Lightweight and Trust-aware Routing in NoC-based SoCs

Subodha Charles and Prabhat Mishra
Department of Computer and Information Science and Engineering
University of Florida, Gainesville, Florida, USA

***Abstract*—Increasing System-on-Chip (SoC) design complexity coupled with time-to-market constraints have motivated manufacturers to integrate several third-party Intellectual Property (IP) cores in their SoC designs. IPs acquired from potentially untrusted vendors can be a serious threat to the trusted IPs when they are connected using the same Network-on-Chip (NoC). For example, the malicious IPs can tamper packets as well as degrade SoC performance by launching DoS attacks. While existing authentication schemes can check the data integrity of packets, it can introduce unacceptable overhead on resource-constrained SoCs. In this paper, we propose a lightweight and trust-aware routing mechanism to bypass malicious IPs during packet transfers. This reduces the number of re-transmissions due to tampered data, minimizes DoS attack risk, and as a result, improves SoC performance even in the presence of malicious IPs. Experimental results demonstrate significant improvement in both performance and energy efficiency with minor impact on area overhead.**

***Keywords*-system-on-chip; network-on-chip; security**

I. Introduction

Reusable hardware Intellectual Property (IP) based System-on-Chip (SoC) design has emerged as a pervasive design practice in the industry to dramatically reduce design/verification cost while meeting aggressive time-to-market constraints. Growing reliance on these pre-verified hardware IPs, often gathered from untrusted third-party vendors, severely affects the security and trustworthiness of SoC computing platforms [1; 2]. Since the malicious third party IPs share the same Network-on-Chip (NoC) with secure IPs, malicious IPs can adversely affect the communication between the secure IPs. Figure 1 shows an NoC-based SoC divided into secure and non-secure zones similar to the architecture proposed in ARM TrustZone architecture [3]. An IP in one secure zone (top left) communicates secure information with a secure IP in the other zone (bottom right). Since the packets traverse through the non-secure zone, a malicious IP can tamper the packets.

Consider a scenario where the integrity of exchanged data is ensured using a message authentication code (MAC) . The sender IP sends a packet together with an authentication tag, and the receiver re-computes the tag to check for data integrity. If it doesn't match, the packet has been tampered during communication, and a re-transmission is required. This method of error correction is widely employed in NoC-based SoCs [4]. However, re-transmissions due to corrupt packets can lead to several problems:

- Increased latency because of re-transmission as well as additional stall cycles introduced by the IP cores while waiting for the requested data.
- This can increase the number of packets traversing the network, and as a result, increase energy consumption and performance penalty [5].
- In MAC-then-encrypt protocols [6][1], authentication tag is computed on the plaintext, appended to the data, and then tag and plaintext are encrypted together. When MAC is computed in this way, the receiver IP has no way of knowing whether the message is indeed authentic or tampered until the message is decrypted. Therefore, the resources spent to decrypt a tampered packet is wasted.

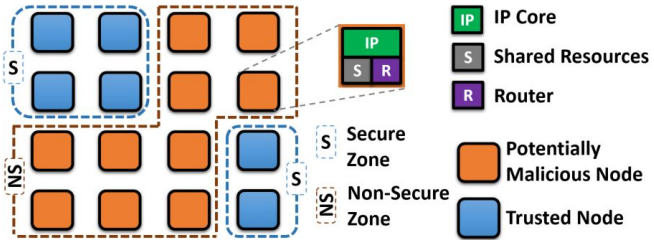


Figure 1: Overview of a typical SoC architecture with secure and non-secure zones.

Systematic exploitation of error correction protocols, such as the one explained above, can lead to Denial-of-Service (DoS) attacks. For example, a malicious IP can corrupt data on purpose and cause continuous re-transmissions leading to a DoS attack [7]. Specifically, our threat model is as follows.

Threat Model: Figure 1 shows a standard NoC-based many-core architecture with IPs connected in a Mesh topology. Each IP connects to a router via a network interface. The network interface accommodates the authentication scheme which implements MAC-based authentication [8]. A packet originating from a source IP (*src*) in a secure zone has to traverse through the non-secure zone in order to reach the destination IP (*dest*) in another secure zone. The IPs in the non-secure zone are potentially malicious. In reality, out of all the potentially malicious IPs, only a small fraction is actually malicious. We call them malicious IPs (MIP) in this paper. If the packet traverses through such an MIP, it can tamper with the packet and therefore,

[1]MAC-then-encrypt is the standard method used in TLS [6].

at *dest*, the authentication tag computation will not match and the packet will be dropped. The *src* will re-transmit the packet since a response is not received from the *dest* within the time-out period. The problem of minimizing this impact gets aggravated due to two challenges. (1) The MIP will not always behave maliciously. In other words, it will tamper packets only in sporadic intervals. (2). Since the *src* depends on the response from the *dest* to know whether the packet was received or not, the MIP can tamper the packet between *src* and *dest* or tamper the response packet between *dest* and *src*, and both of these scenarios lead to the same outcome from the *src*'s point of view. We consider both of these challenges when proposing our solution. A similar threat model was used in a previous study that proposed countermeasures for DoS attacks [7].

Previous work on securing NoC explored lightweight encryption and authentication [8], route randomization, data scrambling and node obfuscation [9]. DoS attack mitigation has also been studied in the context of NoC [10]. However, none of these solutions are capable of mitigating the performance and energy overhead caused by MIPs during an attack under the given threat model. In this paper, we propose a trust-aware routing protocol that avoids MIPs when two secure IPs are communicating with each other. Our proposed approach leads to less re-transmissions, and as a result, improved performance and energy efficiency. Trust-aware routing can complement existing NoC attack detection and mitigation techniques by allowing on-chip communication even in the presence of an adversary while minimizing the energy and performance overhead.

Our major contributions can be summarized as follows;

1) We propose a "trust model" that effectively calculates trust between neighboring routers and propagates trust values through the NoC.
2) We have developed a routing protocol that uses the trust values between routers to make routing decisions such that the MIPs are avoided by packets when routing from source to destination.
3) We have evaluated the effectiveness of our approach using both real benchmarks and synthetic traffic patterns to demonstrate that it leads to significant improvement in both performance and energy efficiency.

The paper proposes a routing protocol that avoids MIPs in the NoC rather than detecting them. Therefore, our approach can be used together with any MIP detection mechanism while increasing the overall performance and energy efficiency. The remainder of the paper is organized as follows. Section II discusses the related approaches to highlight the need for our proposed work. Section III presents our proposed NoC trust model. Section IV describes our trust-aware routing protocol that utilizes the NoC trust model. Section V presents the experimental results. Finally, Section VI concludes the paper.

II. Background and Motivation

In this section, we first survey the related approaches. Next, we demonstrate why the existing approaches can lead to unacceptable design overhead.

A. Related Work

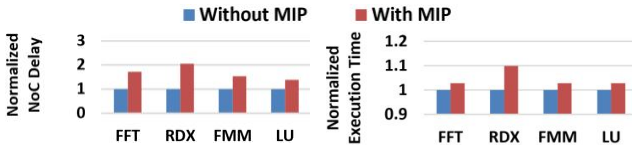
Previous research on securing the NoC has proposed lightweight security schemes [11; 12], DoS attack mitigation techniques [10; 13; 14], and methods to prevent side-channel attacks [15; 9]. Most of the methods try to exploit the unique characteristics offered by the structure of the NoC and traffic transferred through the NoC when developing security schemes [16]. The SurfNoC architecture utilized the unique nature of NoC communication by time multiplexing the links for different domains [17]. This allows packets of one type to traverse the network without interfering with packets from other domains, and as a result, it improves performance and minimizes DoS attack risk. Solutions to address the challenges associated with side channel attacks on NoC try to use route randomization, data scrambling and node obfuscation [9]. ARM presented the TrustZone architecture which divides the NoC as secure and non-secure zones [3]. However, to the best of our knowledge, none of the previous work address the scenarios outlined in this paper.

The concept of "trust" between inter-connected entities has been studied before in the networking domain [18]. It tries to enhance the security of distributed networks such as ad-hoc networks by identifying attacks against trust evaluation systems and building defense techniques based on trust models. Concepts such as "web-of-trust" and "pretty-good-privacy" (PGP), which are widely used in internet communication, establish a similar notion that discuss the authenticity of binding a public key to the owner [19]. The OpenPGP email protocol is one such example [20]. Being tightly coupled together, entities on an NoC interact heavily with each other when facilitating on-chip communication. Yet, the concept of trust when packets are routed through the NoC has not yet been explored.

B. Motivation

Lightweight authentication schemes implemented on NoC-based SoCs, try to provide desired security while consuming minimum number of cycles. However, if the MAC fails to match at the receiver's end, the *src* has to re-transmit again, leading to wasted effort in repeated NoC traversal and MAC calculation [4]. The challenge is aggravated in MAC-then-encrypt protocols because MAC can only be calculated and matched after decryption is done. If the packet is tampered, time and energy spent on decryption is wasted. To analyze these overheads, we ran FFT, RADIX (RDX), FMM and LU benchmarks from the SPLASH-2 benchmark suite on an 8×8 Mesh NoC-based SoC with 64 cores which implements a MAC-then-encrypt security protocol

and XY routing protocol. The behavior of an MIP was simulated by one of the IPs along the routing path dropping n consecutive packets after every p (period) packets. NoC delay (total NoC traversal delay for all packets) including encryption/decryption and MAC calculation time, execution time and number of packets injected were recorded with and without the presence of an MIP. The encryption/decryption and authentication process is assumed to take 20 cycles per transmission [21]. Results are shown in Figure 2a, Figure 2b, and Figure 2c, respectively. We observed 67.2% increase in NoC delay and a 4.7% increase in execution time on average across all benchmarks. The number of packets injected increased by 60.1%. The combination of execution time and number of injected packets directly affect the energy consumption since both time spent to execute the task and dynamic power are increased.



(a) NoC delay (b) Execution time

(c) No. of packets injected

Figure 2: NoC delay, execution time and number of packets injected comparison with and without the presence of an MIP when $p = 20$ and $n = 14$.

It is evident that in addition to checking data integrity, a mechanism to avoid MIPs when routing through the non-secure zone can lead to less re-transmissions, and as a result, increased performance and energy efficiency.

III. NoC Trust Model

This section describes our proposed trust model to quantitatively measure the trust between two nodes. Trust is established between two nodes to handle packets without tampering with the data. In particular, one node trusts the other node to perform the intended action on the received packet (in the case of routing, forward the packet to the next hop). In this paper, the first node is referred to as the *producer* (α) and the second node as the *consumer* (β). We introduce the notation $\{producer \rightarrow consumer\}$ ($\alpha \rightarrow \beta$) to denote a trust relationship[2]. Trust can be established in two ways - (1) delegated trust, and (2) direct trust. Direct trust is established when a node calculates trust about one of its neighbors. Trust is said to be delegated when one node recommends a consumer node to another producer node that is not directly connected to the consumer. The recommending node is referred to as *recommender*. Figure 3a shows such an example. In this three node setup, direct trust can be established between B and C, and A and B. But, trust between A and C can only be established via B's recommendation. Therefore, $A \rightarrow C$ has a delegated trust relationship.

[2]The *producer* and *consumer* notations are different from *src* and *dest* since any two routers along the routing path can be producer/consumer whereas *src* and *dest* refer to the origin of the packet and its destination, respectively.

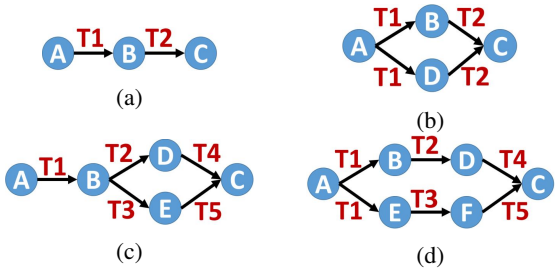


Figure 3: Trust delegation across NoC. The values on the arrows represent the trust. For example, $T1$ in (a) denotes $T^{(a)}_{A \rightarrow B}$ where the superscript (a) corresponds to figure 3a.

To quantify trust between two entities, a measure of trust is required. Keeping a binary value per node (either trusted or not) doesn't capture the entire trust model due to several reasons: (i) Trust can be delegated (in the example in Figure 3a, the amount of trust A places on C depends on how much A trusts B), (ii) a malicious node might not launch an attack at first, but do so after a while or periodically. Therefore, we assign a value (denoted $T_{\alpha \rightarrow \beta}$) between -1 and 1 for each trust relationship ($-1 \leq T_{\alpha \rightarrow \beta} \leq 1$) to indicate a trust value in the "potentially malicious" spectrum. The two bounds are defined as follows:

- When the producer is confident that the consumer will always function correctly: $T_{\alpha \rightarrow \beta} = 1$.
- When the producer is confident that the consumer is definitely malicious: $T_{\alpha \rightarrow \beta} = -1$

In addition to the two bounds, $T_{\alpha \rightarrow \beta} = 0$ implies that the producer has no idea whether the consumer is malicious or not. Therefore, at the beginning of network packet transmission, all trust relationships are initialized to the value of zero. During operation, with information received from nodes, trust values are calculated. It is important to note that, when B recommends C to A (delegated trust), $T^{(a)}_{A \rightarrow C}$ can be established only if $T^{(a)}_{A \rightarrow B} \geq 0$. In other words, A should not trust its enemy to recommend someone as trustworthy. Once this condition is met, we present three axioms such that the trust delegation calculation adheres to those. The remainder of this section describes these axioms (Section III-A) and elaborate how delegated trust (Section III-B) and direct trust (Section III-C) are calculated.

A. Axioms for Trust Delegation

Axiom 1: In delegated trust, trust value between producer and consumer should not be higher than the trust between producer and recommender as well as the trust between recommender and consumer. This can be formalized using Figure 3a;

$$\left|T^{(a)}_{A\to C}\right| \leq min(T^{(a)}_{A\to B}, T^{(a)}_{B\to C}) \tag{1}$$

Axiom 2: Producer receiving the same recommendation about the same consumer via multiple different recommenders should not reduce the trust between producer and consumer. In other words, the producer will be more certain about the consumer or at least maintain the same level of certainty if the producer obtains an extra recommendation that agrees with the producer's current opinion. For example, Figure 3a and Figure 3b show two scenarios where A in first figure establishes trust with C via only one path and in the second scenario, trust with C is established through two same-trust paths.

$$T^{(b)}_{A\to C} \geq T^{(a)}_{A\to C} \geq 0,\ for\ T1 > 0\ and\ T2 \geq 0 \tag{2}$$

$$T^{(b)}_{A\to C} \leq T^{(a)}_{A\to C} \leq 0,\ for\ T1 > 0\ and\ T2 < 0 \tag{3}$$

This holds only if the multiple paths give the same recommendations.

Axiom 3: In a setup similar to Figure 3c, it is possible to receive multiple recommendations from a single node (B). Compared to that, recommendations from independent nodes such as the ones shown in Figure 3d (B and E) should always be trusted more. In other words, recommendations from independent nodes can reduce uncertainty more effectively than the recommendations from correlated nodes. Formally;

$$T^{(d)}_{A\to C} \geq T^{(c)}_{A\to C} \geq 0,\ if\ T^{(c)}_{A\to C} \geq 0 \tag{4}$$

$$T^{(d)}_{A\to C} \leq T^{(c)}_{A\to C} \leq 0,\ if\ T^{(c)}_{A\to C} < 0 \tag{5}$$

B. Delegated Trust Calculation

The calculation of trust from the point of view of any given node should adhere to the above axioms. For the example shown in Figure 3a, we established that the necessary condition is to satisfy Axiom 1. To achieve this, trust can be calculated by concatenation as $T^{(a)}_{A\to C} = T^{(a)}_{A\to B} \cdot T^{(a)}_{B\to C}$. In general;

$$T_{\alpha\to\beta} = T_{\alpha\to\gamma} \cdot T_{\gamma\to\beta} \tag{6}$$

where γ is the recommender. As mentioned before, this can only be calculated if $T_{\alpha\to\gamma} \geq 0$. It can be noticed that if α has no idea about the trustworthiness of γ ($T_{\alpha\to\gamma} = 0$), no matter how much γ trusts β, α won't trust β ($T_{\alpha\to\beta} = 0$).

In case of multi-path trust delegation such as the example in Figure 3b, axioms 2 and 3 have to be satisfied in addition to Axiom 1. When α can establish trust with β via two paths, one via δ and another via ϵ ($\alpha-\delta-\beta$ and $\alpha-\epsilon-\beta$), we combine the ratios of trust concatenation.

$$T_{\alpha\to\beta} = z_1 \cdot (T_{\alpha\to\delta} \cdot T_{\delta\to\beta}) + z_2 \cdot (T_{\alpha\to\epsilon} \cdot T_{\epsilon\to\beta}) \tag{7}$$

where

$$z_1 = \frac{T_{\alpha\to\delta}}{T_{\alpha\to\delta} + T_{\alpha\to\epsilon}},\ and\ z_2 = \frac{T_{\alpha\to\epsilon}}{T_{\alpha\to\delta} + T_{\alpha\to\epsilon}} \tag{8}$$

C. Direct Trust Calculation

We calculate direct trust based on the *sigmoid function* ($\frac{1}{1+e^{-x}}$), where x keeps track of the number of successful transmissions at a given router. Since the sigmoid function ranges between 0 and 1, we scaled it to range between -1 and 1 (Figure 4).

$$S(x) = 2 \cdot \frac{1}{1+e^{-x}} - 1 \tag{9}$$

Assume that α and β are neighbors. Initially, α has no trust information about β. Therefore, $x = 0$, and as a result, $S(x) = 0$. When α learns about β's behavior, it changes the value x and re-calculates $S(x)$. For example, if α gets a positive feedback about β's trust, direct trust is calculated as $T_{\alpha\to\beta} = S(x+\delta)$ where δ is a small positive number. Since $S(x)$ is an increasing function as shown in Figure 4, α's trust about β is now increased. Similarly, to reduce trust, $T_{\alpha\to\beta} = S(x-\delta)$. Therefore, direct trust is calculated as;

$$x = x \pm \delta, \qquad T_{\alpha\to\beta} = S(x) \tag{10}$$

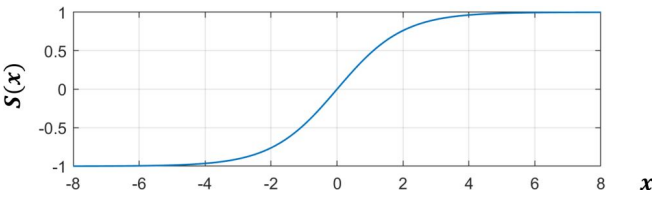


Figure 4: Sigmoid function $S(x)$ variation with input x.

IV. Trust-aware Routing

Once the trust values are established, they are used by our proposed routing protocol. The basic idea is to route packets through highly trusted nodes so that MIPs are avoided. It is important to note that, trust values have to be dynamically updated during SoC execution since MIPs shift between malicious and non-malicious behavior according to our threat model. The following subsections explain in detail how direct trust and delegated trust are updated at each router (Section IV-A and Section IV-B, respectively) and how those trust values are used in routing (Section IV-C).

A. Updating Trust

According to the threat model described in Section I, if a *src* IP doesn't receive a response to the packet sent, it can be because of two reasons;

- **Message was lost between *src* and *dest***: In this case, a response is never received. The *src* times out after a while and re-transmits the packet. The routers along the routing path observe that this is a re-transmission and reduces the direct trust of their next-hop neighbors. Direct trust is reduced since a packet took that path before and it was tampered. Direct trust re-calculation is done every time a re-transmission is observed. Once the trust values go down compared to the other possible paths, the packet takes an alternate path avoiding the MIP according to the routing protocol and is received at the *dest*.
- **Response was lost between *dest* and *src***: This means that the packet was received at *dest*, but the response was not received by *src*. Again, *src* sends a re-transmission which is received by *dest*. *dest* observes that this is an address that was previously served and sends the response again. Again, routers along the path observes that this is a re-transmission and reduces direct trust. This process is repeated until the response is received by *src*. This causes the routers between *src* to *dest* to reduce trust unnecessarily (false negative). However, we don't try to correct it because to do that, *src* has to keep track of all the paths the re-transmitted packets took to reset trust values. Furthermore, the routers should also maintain previous trust values. Therefore, we allow false negatives to happen. With several ongoing communications overlapped between routers, the false negatives will regain trust over time.

Considering these scenarios, we use an event-driven approach to update trust. The overview of our algorithm is shown in Algorithm 1. To keep track of the re-transmissions and to increase/decrease direct trust according to that, we implement a separate data structure at each router-*Communication Table* (ComTable). It stores each pending communication using *src*, *dest*, address of corresponding memory location (*addr*), *timestamp* to indicate when the entry was added to the table and a re-transmission flag (*rtx flag*). When a new packet arrives at a router, it checks to see if there is a pending communication between the same *src* and *dest* by matching *src* and *dest* fields in the packet header to entries in the ComTable (line 1). If yes, it can either be for the same address (line 3) or for a different address. If it is for a different address, it means that the previous communication has completed successfully. If it is for the same address, then it is identified as a re-transmission. The rtx flag is set to indicate this (line 4) and direct trust with the next hop ($getNextHop$ routine elaborated in Section IV-C) is reduced (line 6). If it is a new communication, the rtx flag is checked to see whether the previous communication between the same *src* and *dest* has not been flagged as a re-transmission before (line 9). If it has not been flagged before, the path can be trusted. Then the direct trust with next hop router is increased (line 11) and the trust is delegated (line 12) to other neighbors as explained in Section IV-B. If it has already been flagged as a re-transmission, no further action is taken since it has already been penalized and as a result, direct trust has been reduced in a previous iteration (lines 4-6). In both cases, when it is a new communication, the ComTable is updated by removing the old entry and adding the new one (line 14). If it is the first communication that is passing through that router for that *src* and *dest* pair, a new entry is added in the ComTable (line 18). The ComTable also records a timestamp for each entry. The timestamp is used to stop the exponential growth of the ComTable by removing old entries after a certain time threshold.

One limitation of this model is that it assumes that an IP will only send a second request to the same destination once the first one is served. For architectures that support multiple pending requests, we can easily extend this scheme. The sender maintains a list of pending requests and adds a header bit in the next packet to indicate that this is another request with the same *src*, *dest*, but has a different address. Then, the routers check this bit before removing the previous entry and trust is increased only if this bit is not set. The rest of the methodology remains the same.

B. Delegating trust in the NoC

Once a communication is successfully completed, trust about the next-hop ($T_{\alpha\rightarrow\beta}$) is delegated to nearby routers by each router ($delegateTrust$ routine in Algorithm 1). This is done by broadcasting a packet that contains $T_{\alpha\rightarrow\beta}$ with a pre-defined *time-to-live* (τ) value in the header in all directions except for the direction of the next hop router. In our experiments, we set $\tau = 1$. This causes the trust about the next hop router to be delegated to all other neighbouring routers. An illustrative example of this mechanism is shown in Figure 5. Once router α completes a communication where according to the routing protocol, the next hop router is β, it sends the direct trust value ($T_{\alpha\rightarrow\beta}$) to B, D and E. These three routers now calculate $T_{B\rightarrow\beta}$, $T_{D\rightarrow\beta}$ and $T_{E\rightarrow\beta}$, which are delegated trust values, according to the trust model in Section III-B (Equations 6). As a result, B, D and E learn about the trustworthiness of a router (β) two hops away from them.

It is possible that this delegated trust packet itself is tampered and in that case, delegated trust will not be updated. This has no impact since a delegated trust packet being dropped means an MIP is on that path and its trust value will be negative. Delegated trust is updated only when it comes from a trusted source with a positive trust value according to Equation 6.

C. Routing Protocol

The goal of the routing protocol is to avoid MIPs in the non-secure zone while routing through the most trusted routers. Each router stores the trust values of routers that

Algorithm 1: Updating direct and delegated trust

```
/* This routine is called by each router every time
   a packet arrives.                                    */
/* Input: packet.                                       */
/* Current node is assumed to be α                      */
entry ← checkComTable(packet)
if entry ≠ NULL then
    if entry.addr = packet.addr then
        entry.rtxFlag ← 1
        β ← getNextHop(packet)
        T_{α→β} ← S(x − δ)
    end
    else
        if entry.rtxFlag ≠ 1 then
            β ← getNextHop(packet)
            T_{α→β} ← S(x + δ)
            delegateTrust()
        end
        updateComTable(packet)
    end
end
else
    updateComTable(packet)
end

/* Routine: checkComTable                               */
/* Input: packet                                        */
for entry ∈ comTable do
    if entry.src = packet.src & entry.dest = packet.dest then
        return entry
    end
end
return NULL

/* Routine: updateComTable                              */
/* Input: packet                                        */
for entry ∈ comTable do
    if entry.src = packet.src & entry.dest = packet.dest then
        comTable.delete(entry)
    end
end
newEntry.src ← packet.src, newEntry.dest ← packet.dest
newEntry.addr ← packet.addr, newEntry.rtxFlag ← 0
newEntry.timestamp ← 0
comTable.add(newEntry)
```

are one (direct trust) and two hops away from it (delegated trust). When a router receives a packet, it first updates the trust values according to Algorithm 1. Next, the packet is forwarded to the next hop. Both forwarding and Algorithm 1 use the $getNextHop$ routine, which works as follows;

- Read the *dest* ID of the packet
- Compare *dest* and current router IDs
  - If *dest* is located in the same row or column as the current router, next hop is the neighbouring router along that row or column towards *dest*.
  - Else, check the sum of trust values of routers one and two hops towards the *dest*, and select the neighbor along the path which has the largest trust value as the next hop. If two paths have the same largest trust value, randomly pick one.

For example, in Figure 5, assume a packet arrives at router B with the destination G. Since B is not in the same row or column as G, next hop is selected based on trust values. When considering routers that are one and two hops away from B in the direction of G, there are three possible paths:

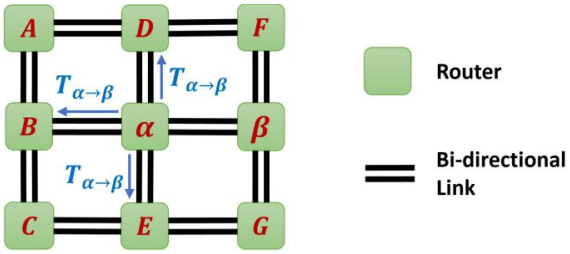


Figure 5: NoC with routers connected in a 3×3 Mesh topology. Illustrative example showing that once a communication completes, the direct trust between α and β ($T_{\alpha\to\beta}$) is delegated to nodes one hop away from α.

$B-\alpha-\beta$, $B-\alpha-E$, and $B-C-E$. Therefore, B calculates $max(T_{B\to\alpha}+T_{B\to\beta}, T_{B\to\alpha}+T_{B\to E}, T_{B\to C}+T_{B\to E})$ and if $T_{B\to C}+T_{B\to E}$ gives the maximum trust value, next hop is C. Considering nodes that are always towards the destination (in the example, B only considers α and C as next hops) ensures that the packet traverses the network following only one of the shortest paths. This together with the use of bi-directional links ensures the deadlock and livelock free nature of the routing algorithm. Our routing protocol is identical to the congestion-aware routing protocol presented in [22] except that we are using trust values instead of congestion values. Therefore, we can show that our routing protocol is also is also deadlock-free and livelock-free using the same arguments from [22].

It is important to note that our trust-aware routing protocol works even if all the IPs in the non-secure zone are malicious or, MIPs isolate the untrusted zone into several disconnected sub zones of secure IPs. If all the neighbors of a router has a trust value of -1 (all routers are malicious) it will still be routed through that path since -1 is the largest value. Therefore, the packet is guaranteed to reach the destination, but might be corrupted. If there is a path from source to destination that does not contain an MIP, our approach is guaranteed to find that path and deliver the packets without being corrupted.

V. Experimental Results

This section explores the feasibility and effectiveness of our approach by presenting experimental results and discussing the overheads associated with it.

A. Experimental Setup

We modeled an 8×8 Mesh NoC-based SoC with 64 cores using the gem5 cycle-accurate full-system simulator [23; 24]. The interconnection network was built on top of "GARNET2.0" model that is integrated with gem5 [25]. Each router in the Mesh topology connects to four neighbors and a local IP via bidirectional links. Each IP connects to the local router through a network interface, which implements the MAC-then-encrypt protocol. The default XY routing protocol was modified to implement our trust-aware routing

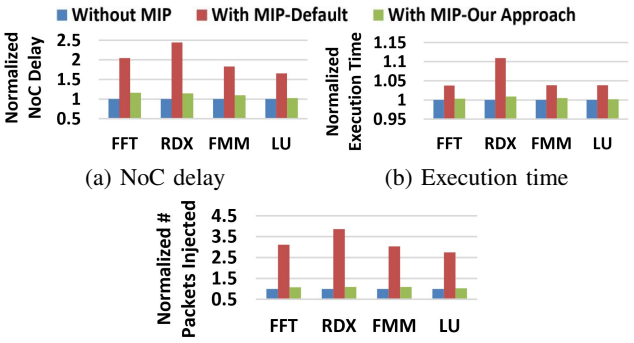
protocol. In our experiments, we used $\delta = 0.5$ (Equation 10) when increasing/reducing direct trust. The value 0.5 was chosen experimentally such that the algorithm chooses alternative paths as quickly as possible while minimizing the impact of false negatives.

We tested the system using 4 real benchmarks (FFT, RADIX, FMM, LU) from the SPLASH-2 benchmark suite and 7 synthetic traffic patterns (*uniform random (URD), tornado (TRD), bit complement (BCT), bit reverse (BRS), bit rotation (BRT), shuffle (SHF), transpose (TPS)*). When running both real benchmarks and synthetic traffic patterns, each IP in the top (first) row of the Mesh NoC instantiated an instance of the task. Real benchmarks used 8 memory controllers that provide the interface to off-chip memory which were connected to the bottom eight IPs. As synthetic traffic patterns don't use memory controllers, the destination of injected packets were selected based on the traffic pattern. For example, uniform random selected the destination from the IPs at the bottom row with equal probability. Source and destination modelling was done this way to mimic the secure and non-secure zones. Four MIPs were modeled and assigned at random to IPs in the other six rows. To simulate the sporadic behavior of the MIPs as discussed in the threat model, each MIP corrupted n consecutive packets after every p (period) packets. According to our architecture model, the IPs in the top row (secure zone) communicate with the IPs in the bottom row (secure zone) through the other 6 rows (non-secure zone) of IPs out of which, 4 are malicious. Our approach will work the same for any other secure, non-secure zone selection and MIP placement. The output of the gem5 simulation statistics was fed to the McPAT power modelling framework to obtain power consumption [26].

B. Performance Improvement

Figure 6 shows results related to the performance improvement when running real benchmarks. The figure compares performance results without the presence of MIPs (Without MIP), with the presence of MIPs when default XY routing is used (With MIP-Default), and when our approach is used with the presence of MIPs (With MIP-Our Approach). We can observe that our approach reduces NoC delay by 53% (43.6% on average) compared to the default XY routing protocol. Execution time and number of packets injected are reduced by 9% (4.7% on average) and 71.8% (66% on average), respectively. When the MIPs corrupt packets, re-transmissions are caused and its trust is reduced. As a result, alternative paths are chosen. The performance improvement depends on how quickly the algorithm chooses an alternative path once an attack is initiated.

In addition to real benchmarks, we experimented with synthetic traffic traces as well. Results related to synthetic traffic patterns are shown in Figure 7. The comparison is the same as that of Figure 6. It shows that NoC delay and number of packets injected on the NoC are reduced by

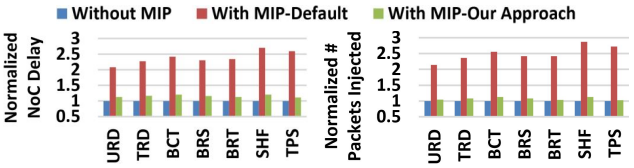


(a) NoC delay (b) Execution time

(c) No. of packets injected

Figure 6: NoC delay, execution time and number of packets injected with and without our trust-aware routing model when running real benchmarks. $p = 20$ and $n = 14$. This figure is an extension of Figure 2.

57.1% (51.2% on average) and 56.7% (50.1% on average), respectively.



(a) NoC delay (b) No. of packets injected

Figure 7: Execution time and number of packets injected with and without our trust-aware routing model when running synthetic traffic patterns. $p = 20$, $n = 14$.

C. Energy Efficiency Improvement

As a result of reduced execution time and reduced number of re-transmissions, the energy consumption of the SoC also reduces. Figure 8 shows the energy consumption comparison. Note that 47.4% (28.3% on average) less energy is consumed by real benchmarks when routing using our approach compared to the default XY routing in the presence of MIPs. Synthetic traffic demonstrate energy savings of up to 75.6% (67.6% on average). Compared to real benchmarks, synthetic traffic patterns show more energy reduction since synthetic traffic focuses only on network traversals unlike real benchmarks which goes through the entire processor pipeline including instruction execution, NoC traversal and memory operations.

D. Overhead Analysis

To implement our routing protocol, additional hardware is required at each router. This includes extra memory to store trust values and hardware to calculate, update and propagate trust. To accommodate a row in the ComTable, 10 bytes of memory is required (6-bit *src*, 6-bit *dest*, 32-bit *addr*,

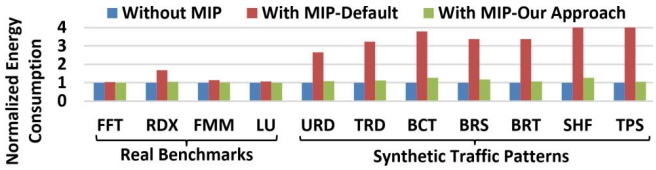


Figure 8: Energy consumption with and without our trust-aware routing model when running real benchmarks and synthetic traffic patterns. $p = 20$, $n = 14$.

1-bit *rtx flag*, 32-bit *timestamp*). The maximum size of the ComTable during our experiments was 24. This leads to 240 bytes of extra memory requirement per router.

We used the default 5-stage router pipeline (buffer write, virtual channel allocation, switch allocation, switch traversal and link traversal) implemented in gem5. Once separate hardware is implemented, computations related to trust can be carried out in a pipelined fashion in parallel to the computations in the router pipeline. To evaluate the area overhead, we modified the RTL design of an open source NoC router [27] and synthesized the design with 180nm GSCLib library from Cadence using Synopsis Design Compiler. This resulted in an area overhead of 6% compared to the default router. This shows that the proposed trust-aware routing protocol is lightweight and can be effectively implemented at routers in an NoC-based SoC.

VI. Conclusions

In this paper, we proposed a trust-aware routing protocol that is capable of routing packets by avoiding malicious IPs in NoC-based SoCs. The routing protocol is implemented based on a trust model that calculates how much a neighboring node can be trusted to route packets through that router. We proposed an effective trust model that can be easily scaled to any number of IPs. The experiments conducted by using both real benchmarks and synthetic traffic patterns demonstrated significant performance and energy efficiency improvements compared to traditional XY routing in the presence of a MAC-then-encrypt security protocol. Overhead analysis has revealed that the area overhead to implement the routing protocol is only 6%. This approach can be integrated with any existing authentication scheme as well as other threat mitigation techniques, to secure the SoC while minimizing the performance and energy efficiency degradation caused by a malicious IP tampering packets.

Acknowledgment

This work was partially supported by the National Science Foundation (NSF) grant SaTC-1936040.

References

[1] P. Mishra, S. Bhunia, and M. Tehranipoor, *Hardware IP security and trust*. Springer, 2017.

[2] F. Farahmandi, Y. Huang, and P. Mishra, *System-on-Chip Security: Validation and Verification*. Springer Nature, 2019.

[3] (2008) Security on arm trustzone. [Online]. Available: https://www.arm.com/products/security-on-arm/trustzone

[4] S. Murali *et al.*, "Analysis of error recovery schemes for networks on chips," *D&T*, vol. 22, no. 5, pp. 434–442, 2005.

[5] S. Charles, A. Ahmed, U. Y. Ogras, and P. Mishra, "Efficient cache reconfiguration using machine learning in noc-based many-core cmps," *TODAES*, vol. 24, no. 6, pp. 1–23, 2019.

[6] T. Dierks and E. Rescorla, "The transport layer security (tls) protocol version 1.2," Tech. Rep., 2008.

[7] T. Boraten *et al.*, "Secure model checkers for network-on-chip (noc) architectures," in *GLVLSI*, 2016, pp. 45–50.

[8] H. K. Kapoor *et al.*, "A security framework for noc using authenticated encryption and session keys," *CSSP*, vol. 32 (6), pp. 2605–2622, 2013.

[9] L. S. Indrusiak *et al.*, "Side-channel attack resilience through route randomisation in secure real-time networks-on-chip," in *ReCoSoC*, 2017, pp. 1–8.

[10] R. JS *et al.*, "Runtime detection of a bandwidth denial attack from a rogue network-on-chip," in *NOCS*, 2015, p. 8.

[11] S. Charles, M. Logan, and P. Mishra, "Lightweight anonymous routing in noc based socs," in *DATE*, 2020.

[12] S. Charles and P. Mishra, "Securing network-on-chip using incremental cryptography," *ISVLSI*, 2020.

[13] S. Charles, Y. Lyu, and P. Mishra, "Real-time detection and localization of distributed dos attacks in noc based socs," *IEEE TCAD*, 2020.

[14] S. Charles, Y. Lyu and P. Mishra, "Real-time detection and localization of dos attacks in noc based socs," in *DATE*, 2019, pp. 1160–1165.

[15] Y. Lyu and P. Mishra, "A survey of side-channel attacks on caches and countermeasures," *Journal of Hardware and Systems Security*, vol. 2, no. 1, pp. 33–50, 2018.

[16] B. Lebiednik *et al.*, "Architecting a secure wireless network-on-chip," in *NOCS*, 2018, pp. 1–8.

[17] H. M. Wassel *et al.*, "Surfnoc: a low latency and provably non-interfering approach to secure networks-on-chip," in *ACM SIGARCH Computer Architecture News*, 2013.

[18] Y. L. Sun *et al.*, "A trust evaluation framework in distributed networks: Vulnerability analysis and defense against attacks." in *INFOCOM*, vol. 6, 2006, pp. 1–13.

[19] P. Zimmerman, "Pretty good privacy v2. 62," *Online. Ftp://net-dist. mit. edu*, 1995.

[20] J. Callas *et al.*, "Openpgp message format," Tech. Rep., 2007.

[21] D. Engels *et al.*, "Ultra-lightweight cryptography for low-cost rfid tags: Hummingbird algorithm and protocol," *CARC Tech. Rep.*, 2009.

[22] M. Li *et al.*, "Dyxy: a proximity congestion-aware deadlock-free dynamic routing method for network on chip," in *DAC*, 2006.

[23] N. Binkert *et al.*, "The gem5 simulator," *SIGARCH Computer Architecture News*, 2011.

[24] S. Charles, C. A. Patil, U. Y. Ogras, and P. Mishra, "Exploration of memory and cluster modes in directory-based many-core cmps," in *NOCS*, 2018, pp. 1–8.

[25] N. Agarwal *et al.*, "GARNET: A detailed on-chip network model inside a full-system simulator," *ISPASS*, 2009.

[26] S. Li *et al.*, "Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures," in *MICRO*, 2009, pp. 469–480.

[27] A. Monemi *et al.*, "ProNoC: A low latency network-on-chip based many-core system-on-chip prototyping platform," *Microprocessors and Microsystems*, 2017.