern 2 – tile i chooses as random tile as destination (uniform random)
  - Pattern 3 – only one tile sends traffic to top-left, others are dormant (zero load)



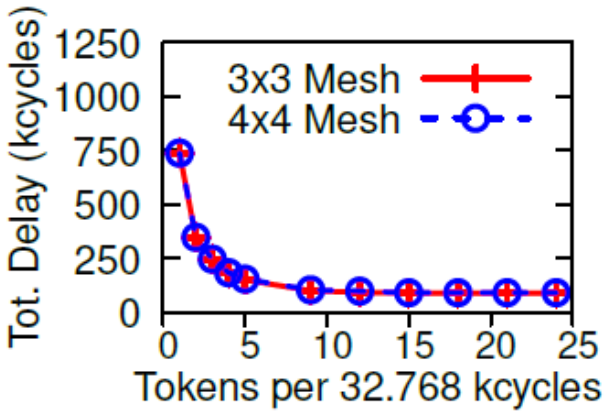
(a) Traffic Pattern 0



(b) Traffic Pattern 1

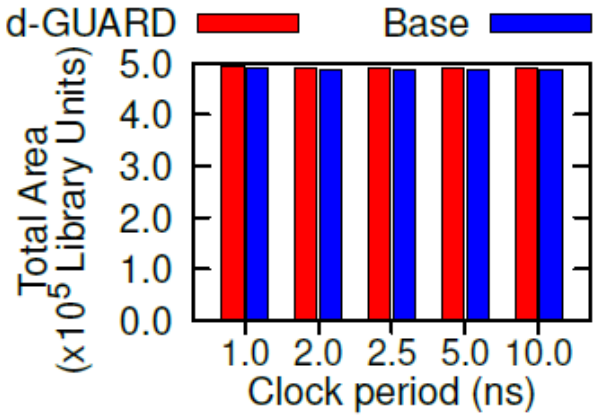


(c) Traffic Pattern 2

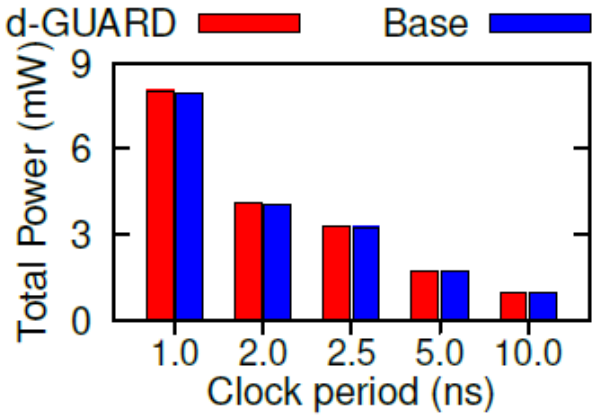


(d) Traffic Pattern 3

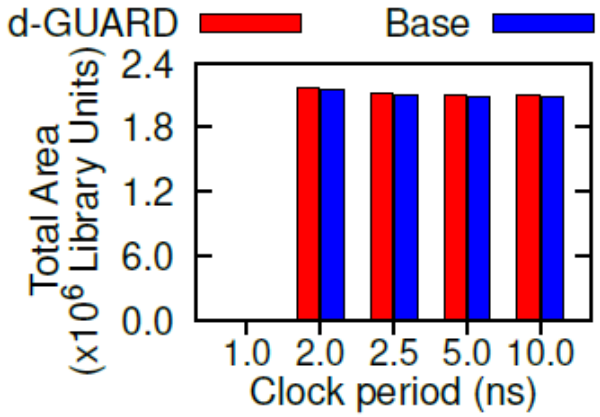
POWER AND AREA SYNTHESIS FOR NOC



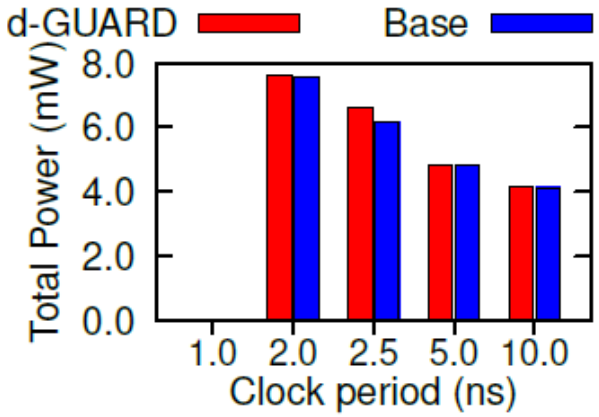
(a) Synopsys 14nm



(b) Synopsys 14nm



(c) Nangate 45nm



(d) Nangate 45nm

- Negligible area and power overhead by the implementation of hardware policies for NoC

CONCLUSIONS & FUTURE WORK

- Enforcing safety guarantees on modern embedded systems comes with non-trivial performance cost
- d-GUARD offers flexibility of implementing dynamic policies that can be compiled to Verilog for both processor core and NoC
- Policies written in d-GUARD can be verified in CoQ assistant proof further validating the design and implementation of the policies
- Future designs will expand to implementing secure policies for KAMI and RISC-V processors

THANK YOU

QUESTIONS?

d-GUARD: THWARTING DENIAL-OF-SERVICE ATTACKS VIA HARDWARE MONITORING OF INFORMATION FLOW USING LANGUAGE SEMANTICS IN EMBEDDED SYSTEMS

GARETT CUNNIGHAM, HARSHA CHENJI, DAVID JUEDES, AND AVINASH KARANTH

School of Electrical Engineering and Computer Science
Ohio University, Athens, OH 43110
Email: juedes@ohio.edu