J Netw Syst Manage (2013) 21:335–352
DOI 10.1007/s10922-012-9238-0

TCAMChecker: A Software Approach to the Error Detection and Correction of TCAM-Based Networking Systems

M. Zubair Shafiq · Chad Meiners · Zheng Qin · Ke Shen · Alex X. Liu

Received: 23 September 2010 / Revised: 12 April 2012 / Accepted: 10 May 2012 / Published online: 26 May 2012

Abstract Ternary content addressable memories (TCAMs) are widely used in network devices carrying out the core operation of single-operation lookups. TCAMs are the core component of many networking devices such as routers, switches, firewalls and intrusion detection/prevention systems. Unfortunately, they are susceptible to errors caused by environmental factors such as radiation. TCAM errors may have significant impact on search results. In fact, only one error in a TCAM can cause 100 % of search keys to have wrong lookup results. Therefore, TCAM error detection and correction schemes are needed to enhance the reliability of TCAM-based systems. All prior solutions require hardware changes to TCAM circuitry and therefore are difficult to deploy. In this paper, we propose

The work of Alex X. Liu is partially supported by the National Science Foundation under Grant No. CNS-0916044 and Grant No. CNS-0845513. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. The work of Zheng Qin is partially supported by the National Science Foundation of China under Grant No. 61070194 and the Science and Technology Key Projects of Hunan Province (2011FJ2003). Zheng Qin is the corresponding author of this paper.

M. Z. Shafiq · C. Meiners · K. Shen · A. X. Liu
Department of Computer Science and Engineering, Michigan State University, East Lansing, MI 48824, USA
e-mail: shafiqmu@cse.msu.edu

C. Meiners
e-mail: meinersc@cse.msu.edu

K. Shen
e-mail: shenke@cse.msu.edu

A. X. Liu
e-mail: alexliu@cse.msu.edu

Z. Qin (✉)
College of Software, Hunan University, Changsha 410082, Hunan, China
e-mail: qz88@263.net

TCAMChecker, the first software-based solution for TCAM error detection and correction. Given a search key, TCAMChecker probabilistically decides to verify the lookup result. If TCAMChecker decides to verify the lookup result then it performs two parallel lookups for the given search key. If the lookup results do not match then at least one error is detected and is corrected by using a backup error-free memory. Note that the probability of lookup verification can be tuned for tradeoff between performance and reliability. A higher probability of lookup verification provides a more reliable TCAM system at the cost of performance. Our proposed TCAMChecker can be easily deployed on existing TCAM-based networking devices to improve the system reliability.

Keywords Intrusion detection systems · *N*-version programming · Ternary content addressable memory

1 Introduction

Ternary Content Addressable Memory (TCAM) is a special type of memory. For random access memory (RAM), a bit has only two possible values (i.e., 0 or 1), and the input is the index and the output is the content of the memory at that index. Whereas, a bit in TCAM has three possible values: 0, 1, or *, where * means “don’t care”. Each entry in TCAMs is an array of 0s, 1s, or *s. TCAMs work in a reverse manner as compared to RAMs: the input to a TCAM chip is a packet header (i.e., a search key) of 0s and 1s, and the output is the index of the *first* entry that the key matches where decisions are stored [1]. A search key matches an entry if and only if their bits of 0s and 1s match. For example, 1,001 matches a TCAM entry 10**. Furthermore, a TCAMs searches the entire memory in one operation, so it is considerably faster than RAMs. In essence, a TCAM is a parallel search machine, not just memory.

TCAMs are the core component of many networking devices such as routers, switches, firewalls and intrusion detection/prevention systems. One major task performed by TCAMs is packet classification, the processing of finding the first rule that a given packet matches in a rule list. In TCAM-based packet classification, rules are stored in the TCAM in ternary format. Thus, when a packet comes, only one TCAM lookup is needed. The result of the TCAM lookup is the index in the memory where packet decisions, such as accept or drop, are stored. This superior constant lookup performance is the key reason that many networking devices use TCAMs to implement their lookup operations.

A TCAM bit may flip to any of the other two values of 0, 1, or * due to environmental impacts. Memory and electronics in general are susceptible to non-persistent faults, called *soft errors*, due to radiation events that generate enough electricity to cause a bit to flip in memory or for a transistor to trigger or fail to trigger [2, 3]. TCAMs are even more susceptible to soft errors than RAMs because of much higher circuit density [4]. Furthermore, the soft errors in TCAMs are more damaging than the soft errors in RAMs. While a soft error in RAM only causes one lookup to fail, a soft error in a TCAM chip can cause many different lookups to fail

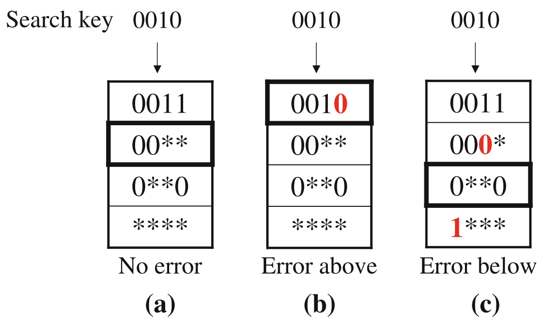


Fig. 1 Effects of TCAM soft errors

due to the first matching semantics: the lookup result of a search key is the index of the first entry that the key matches [5]. For example, if a TCAM entry "1**" changes to "100", then the lookup results of the three keys "101", "110", and "111" may become erroneous. In fact, a single erroneous entry in a TCAM can cause 100 % of search keys to have erroneous lookup results in the worst case. TCAMs are increasingly prone to more soft errors as the area efficiency of memory devices decreases [5]. Furthermore, Mastipuram et al. [5] reported that soft errors induced the highest failure rate of all other reliability mechanisms combined in TCAMs. Consequently, it is important to ensure reliability in TCAM operations because soft errors can potentially jeopardize their application in mission-critical network management applications.

Figure 1 shows different effects of soft errors in TCAMs. Note that the index returned by an erroneous lookup may be smaller or greater than the index returned by a correct lookup. Figure 1a shows an example TCAM table without errors, on which the lookup result for search key 0010 is the second entry. Figure 1b shows an implementation of the table in Fig. 1a where the last bit in the first entry is flipped from 1 to 0. On this table, the lookup result for search key 0010 is the first entry. Figure 1c shows an implementation of the table in Fig. 1a where the third bit in the second entry is flipped from * to 0. On this table, the the lookup result for search key 0010 is the third entry. Moreover, some soft errors do not affect the lookup result of any search key in a TCAM. For example, the first bit in the fourth entry in Fig. 1c erroneously flipped from * to 1. However, this error does not affect any lookup. In this paper, we do not consider such benign errors as they cause no harm. In the rest of this paper, the term TCAM errors refers to harmful TCAM errors.

1.1 Limitations of Prior Work

Due to the ternary nature and first match semantics of TCAMs, traditional methods of mitigating the effects of soft error, such as error correcting codes, are difficult to apply to TCAMs. Prior TCAM error detection and correction schemes [4, 6–9] all require hardware modifications to TCAM circuitry. Such changes often lower performance and increase TCAM die area. Increasing die area directly increases manufacturing cost in semiconductor industry, including TCAM fabrication. Furthermore, the cost of manufacturing TCAM chips of new design is prohibitive [10]. Recently, Kirishnan et al. [7] have proved that any error correcting codes for

TCAM, even assuming at most one erroneous bit per entry, will require at least an extra two times die area.

1.2 Proposed Approach

TCAMChecker idea is inspired by *N*-version programming [11, 12] and back-to-back testing [13]. *N*-version programming works by giving the same requirement specification to *N* teams to independently design and implement *N* programs using different algorithms, languages, or tools. The resulting *N* programs are executed in parallel with a decision selection mechanism, which is deployed to examine the *N* results for each input from the *N* programs and selects a correct or "best" result. *N*-version programming method has been used for building fault-tolerant software in a variety of safety-critical systems built since the 1970s, such as railway interlocking and train control [14], flight control [15], and nuclear reactor protection [16]. Back-to-back testing is associated with *N*-version programming. It is used to test the resulting *N* versions before deploying them in parallel. Back-to-back testing works as follows. First, a suite of test cases needs to be created. Second, for each test case, *N* programs are executed in parallel and the *N* results are cross-compared. Finally, all discovered discrepancies need to be investigated and errors discovered in the investigation need to be corrected.

In this paper, we propose TCAMChecker, the first pure software-based solution for TCAM error detection and correction. TCAMChecker assumes that any bit in a TCAM entry can go erroneous. Given a search key, TCAMChecker probabilistically decides whether it should verify the lookup result. If yes, TCAMChecker performs two independent lookups for the search key; if the lookup results disagree, one error is guaranteed to be detected and TCAMChecker corrects it accordingly using backup table stored in an error-free memory such as ECC-RAM (ECC stands for Error Correction Codes) or NVRAM (Non-volative RAM). Figure 2 shows the architecture of TCAMChecker. To enable two independent lookups, TCAMChecker stores two copies of the same table in the TCAM. By setting the probability of

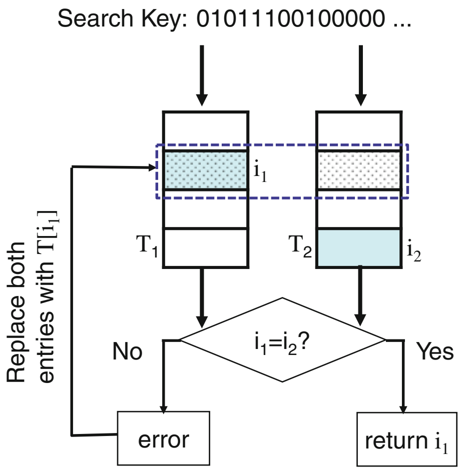

Search Key: 01011100100000 ...
i_1
T_1
T_2
i_2
$i_1=i_2$?
No
Yes
error
return i_1
Replace both entries with $T[i_1]$

Fig. 2 TCAMChecker architecture

TCAM lookup verification, TCAMChecker provides a fine grain and adjustable mechanism for trading system performance for accuracy. In terms of accuracy, TCAMChecker has no false positives, but has false negatives with an extremely small probability. The false negative probability can be tuned by changing the probability of TCAM lookup verification. In terms of performance, majority of packets only incur a single lookup because most lookups are not going to result in errors. This claim is validated during the experimental evaluation of TCAMChecker. It is important to note here that the software component of TCAMChecker is not a bottleneck in the packet processing pipeline. The only component of TCAMChecker that is implemented in software is the one that matches lookup results. The matching of two integers on a typical processor with clock speed in the order of gigahertz (GHz) consumes much less time than a lookup on a typical TCAM with clock speed in the order of megahertz (MHz).

1.3 Advantages Over Prior Work

TCAMChecker is the first software only method that can detect and fix errors during the search process. TCAMChecker has many advantages over prior TCAM error detection and correction schemes. First, it requires no hardware changes to TCAM circuitry. Therefore, it can be easily deployed on existing TCAM-based systems, unlike solutions in the prior work. Adopting TCAMChecker could be as simple as updating firmware or software. It incurs low cost as TCAM chips do not need to be specially fabricated. Second, TCAMChecker provides excellent flexibility to tradeoff between performance and reliability by adjusting the verification rate. Third, the performance and cost of TCAMChecker do not depend on the maximum number of erroneous bits in an entry. In contrast, many prior TCAM detection and correction schemes, such as [4, 7] have their performance or cost depend on the maximum number of bits in an entry that can be erroneous.

We make the following key contributions in this paper.

1. We propose the first software-based method, TCAMChecker, that detects and corrects soft errors in TCAM entries without requiring changes to TCAM circuitry.
2. We provide a probabilistic model for analyzing the expected number of times a TCAM could return erroneous results after a soft error has occurred. This model shows that TCAMChecker intelligently detects and fixes soft errors more reliably than prior techniques. We also evaluate the effectiveness of TCAMChecker using filter sets and packet traces generated using an open-source tool. For baseline comparison, we also compare TCAMChecker with a prior scheme called TCAM scrubbing.

The rest of the paper proceeds as follows. We first discuss related work in Sect. 2 We then present an overview and algorithmic details of TCAMChecker in Sect. 3 In Sect. 4, we describe different ways to implement TCAMChecker in existing TCAM architectures and discuss their advantages and disadvantages. In Sect. 5, we evaluate the performance of our TCAMChecker and compare it with TCAM scrubbing technique for baseline comparison. Finally, we conclude the paper with Sect. 6

2 Related Work

To the best of our knowledge, all prior solutions for TCAM error detection and correction require modifying TCAM circuitry. Therefore, they share the same fundamental limitation of being expensive and difficult to deploy on existing TCAM architectures. Modifying TCAM circuitry "involve millions of dollars of investment and more than 2 years of development time" [10]. The degree by which the circuitry is modified varies in prior TCAM error detection and correction schemes. We accordingly classify prior work into two categories: hardware-based approaches, and hardware-software hybrid approaches.

2.1 Hardware-Based Approaches

Hardware-based approaches focus on maintaining the integrity of an entry's content in the presence of soft errors. Noda et al. [9] proposed to build TCAMs with dynamic RAM (DRAM) cells instead of static RAM (SRAM) cells, which gives TCAM cells higher resistance to soft errors. Later, Noda et al. [8] proposed to use ECC-DRAM cells to store a backup of SRAM cells. The contents of the backup cells would be constantly used to set the values of the SRAM cells to provide the reliability of DRAM at the speed of SRAM. Both techniques have serious drawbacks. For the first technique, the DRAM-based TCAMs are much slower than the traditional SRAM-based TCAMs. For the second technique, the TCAMs with ECC-DRAM backup require a much larger die area than traditional SRAM cell TCAMs, which will significantly increase the chip price. The final product cost of a chip in the semiconductor industry is strongly dependent on the die area of the chip [17].

Azizi and Najm proposed two methods for making the SRAM-based TCAMs more resistant to soft errors [6]. These two methods use extra transistors in each TCAM cell to make the ternary state stable so that they are more resistant to soft errors. Each method offers a distinct trade-off between the extra die area used by the extra circuitry and the additional reliability of the SRAM cells.

Most recently, Kirishnan et al. [7] developed a theory for ternary Error Correction Codes (ECCs). Prior to this theory, ECCs can only be applied to Binary Content Addressable Memories (BCAMs) [18]. They show that ECCs can be adapted to work in TCAMs, but the cost associated with these ECCs is very high. For example, tolerating only a single bit error would require a die area increase of 200 %.

2.2 Hardware-Software Hybrid Approaches

One approach is to store additional information with each entry so that each entry can be individually verified for correctness via a dedicated circuit [19]. This approach is called scrubbing because each entry is checked individually and refreshed if it fails a parity check. Scrubbing requires the support of software to indicate when an entry should be checked by the parity hardware. It is a compromise between the hardware-based error detection, which masks the occurrence of soft errors to the software, and software-based error correction, which corrects TCAM entry without additional TCAM circuitry. This work differs from *TCAM scrubbing*, which is a technique that periodically rewrites TCAM entries during idle cycles, because the built in parity checks avoid unnecessary rewrites.

Bremler-Bar et al. [4] pioneer the first hardware-software hybrid approach that outperforms scrubbing. Their technique is called PEDS. They use the unused bits in a TCAM entry to encode its parity matrix information. During idle cycles, PEDS periodically checks the integrity of bit columns using the parity matrix information. It differs from prior scrubbing techniques in that the number of lookups required to verify the entire TCAM is based upon the entry width rather than the number of entries; therefore, it uses less idles cycles than scrubbing to detect and correct errors. However, this technique still requires modifications to the TCAM circuitry.

In contrast to the above-mentioned prior work, our TCAMChecker is a pure software approach to TCAM error detection and correction. TCAMChecker has the advantage of being cheap and easy to deploy. Furthermore, TCAMChecker can be used in conjunction with prior TCAM error detection and correction schemes to further enhance reliability.

3 TCAMChecker

In this section, we present TCAMChecker, a pure software-based method for detecting and correcting TCAM errors while performing lookups.

3.1 Overview of TCAMChecker

In TCAMChecker, a ternary lookup table is independently stored in two separate tables. Given a search key over a ternary table, TCAMChecker first performs two independent lookups on the two copies of the table and then compares the search results. If the lookup results mismatch, then at least one error is detected and a correction is accordingly performed. Using two independent lookups, we achieve the purpose of error detection and correction without any hardware modification to TCAM circuitry. Note that in practice, performing a double lookup for every search key is excessively expensive because the majority of lookups will be correct since the probability of soft-errors is low. Instead, we probabilistically choose search keys to go through double lookups. The probabilistic analysis of TCAMChecker is in Sect. 5 shows that by setting the rate at which we check twice, we can control the expected number of misclassifications that result from any error. TCAMChecker has no false positives. For any search key, when the two lookup results do not agree, we guarantee to detect and correct at least one error. Unfortunately, TCAMChecker does have false negatives, although the probability of a false negative happening is extremely low. We discuss the probability of false negatives later in Sect. 5.

3.2 Algorithmic Details of TCAMChecker

We now discuss the theoretical foundations of our proposed TCAMChecker. TCAMChecker guarantees to detect and correct at least one error for every lookup mismatch based on Theorem 1. In the rest of this paper, for any ternary table *T*, an index r, and a search key x, we use $T[r]$ to denote its rth entry and $T(x)$ to denote the index of the first entry that matches x. Given a search key for a TCAM, we assume that the lookup result is the index of the first entry that the key matches.

Theorem 1 *Given a search key x and two TCAM tables T_1 and T_2 that independently implement the same error-free ternary table T_0, if $T_1(x) \neq T_2(x)$, and denoting* $\min\{T_1(x), T_2(x)\}$ *as r, then $T_0[r] \neq T_1[r]$ or $T_0[r] \neq T_2[r]$.*

Proof Without loss of generality, we assume $i = T_1(x) < T_2(x)$. For the sake of contradiction, we suppose $T_0[r] = T_1[r] = T_2[r]$. Thus, x matches entry $T_2[r]$, which implies that $T_2(x) \leq i$ based on the first-match semantics. This contradicts with the assumption that $r < T_2(x)$.

Based on Theorem 1, TCAMChecker works as follows. Given a ternary table T_0 that we assume to be stored in an error-free memory such as ECC-RAM or NVRAM, TCAMChecker stores copies of T_0 in two TCAM tables T_1 and T_2. Given a search key x, TCAMChecker independently searches with x in both tables. If $T_1(x) \neq T_2(x)$, assuming $R_{min} = \min\{T_1(x), T_2(x)\}$, TCAMChecker rewrites both entries $T_1[R_{min}]$ and $T_2[R_{min}]$ using $T_0[R_{min}]$, which is guaranteed to correct at least one TCAM error, and then searches x on both T_1 and T_2 again. The above process continues until the two lookup results $T_1(x)$ and $T_2(x)$ match the ground truth $T_0(x)$.

For example in Fig. 1, if (a) is T_0, (b) is T_1 and (c) is T_2, the search key 0010 produces the results $T_1(0010) = 1$ and $T_2(0010) = 3$. TCAMChecker will refresh $T_1[1]$ and $T_2[1]$ using $T_0[1]$, and repeat the search. This search results in $T_1(0010) = 1$ and $T_2(0010) = 3$ and causes $T_1[2]$ and $T_2[2]$ to be refreshed from $T_0[2]$. Final search will return a consistent result of $T_1(0010) = 2$ and $T_2(0010) = 2$.

So far, we have assumed that for any search key x, both T_1 and T_2 have matching entries for x. Next, we address the corner case that T_1 or T_2 does not have matching entries for a search key x due to TCAM errors. We assume that the last entry in T_0 matches any search key (i.e., every bit in the last entry in T_0 is *); otherwise, we cannot ensure that the last entry matches any search key. Note that we can always replace the last entry in a TCAM table by an entry of all *s without changing the semantics of the TCAM table. Thus, given any implementation T of T_0 and a search key x, if x does not match any entry in T, then the last entry of T must contain errors. To cater for this case, TCAMChecker also checks if $T_1(x) > |T|$ or $T_2(x) > |T|$ along with $T_1(x) \neq T_2(x)$. Again assuming $R_{min} = \min\{T_1(x), T_2(x)\}$, it first identifies the smaller of R_{min} and $|T|$. Let $R_{correct}$ denote the smaller of R_{min} and $|T|$. TCAMChecker then rewrites both entries $T_1[R_{correct}]$ and $T_2[R_{correct}]$ using $T_0[R_{correct}]$, and searches x on both T_1 and T_2 again. Now, this process continues till $T_1(x) = T_2(x)$, $T_1(x) \leq |T|$ and $T_2(x) \leq |T|$.

Algorithm 1 shows the pseudocode of TCAMChecker algorithm. In Algorithm 1, we use temporary variables R_1 and R_2 to hold lookup results for $T_1(x)$ and $T_2(x)$ respectively for efficiency reasons. Furthermore, to avoid TCAMChecker going in an infinite loop in case we have a hard error (e.g. a bit keeps flipping to 0 when the entry should have 1), we limit the maximum number of times TCAMChecker tries to correct error to *MAX_ITERATIONS*. If the number of iterations exceeds *MAX_ITERATIONS*, then an error flag HW_ERROR_FLAG is set true and TCAMChecker quits.

4 Storage Schemes for TCAMChecker

In this section, we describe different ways to store the two lookup tables and discuss their advantages and disadvantages. There are two schemes to store tables in TCAMChecker: 2-chip scheme and 1-chip scheme. For either storage scheme, we store the trustworthy copy in ECC-RAM, which is assumed to be error-free. For safety critical systems where ECC-RAM cannot be practically assumed to be free of errors, redundant copies or other error masking techniques can be used to achieve storage with sufficient reliability.

4.1 2-Chip Scheme

In this scheme, T_1 and T_2 are stored separately in two TCAM chips and T_0 is stored in RAM (either SRAM or DRAM). Given a search key, we perform one *parallel* lookup on each of the two TCAM chips; if the search results mismatch, we use the table in RAM to rewrite one entry in each table of T_1 and T_2, correcting at least one error. Note that most TCAM-based routers are equipped with two TCAM chips [20], one as a backup chip that allows TCAM tables to be updated without interrupting searches. For such systems, TCAMChecker offers an additional use for the secondary chip to provide a higher degree of reliability.

Algorithm 1: TCAMChecker

```
Input: A search key $x$ and two TCAM tables $T_1$ and $T_2$ that independently implement
       the error-free table $T_0$ each containing $|T|$ entries
1  if $Rand() \leq P_C$ then
2      return $T_1(x)$
3  $R_1 = T_1(x)$    //Temporary variable that stores $T_1$ index lookup value
4  $R_2 = T_2(x)$    //Temporary variable that stores $T_2$ index lookup value
5  Iterations = 0
6  while $(R_1 \neq R_2) \vee (R_1 > |T|) \vee (R_2 > |T|)$ do
7      $R_{min} = min(R_1, R_2)$
8      if $R_{min} < |T|$ then
9          $R_{correct} = R_{min}$
10     else
11         $R_{correct} = |T|$
12     $T_1[R_{correct}] = T_2[R_{correct}] = T_0[R_{correct}]$    //Refresh $T_1$ and $T_2$ from $T_0$
13     Iterations++
14     //Refresh lookup results in the temporary variables for next iteration
15     $R_1 = T_1(x)$
16     $R_2 = T_2(x)$
17     if $Iterations > MAX\_ITERATIONS$ then
18         HW_ERROR_FLAG = true
19         break    //To avoid infinite loop in case of hardware error
20 return $R_1$
```

4.2 1-Chip Scheme

In this scheme, T_1 and T_2 are stored in one TCAM chip and T_0 is stored in RAM (either SRAM or DRAM). There are two possible methods for efficiently storing two identical tables: bank discrimination and bit discrimination. We discuss both of them separately below.

4.2.1 Bank Discrimination

With bank discrimination, we use the row banks within TCAMs to distinguish the two tables residing in the same chip. Row banks are series of TCAM entries that can be selectively enabled or disabled during a TCAM search. The primary function of row banks is power savings because disabled banks are not searched. TCAMs are most commonly divided into 64 row banks. Bank discrimination distinguishes between the two tables storing each table in a distinct set of row banks and disabling the row banks that contain the table that is not being searched. This technique has the advantage that storing the extra table in the TCAM does not increase the power required for searching either table.

4.2.2 Bit Discrimination

With bit discrimination, we use one extra bit in the TCAM chip to distinguish the two tables residing in the same chip. Given two ternary tables T_1 and T_2, we first prepend each entry in T_1 with 0 and each entry in T_2 with 1, and then concatenate the two resulting tables into one table, denoting T_{12}. Thus, for any search key x, assuming T_1 is stored on top of T_2, we have $T_1(x) = T_{12}(0x)$ and $T_2(x) = T_{12}(1x) - |T_1|$. Given a search key x, we perform two *consecutive* lookups on T_{12} using key $0x$ and $1x$, respectively; if the search results mismatch, we use the table in RAM to rewrite the two corresponding entries in T_{12}.

In bit discrimination, we can detect the case when discriminator bits get corrupted. First, for any search key $0x$, if T_{12} has no entry that matches this key, then the entry $T_{12}[|T_1| - 1]$ must contain errors and therefore should be rewritten. Similarly, for any search key $1x$, if T_{12} has no entry that matches this key, then the entry $T_{12}[|T_{12}| - 1]$ must contain errors and therefore should be rewritten. Second, if $T_{12}(0x) > |T_1| - 1$, which means that $0x$ does not match entry $T_{12}[|T_1| - 1]$ although it should, then $T_{12}[|T_1| - 1]$ must contain errors and therefore should be rewritten. Third, if $T_{12}(1x) \leq |T_1| - 1$, which means that $1x$ matches an entry that it should not match, then entry $T_{12}[T_{12}(1x)]$ must contain errors and therefore should be rewritten.

4.2.3 Bank Discrimination Versus Bit Discrimination

Bit discrimination has the advantage that two small tables can be stored in the same row bank with bit discrimination. We recommend using bank discrimination whenever both tables cannot be stored in a single bank because using bit discrimination across multiple bank increases the power consumption for searching each table; however, when both tables fit in a single bank, bit discrimination does not increase power consumption because the number of banks searched is not increased.

4.3 2-Chip Scheme Versus 1-Chip Scheme

The 2-chip scheme has better performance than 1-chip scheme because it uses parallel lookups rather than two sequential lookups. The 2-chip scheme has higher cost than 1-chip scheme because it requires two separate TCAM chips rather than only one TCAM chip. Therefore, if the TCAM system has two TCAM chips and each is enough to store the ternary table then we should use the 2-chip storage scheme; whereas, if the system has only one TCAM chip that is large enough to store two copies of the table then we should use the 1-chip scheme.

5 Analysis and Comparison of TCAMChecker and TCAM Scrubbing

In this section, we present the analysis and comparison of TCAMChecker and TCAM scrubbing. Towards this end, we first mathematically model the probability of false negatives for TCAMChecker and show that it is negligible. We then estimate the expected number of misclassifications for TCAMChecker caused by TCAM bit errors during time between their occurrence and their correction. For baseline comparison, we also estimate the expected number of misclassifications for TCAM scrubbing. We then compare the actual number of misclassifications of TCAMChecker and TCAM scrubbing by evaluating them on filter sets and packet traces generated using an open-source tool called ClassBench [21]. We also provide insights about selecting appropriate configuration parameters for TCAMChecker and TCAM scrubbing. Finally, we provide some discussion about the power analysis of TCAMChecker and TCAM scrubbing.

5.1 Probability of False Negatives for TCAMChecker

Recall that TCAMChecker guarantees to detect an error when the results of parallel lookups do not match, i.e. $T_1(x) \neq T_2(x)$. In addition to the case when $T_1(x) \neq T_2(x)$, in a rare scenario, some errors may occur in T_1 and T_2 such that $T_1(x) = T_2(x)$ but $T_1(x)$ or $T_2(x)$ do not match $T_0(x)$. Such errors are termed as false negatives of TCAMChecker and can occur when the same ternary bit changes in both TCAMs. Note that the probability of false negatives is bounded by the rate at which the TCAMs are periodically refreshed.

We now derive an expression for the probability of false negatives for TCAMChecker. Since TCAM errors are discrete events, we first model the probability of error according to the binomial distribution. Let p be the probability that a given ternary bit changes during a single TCAM cycle and let n be the size of a TCAM (in bits). Let K be the random variable denoting the number of bit errors during a single TCAM cycle. We assume that the event of error occurrence in different bits are independent of each other. Note that this assumption does not rule out the possibility of block errors, whose probability can be computed using the geometric distribution under the above-mentioned assumption. Using binomial distribution, the probability that k bits out of a total of n become erroneous in a single TCAM cycle is given by:

$$P(K = k) = \binom{n}{k} p^k (1 - p)^k. \tag{1}$$

Consequently, the expected number of bit errors during a single TCAM cycle is given as: $E[K] = np$.

With TCAMChecker the only way to have a false negative is to have the same bit error happen in both TCAMs in a given cycle. The probability to have the same bit error happen in both TCAMs in a given cycle, denoted by P_{FN}, is given by $1/n^2$. For k bit errors in one cycle, this probability is given as: k/n^2. Here we also assume that the number of bit errors in one cycle is not large in TCAMs, i.e., $n \gg E[K]$, which is a reasonable assumption given the fact that modern TCAMs are fairly reliable [22, 23]. Putting the expression for the expected number of single bit errors during a single TCAM cycle in place of k, the obtain the following final expression: $P_{FN} = p/n$. Given that $p \ll 1$ and n is a large number, we can deduce that the probability of a false negative is negligible, i.e. $P_{FN} \to 0$.

5.2 Expected Number of Misclassifications

Each TCAM error event e has a set of search keys (denoted by $K_e \subset K$) that return erroneous result from the TCAM. We call such erroneous TCAM lookups a *misclassification* for the given error e. We now estimate the expected number of misclassifications for an error during the interval from it occurs until the time in which it is corrected. This expected number quantifies the damage an error causes on average. Note that we are not computing the expected number of cycles until an error is fixed because as long as the error does not cause misclassifications it is effectively benign. Therefore, directly calculating the number of misclassifications gives us a better indication of how each technique increases reliability in the system.

5.2.1 TCAMChecker

For any given lookup, it has a P_C probability of being checked by TCAMChecker. If an error can be corrected by TCAMChecker, the expected number of misclassifications for e (from its occurrence) till it is corrected by TCAMChecker is modeled using geometric distribution as follows:

$$\lim_{i \to \infty} \sum_{j=1}^{i} j \times (1 - P_C)^{j-1} = \frac{1}{P_C}, \tag{2}$$

where i and j represent the count of events when misclassifications occur. Equation 2 shows that TCAMChecker does not prefer any specific TCAM lookups for verification. However, it is expected that some keys are more popular than others in TCAM lookups. Consequently, TCAMChecker checks a popular key with a higher probability than a less popular key. Clearly, errors that are associated with more popular keys have a higher likelihood of causing misclassifications. Therefore, we can deduce that TCAMChecker automatically allocates it effort towards fixing errors that are associated with more popular keys, which in turn have a higher likelihood of causing misclassifications.

5.2.2 *TCAM Scrubbing*

We now estimate the expected number of misclassifications for an error during the interval from it occurs through it is corrected by TCAM scrubbing. Recall that TCAM scrubbing refreshes the TCAM with periodic entry rewrites. Let P_S be the probability that we run TCAM scrubbing in one idle cycle. Note that a single TCAM entry in the table is replaced every P_S cycles by TCAM scrubbing on average. Usually a counter is used to track the last entry replaced so that the entire table is replaced over time. Also, let K_e denote the set of keys affected by an error e. Let pop_k denote the relative popularity of a key k such that $\sum_{\forall k \in K} pop_k = 1$. For a TCAM with utilization ratio U, on average there are $\frac{U}{1-U}$ search cycles between two idle cycles. As the total number of entries in a TCAM is n, the expected number of scrubs needed to correct an error entry is $\frac{1}{n} \times \sum_{i=1}^{n} i = \frac{n+1}{2}$. Combining these terms together, the expected number of misclassifications before the error is corrected is:

$$\left(\frac{1}{P_S} - 1\right) \times \left(\sum_{\forall k \in K_e} pop_k\right) \times \frac{U}{1-U} \times \frac{n+1}{2} \tag{3}$$

It is evident from Equation 3 that expected number of misclassifications is proportional to the popularity of keys affected the error for TCAM scrubbing. This is because TCAM scrubbing uniformly allocates its effort to all TCAM entries, irrespective of their popularity. This is in sharp contrast to TCAMChecker, which naturally verifies popular keys more frequently than less popular ones. We observe later in our experimental evaluation that TCAMChecker significantly outperforms TCAM scrubbing because the former intelligently detects and corrects only the erroneous filters and the latter blindly rewrites filters from the backup source.

5.3 Experimental Evaluation

In this section, we evaluate and compare the accuracy of TCAMChecker and TCAM scrubbing. Due to privacy related concerns, real-world filter sets and packet traces are not publicly available. Therefore, we revert to ClassBench, which is an open-source packet classification performance evaluation tool [21]. ClassBench generates realistic filter sets and packet traces for benchmarking. For each generated filter set, we generate a packet trace that is approximately 1,000 times larger than the size of filter set. An interested reader can find more details in [21]. Using ClassBench, we generate filter sets of sizes varying in the range of 100–500 filters. The fields of filter sets include source IP address, destination IP address, source port and destination port. Note that source and destination ports are in the range format.

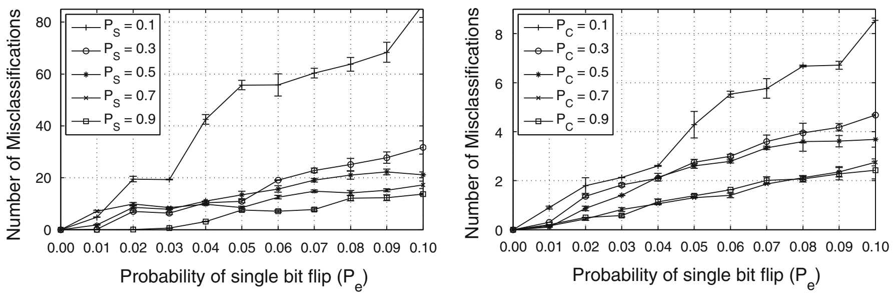
All fields of the filter sets are converted to the prefix format, which increased the size of resulting filter set.

To perform a realistic experimental evaluation of TCAMChecker and TCAM scrubbing for TCAM error detection and correction, we simulate two different error models over a range of error probabilities. A single bit error is simulated by flipping a random bit in a random filter of the filter set; whereas, a block error is simulated by flipping multiple co-located bits in a particular filter of the filter set. The goal of our experimental evaluation is to evaluate and compare the accuracies of TCAMChecker and TCAM scrubbing for different error models, size of filter sets and different values of P_C and P_S. Recall that P_C and P_S are the probabilities of applying TCAMChecker or TCAM scrubbing for TCAM lookup verification of a given packet header, respectively. In our experimental evaluation, P_e denotes the probability of a bit error in a TCAM for each packet header lookup. Therefore, for a packet trace containing 100 thousand entries, $P_e = 0.01$ is equivalent to generating 1000 bit errors. We have reported the average of 100 independent experimental runs along with error bars showing the 95 % confidence intervals for each data point reported in the following figures.

We first investigate the effect of varying the error probability of single bit errors on the number of misclassification for TCAMChecker and TCAM scrubbing while varying values of P_C and P_S. For this investigation, we use a filter set containing 136 filters and a packet trace containing 136 thousand packet headers. Figure 3 shows the plot of the number of misclassifications for TCAM scrubbing and TCAMChecker. As expected, we observe that the number of misclassifications increase for both TCAM scrubbing and TCAMChecker as P_e increases. Furthermore, the number of misclassifications decrease as we increase the values of P_S and P_C parameters.

Before comparing TCAMChecker and TCAM scrubbing, we reiterate that TCAMChecker uses selected lookup verification, i.e. with probability P_C two lookups are carried out for a given packet header and TCAMChecker is enabled only if the lookup results disagree. On the other hand, TCAM scrubbing incrementally refreshes an entry in TCAM from the backup table stored in ECC-RAM or NVRAM with probability P_S for a given packet header. The results of our experiments clearly show that TCAMChecker has lesser average number of misclassifications than TCAM scrubbing for all values of P_e and P_C or P_S. For example, TCAMChecker results in only 4 misclassifications for a packet trace containing 136 thousand packets when only one in ten packet headers are double-checked for lookup verification. TCAM scrubbing misclassifies more than 80 packets out of 136 thousand packets when after every ten packets a rule is refreshed from the backup table.

We now investigate the effect of varying the error probability of block errors P_e on the number of misclassifications for TCAMChecker and TCAM scrubbing while varying values of P_C and P_S. Figure 4 shows the plot of the number of misclassifications for TCAM scrubbing and TCAMChecker in this scenario. We observe trends similar to those observed in Fig. 3; however, the increase in the number of misclassifications for increasing values of P_e decreases significantly.



(a) TCAM scrubbing **(b)** TCAMChecker

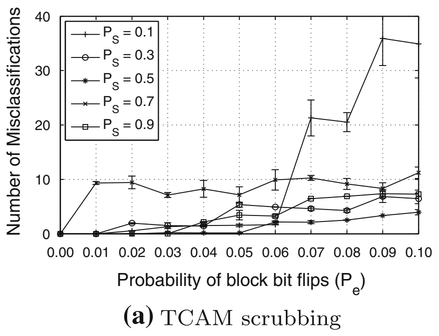
Fig. 3 Comparison of TCAM scrubbing and TCAMChecker for single bit errors while varying values of P_S, P_C and P_e. The results are reported for a filter set with 136 rules and the accompanying network trace containing approximately 136 thousand packet headers

We now study the effect of varying the values of P_C and P_S parameters on the number of misclassifications for TCAMChecker and TCAM scrubbing while varying the sizes of filter sets. Figure 3 shows the plot of the number of misclassifications for TCAM scrubbing and TCAMChecker for single bit errors and block errors. Note that a fixed error probability is used for this evaluation. For both single bit errors and block errors, we observe that the number of misclassifications decrease exponentially as we increase the values of P_C and P_S parameters. This is expected because TCAMChecker performs lookup verification more often for higher values of P_C and TCAM scrubbing refreshes the TCAM entries from the backup table more often for higher values of P_S (Fig. 5).

The results of our experiments have clearly showed that TCAMChecker outperforms TCAM scrubbing in terms of accuracy. The improved performance of TCAMChecker is attributed to the fact that it intelligently detects and corrects only erroneous TCAM entries by comparing the results of two TCAMs operating in parallel. The underlying principle of TCAMChecker is based on Theorem 1, which states that at least one error is detected (and corrected) if the results of two TCAMs mismatch. TCAM scrubbing, on the other hand, blindly refreshes entries from the backup table.

5.4 Selecting Appropriate P_C or P_S Values

We now provide insights to choose an appropriate values of P_C and P_S based on our findings from experimental evaluations. The effectiveness of both TCAMChecker and TCAM scrubbing have a strong dependence on the error probability and the size of filter sets. Therefore, the appropriate value of P_C and P_S is also dependent on the above-mentioned factors. To reduce the number of misclassifications, values of P_C and P_S should be chosen proportional to the estimated error probability values. The size of filter sets is also proportional to the number of misclassifications; therefore, higher values of P_C and P_S should be chosen for larger filter sets. Given the fact that modern TCAMs are fairly reliable, it is reasonable to assume that error probabilities in real-world TCAMs are significantly low with $P_e < 0.1$ in most cases [22, 23]. However, we do not have any control over the size of filter sets, which can typically



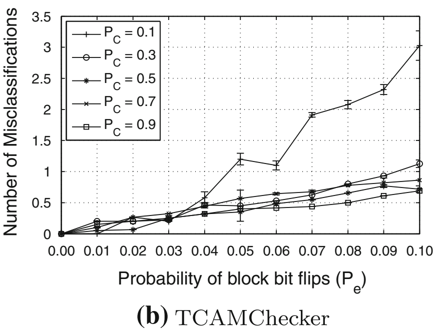


Fig. 4 Comparison of TCAM scrubbing and TCAMChecker for block errors while varying values of P_S, P_C and P_e. The results are reported for a filter set with 136 rules and the accompanying network trace containing approximately 136 thousand packet headers

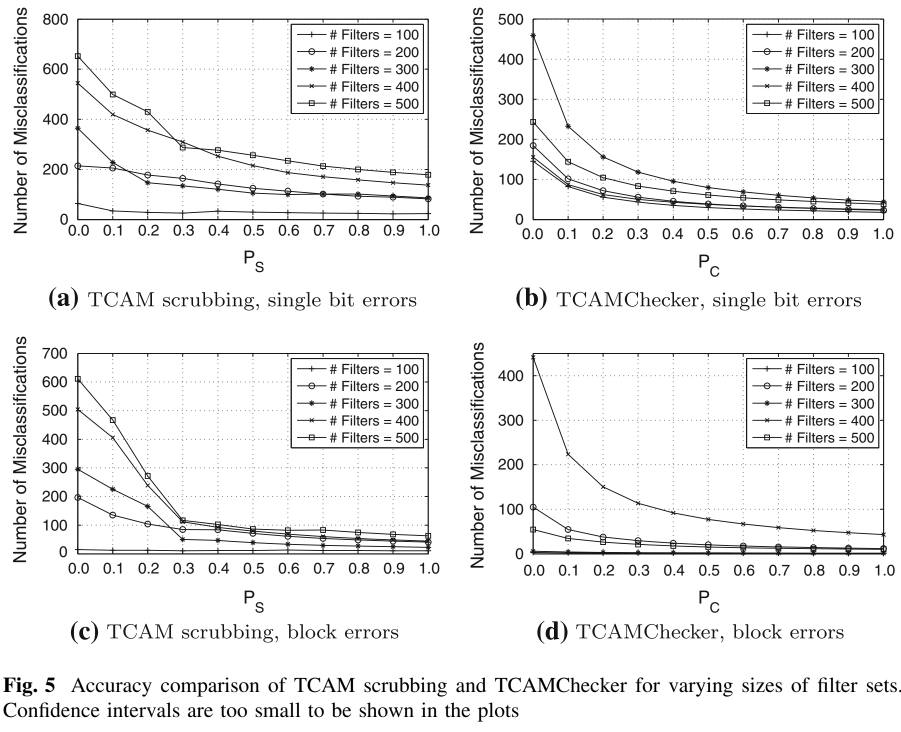


Fig. 5 Accuracy comparison of TCAM scrubbing and TCAMChecker for varying sizes of filter sets. Confidence intervals are too small to be shown in the plots

contain several hundred filters. Therefore, a moderate value of P_C or $P_S \approx 0.5$ should be chosen to keep the number of misclassifications in check.

6 Conclusions

This paper represents the first step towards software-based TCAM error detection and correction. Our proposed technique, TCAMChecker, has three major advantages over prior schemes. First, TCAMChecker requires no hardware modification, which makes it cheap and practical to be implemented in existing and future systems. Second, TCAMChecker provides mechanisms for system designers to flexibly balance between performance and reliability. Third, TCAMChecker is not constrained by the number of bit errors within a single TCAM entry.

Acknowledgments The authors would like to thank anonymous reviewers whose precise suggestions considerably helped to improve the quality of the manuscript. The work of Chad Meiners and Alex X. Liu is partially supported by the National Science Foundation under Grant Numbers CNS-0716407, CNS-0916044, and CNS-0845513. The work of Zheng Qin is partially supported by the information security industrialization fund from NDRC of China in 2009 (No. [2009]1886); the major achievements transfer projects of MOF and MIIT of China in 2010 (No. CJ [2010] 341); NSF of Hunan Province, China (No. 09JJ3124), and the National Science Foundation of China under Grant No. 61070194. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the above mentioned funding agencies.

References

1. Meiners, C.R., Liu, A.X., Torng, E.: Topological transformation approaches to optimizing tcam-based packet processing systems. In: Proceedings of the ACM SIGMETRICS, pp. 73–84 (2009)
2. Baumann, R.: Soft errors in advanced computer systems. IEEE Des. Test Comput. **22**(3), 258–266 (2005). doi:10.1109/MDT.2005.69
3. Schrimpf, R., Fleetwood, D. (eds): Radiation Effects and Soft Errors in Integrated Circuits and Electronic Devices. World Scientific, Singapore (2004)
4. Bremler-Barr, A., Hay, D., Hendler, D., Roth, R.: PEDS: a parallel error detection scheme for TCAM devices. In: Proceedings of the 29th Conference on Computer Communications (INFOCOM), pp. 1296–1304 (2009). doi:10.1109/INFCOM.2009.5062044
5. Mastipuram, R., Wee, E.C.: Cypress Semiconductor: Soft Errors' Impact on System Reliability. Cypress Semiconducto—EDN, San Jose (2004)
6. Azizi, N., Najm, F.N.: A family of cells to reduce the soft-error-rate in ternary-CAM. In: Proceedings of the ACM/IEEE Design Automation Conference, pp. 779–784 (2006). doi:10.1109/DAC.2006.229229
7. Krishnan, S.C., Panigrahy, R., Parthasarathy, S.: Error-correcting codes for ternary content addressable memories. IEEE Transactions On Computers **58**(2), 275–279 (2009)
8. Noda, H., Dosaka, K., Mattaush, H.J., Koide, T., Morishita, F., Arimoto, K.: A reliability-enhanced TCAM architecture with associated embedded DRAM and ECC. IEICE Transactions on Electronics **E89-C**, 1612–1619 (2006)
9. Noda, H., Inoue, K., Kuroiwa, M., Igaue, F., Yamamoto, K., Mattausch, H.J., Koide, T., Amo, A., Hachisuka, A., Soeda, S., Hayashi, I., Morishita, F., Dosaka, K., Arimoto, K., Fujishima, K., Anami, K., Yoshihara, T.: A cost-efficient high-performance dynamic TCAM with pipelined hierarchical searching and shift redundancy architecture. IEEE Solid-State Circuits **40**(1), 245–253 (2005). doi:10.1109/JSSC.2004.838016
10. Lakshminarayanan, K., Rangarajan, A., Venkatachary, S.: Algorithms for advanced packet classification with ternary CAMs. In: Proceedings of the ACM SIGCOMM, pp. 193–204 (2005)
11. Avizienis, A.: The n-version approach to fault tolerant software. IEEE Trans. Softw. Eng. **SE-11**(12), 1491–1501 (1985)
12. Avizienis, A.: The methodology of n-version programming. In: Lyu, M.R. (eds) Chapter 2 of Software Fault Tolerance., pp. 23–46. Wiley, New York (1995)
13. Vouk, M.A.: On back-to-back testing. In: Proceedings of the Annual Conference on Computer Assurance (COMPASS), pp. 84–91 (1988)
14. Anderson, H., Hagelin, G.: Computer controlled interlocking system. Ericsson Rev. **58**(2), 74–80 (1981)
15. Traverse, P.: Airbus and atr system architecture and specification. In: Voges, U. (eds) Software Diversity in Computerised Control Systems., Springer, New York (1988)

16. Condor, A., Hinton, G.: Fault tolerant and fail-safe design of CANDU computerised shutdown systems. IAEA Specialist Meeting on Microprocessors important to the Safety of Nuclear Power Plants (1988)
17. Lambiri, C.: Senior staff architect IDT. Private communication (2008)
18. Pagiamtzis, K., Azizi, N., Najm, F.N.: A soft-error tolerant content-addressable memory (cam) using an error-correcting-match scheme. In: Proceedings of the IEEE Custom Integrated Circuits Conference, pp. 301–304 (2006). doi:10.1109/CICC.2006.320887
19. Chao, H.J., Liu, B.: High performance switches and routers. John Wiley, New Jersey (2007)
20. Lekkas, P.C.: Network Processors—Architectures, Protocols, and Platforms. McGraw-Hill, New York (2003)
21. Taylor, D.E., Turner, J.S.: Classbench: A packet classification benchmark. In: Proceedings of the IEEE Infocom, pp. 2068–2079 (2005)
22. Baeg, S., Wen, S., Wong, R.: Minimizing soft errors in TCAM devices: a probabilistic approach to determining scrubbing intervals. IEEE Transactions on Circuits and Systems **57**(4), 814–822 (2010)
23. Mukherjee, S., Emer, J., Fossum, T., Reinhardt, S.: Cache scrubbing in microprocessors: myth or necessity? In: IEEE Pacific Rim International Symposium on Dependable Computing, pp. 37–42 (2004)

Author Biographies

M. Zubair Shafiq received his B.E. degree in Electrical Engineering from National University of Sciences and Technology, Islamabad, Pakistan in 2008, where he received Dean's plaque of excellence for excellent undergraduate research in 2007 and 2008. He is currently a Ph.D. Candidate in the Department of Computer Science and Engineering at the Michigan State University, where he received recognition for outstanding research at 2011 Engineering Graduate Research Symposium. His research interests are in the areas of cellular networks, online social networks, and content distribution networks with emphasis on measurement, performance analysis, and mathematical modeling.

Chad Meiners received his Ph.D. degree in Computer Science at Michigan State University in 2009. He is currently full technical staff at MIT Lincoln Laboratory. His research interests include formal methods, networking, algorithms, and cyber security.

Zheng Qin is a professor at Hunan University, China. He received his Ph.D. degree in Computing Science from Chongqing University, China, in 2001. His main interests are computer network, information security, and cloud computing.

Ke Shen received his Master degree in computer science from Michigan State University in 2010. He is currently a software engineer in the Digital Design and Engineering Department at Autodesk Inc. His research interests include networking, algorithms, and computer graphics.

Alex X. Liu received his Ph.D. degree in computer science from the University of Texas at Austin in 2006. He is currently an assistant professor in the Department of Computer Science and Engineering at Michigan State University. He received the IEEE & IFIP William C. Carter Award in 2004 and an NSF CAREER award in 2009. He received the MSU College of Engineering Withrow Distinguished Scholar Award in 2011. His research interests focus on networking, security, and dependable systems.