IRWIN AND JOAN JACOBS
CENTER FOR COMMUNICATION AND INFORMATION TECHNOLOGIES

TIMEFLIP: Scheduling Network Updates with Timestamp-based TCAM Ranges

Tal Mizrahi, Ori Rottenstreich and Yoram Moses

CCIT Report #877
January 2015

DEPARTMENT OF ELECTRICAL ENGINEERING
TECHNION - ISRAEL INSTITUTE OF TECHNOLOGY, HAIFA 32000, ISRAEL

CCIT Report #877 January 2015

TIMEFLIP: Scheduling Network Updates with Timestamp-based TCAM Ranges

Technical Report◇, January 2015

Tal Mizrahi*
Technion
Email: dew@tx.technion.ac.il

Ori Rottenstreich
Princeton University
Email: orir@cs.princeton.edu

Yoram Moses†*
Technion
Email: moses@ee.technion.ac.il

***Abstract*—Network configuration and policy updates occur frequently, and must be performed in a way that minimizes transient effects caused by intermediate states of the network. It has been shown that accurate time can be used for coordinating network-wide updates, thereby reducing temporary inconsistencies. However, this approach presents a great challenge; even if network devices have perfectly synchronized clocks, how can we guarantee that updates are performed at the exact time for which they were scheduled?**

In this paper we present a practical method for implementing *accurate* time-based updates, using TIMEFLIPs. A TIMEFLIP is a time-based update that is implemented using a timestamp field in a Ternary Content Addressable Memory (TCAM) entry. TIMEFLIPs can be used to implement Atomic Bundle updates, and to coordinate network updates with high accuracy. We analyze the amount of TCAM resources required to encode a TIMEFLIP, and show that if there is enough flexibility in determining the scheduled time, a TIMEFLIP can be encoded by a single TCAM entry, using a single bit to represent the timestamp, and allowing the update to be performed with an accuracy on the order of 1 microsecond.

I. INTRODUCTION

A. Background

Network updates are a routine necessity; policy changes or traffic-engineered route changes may occur frequently, and often require network devices to be reconfigured. This challenge is especially critical in Software Defined Networks (SDN), where the control plane is managed by a logically centralized controller, and configuration updates occur frequently. Such configuration updates can involve multiple network devices, potentially resulting in temporary anomalies such as forwarding loops or packet loss.

Network devices such as routers and switches use TCAMs for various purposes, e.g., packet classification, Access Control Lists (ACLs), and forwarding decisions. TCAMs are an essential building block in network devices. A typical example for the importance of TCAMs is OpenFlow [2], [3]. An OpenFlow switch performs its functionality using one or more *flow tables*, most commonly implemented by TCAMs (see, e.g., [4], [5]).

The order of the entries in a TCAM determines their priority. Hence, installing a new TCAM entry often involves rearranging existing entries, yielding high overhead for each TCAM update. It has been shown [6] that the latency of a TCAM rule installation may vary from a few milliseconds to a few seconds.

A recently introduced approach [7], [8] proposes to use accurate time and local clocks as a means to coordinate network updates. By using synchronized clocks, configuration changes can be scheduled in a way that guarantees a coordinated network-wide update, thereby reducing transient anomalies. One of the main challenges in this approach is to guarantee that scheduled updates are performed *accurately* according to the desired schedule. Even if the clocks in the network are perfectly synchronized, performing configuration changes requires a potentially complex procedure that may be completed at an uncertain time.

B. Introducing TIMEFLIPs

In this paper we present a method that uses **TIMEFLIPs** to perform *accurate* time-based network updates. We define a TIMEFLIP to be a scheduled update that is implemented using TCAM ranges to represent the scheduled time of operation. We analyze TCAM lookups (Fig. 1) that take place in network devices, such as switches and routers. We assume that the device maintains a local clock, and that a timestamp T recording the local arrival time is associated with every packet that is received by the device. Typically, TCAM search keys consist of fields from the packet header, as well as some additional metadata. In our setting, the metadata includes a timestamp T. Hence, a TCAM entry can specify a range relative to the timestamp T, as a way of implementing time-based decisions. The timestamp T is not integrated into the packet, as it is only used internally in the device, and thus does not compromise the traffic bandwidth of the network device.

Using a simple microbenchmark, we show that TIMEFLIPs can be performed by existing network devices, and analyze the achievable scheduling accuracy of TIMEFLIPs. Accurate clock synchronization has become a common feature in network devices; the Precision Time Protocol (PTP), based on the IEEE 1588 standard [9], typically provides an accuracy on the order of 1 microsecond [10], [11]. We show that the accuracy at

◇This technical report is an extended version of [1], which was accepted to IEEE INFOCOM '15, Hong Kong, April 2015.

*This work was supported in part by the ISF grant 1520/11.

†The Israel Pollak academic chair at Technion.

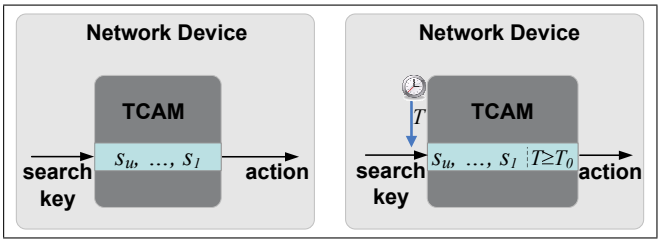


Fig. 1: TCAM lookup: conventional vs. time-based.

which a TIMEFLIP is executed compared to its scheduled time is two orders of magnitude more accurate, and hence that network-wide updates can be timed with a 1 μsec accuracy using PTP.

TIMEFLIPs enable two interesting scenarios:

(i) **Atomic Bundle.** It is sometimes desirable to reconfigure a network device by applying a set of configuration changes as a bundle, i.e., every packet should be processed either before any of the modifications have been applied, or after all have been applied. The *Atomic Bundle* feature in OpenFlow [3] defines such functionality; the OpenFlow specification suggests that Atomic Bundles can be implemented either by temporarily queuing packets during the update, or by using double buffering techniques. Both approaches may yield significant cost in terms of resources. TIMEFLIPs allow a clean and natural way to implement Atomic Bundles; the set of configuration changes can be enabled at all times $T \geq T_0$ for some chosen time T_0, and the timestamp T defines when the bundle commands atomically come into effect.

(ii) **Network-wide coordinated updates.** If network devices use synchronized clocks, then TIMEFLIP can be used for updating different devices at the same time,[1] or for defining a set of scheduled updates according to a specific order [7].

TIMEFLIPs require every TCAM entry to include a timestamp field. We show that this per-entry overhead is relatively small. Moreover, since TCAMs have fixed entry sizes, it is often the case that a portion of the TCAM entry is unused. For example, in many cases TCAM entries are used to store the IPv4 5-tuple, requiring 104 bits, while the smallest TCAM entry that can accommodate the 5-tuple is typically larger, e.g., 144 bits [12] or 160 bits [13], leaving a large number of unused bits that can be used for the timestamp field.

As TCAM resources are scarce and costly, we aim to represent the timestamp field by as few bits as possible, and each TIMEFLIP by as few TCAM entries as possible. Optimal representation of TCAM ranges is a problem that has been widely studied in the literature (e.g., [14], [15]). The problem we address has two unique properties that, to the best of our knowledge, have not been previously analyzed:

- The range values can be *chosen* in a way that minimizes the number of TCAM entries. If the time T_0 for which a time-based network update is scheduled can be selected within a *scheduling tolerance* (Fig. 2), given by a range of time values $[T_{min}, T_{max}]$, then the number of entries required to represent the range can be reduced. Notably, the scheduling tolerance *does not* compromise the accuracy of the TIMEFLIP. It only presents some flexibility in choosing T_0; an SDN controller may choose any T_0 withing the given range, but once T_0 is chosen, the TIMEFLIP should occur *accurately* at T_0.
- If some of the most significant bits of the timestamp value are known to be constant during the TIMEFLIP, the network device can simply ignore these bits, by placing '*don't care*' values in the respective bits in the TCAM. For example, the portion of the timestamp that represents the date is known to be constant during a TIMEFLIP, and thus can be ignored. We refer to time ranges that ignore some of the most significant bits as *periodic ranges*, and show that the use of periodic ranges allows to represent the time ranges by fewer TCAM entries.

C. Contributions

The main contributions of this paper are:

- We introduce TIMEFLIPs and show how to **accurately** perform coordinated network updates and Atomic Bundle updates using them.
- We present an optimal scheduling algorithm: by correctly choosing the update time, T_0, the number of TCAM entries used for representing the timestamp range can be significantly reduced.
- Our analysis provides an upper bound on the number of TCAM entries required for representing a TIMEFLIP.
- We analyze the number of bits required for representing the timestamp field in TCAM entries, and show that it is a function of the *scheduling tolerance*.
- We show that in a system where the *scheduling tolerance* is sufficiently relaxed, using *periodic ranges*, the timestamp field can be represented by a single bit in the TCAM entry, and every TIMEFLIP requires a single TCAM entry.
- We use a microbenchmark to demonstrate that our approach can be effectively used to schedule accurate time-based updates with existing commercial network devices.

D. Related Work

Consistent network updates have been widely analyzed in the literature. A common approach to avoiding inconsistencies during topology updates in routing protocols is to use a sequence of configuration commands [16], whereby the order of execution guarantees that no anomalies are caused in intermediate states of the procedure. Another recently introduced approach for consistent updates [17] uses configuration version

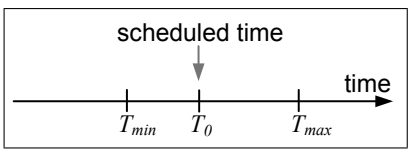


Fig. 2: Scheduling tolerance: $T_0 \in [T_{min}, T_{max}]$.

[1] Subject to the accuracy of the clock synchronization mechanism.

tags to guarantee consistency. Dynamic traffic engineering [6], [18] has been shown to require frequent topology updates that must be applied carefully to optimize the network utilization.

Time and synchronized clocks are used in various distributed applications, from mobile backhaul networks [10] to distributed databases [19]. OpenFlow [3] uses *timeouts* for expiring old forwarding rules. The controller can define a timeout for a flow rule, causing it to be removed when the timeout expires. However, timeouts are defined with a coarse granularity of 1 second, and thus do not allow delicate coordination. Moreover, since timeouts are by nature *relative*, they do not allow the accurate coordination that *absolute* time can provide. In [20] the authors observed that it would be interesting to explore using time synchronization to reconfigure routers at a specific time. In [17] the authors argued that a simultaneous network update does not guarantee consistency, since packets that are en-route during the update may be processed by a mixture of configurations. However, it was shown [7] that time can be used not only to perform simultaneous configuration changes, but also to perform time-based update procedures that guarantee consistent updates.

The Interface to the Routing System (I2RS) working group of the Internet Engineering Task Force (IETF) has recognized the value of time-based state installations [21], but decided not to pursue this concept, as the ability to accurately perform timed installations was not considered viable [22]. **Contrarily, we show that accurate time-based updates are in fact viable**; the current paper proposes a novel approach that enables accurate time-based updates in switches and routers, and demonstrates their applicability to existing network devices.

The simplest way to encode a range in TCAMs is known as *prefix encoding* [14]. In this scheme the set of values that match a rule is presented as a union of prefix TCAM entries (with a sequence of *don't care*s as a suffix of the entry), representing disjoint sets of values. For instance, if we denote by W the number of bits used for encoding the range, then for $W = 4$ the range $[4, 14]$ can be encoded by the four TCAM entries $(01{**}), (10{**}), (110{*}), (1110)$ that respectively represent the ranges $[4, 7], [8, 11], [12, 13]$ and $[14, 14]$, whose union is the requested range $[4, 14]$. With this encoding, any range defined on W bits can be encoded with at most $2W-2$ entries. Moreover, an extremal range of the form $[T_0, 2^W-1]$ or $[0, T_0]$ can be represented using at most W entries. By using complementary ranges these two bounds were improved to W and $\left\lceil \frac{W+1}{2} \right\rceil$, respectively [23]. An alternative encoding that relies on the Gray-code representation of ranges was shown to improve the maximal expansion to $2W-4$ for general ranges [15]. While the above works concentrated on the encoding of a single range, a wide literature discusses efficient encodings of classifiers with an ordered list of range rules, [24]–[31].

II. Model and Notations

A. TCAM Entries

A TCAM is an associative memory device that allows fast classification. It compares a search string against a table of stored entries, and returns the address of the matching data. Each address is associated with a specific *action*. Each TCAM bit can have one of three possible values, 0, 1, or $*$, with the latter representing the *don't care* value. The order of entries in a TCAM determines their priority. A TCAM search returns the address of the first entry that matches the search key.

Our analysis focuses on a TCAM lookup that is performed by a *network device*, or a *device* for short. We assume that the device has access to a clock, and that before a TCAM lookup the device produces a timestamp T, which is obtained by capturing the value of the clock at some instant before the TCAM lookup. For example (Fig. 1), the device can capture the timestamp T for each received packet immediately upon its arrival. The timestamp, together with the packet header, will serve as an input to the TCAM lookup.

A TCAM entry is denoted by $S \rightarrow a$, where $S = (\sigma_U, \ldots, \sigma_1) \in \{0, 1, *\}^U$. The number of bits in a TCAM entry is denoted by U, where $0, 1$ are bit values and $*$ stands for *don't care*. We denote a sequence of m *don't care* bits by $(*^m)$. A bit that is assigned the *don't care* value is said to be *masked*. The set of possible actions is denoted by $\mathcal{A}$, where an individual action is denoted by $a \in \mathcal{A}$.

Specifically, throughout our analysis S will have the form $(s_u, \ldots, s_1, t_W, \ldots, t_1)$ such that $u+W = U$, and $t_W, \ldots, t_1$ represent the bits corresponding to the timestamp T.

We denote the m most significant bits of the timestamp T by T^{m^+}, and the k least significant bits of T by T^{k^-}.

We define a ***time-based TCAM entry*** to be an entry in which at least one of the bits $t_W, \ldots, t_1$ has a value in $\{0, 1\}$, whereas in a ***time-oblivious entry*** all the bits $t_W, \ldots, t_1$ are $*$.

A ***time range*** is defined to be an interval $[T_1, T_2]$, where T_1 and T_2 are W-bit integers. A *time range rule* is denoted by $(s_u, \ldots, s_1, [T_1, T_2])$, or equivalently, $(s_u, \ldots, s_1, [T_1 \leq T \leq T_2])$. Such a rule can be represented by one or more time-based TCAM entries. The *rule expansion* of a range $[T_1, T_2]$ is the minimal number of entries that can be used for representing the range. In the context of this paper we focus on ***prefix-based expansions*** [14], in which only prefix entries are used.

An ***extremal range*** is a range that has one of two possible forms, either a *right extremal range*, which has the form $[T_1, 2^W - 1]$, also denoted by $T \geq T_1$, or a *left extremal range*, $[0, T_2]$. We denote by $r(T_1)$ the prefix-based expansion of a right range, $[T_1, 2^W - 1]$, and by $\ell(T_2)$ the expansion of the left range $[0, T_2 - 1]$. Note that $\ell(0)$ is undefined.

B. TimeFlip: Theory of Operation

Consider a coordinated network update that is due to take place at time T_0 and requires a TCAM update. The naïve approach to update the TCAM is to schedule the device's management software[2] to perform the required modification as close as possible to T_0. This approach allows a limited degree of accuracy, which depends on the operating system's ability to perform real-time operations, and on the load caused by other tasks that run on the CPU.

[2]A network device typically runs a software layer that performs various tasks, including TCAM management.

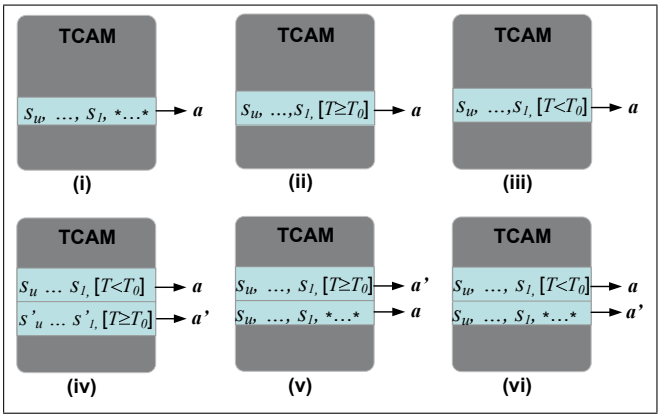


Fig. 3: A timed TCAM Update. Every line in the figure is a time range rule, represented by one or more TCAM entries. (i) Time-oblivious entry. (ii) Installation. (iii) Removal. (iv) Rule update. (v) Action update. (vi) Action update using a complementary timestamp range.

In our approach the TCAM management software installs the time-based TCAM entries ahead of the activation time, T_0, allowing the update to be applied precisely at T_0. After time T_0 the management software performs cleanup operations, e.g., removing rules that apply only to times $T < T_0$.

We address four main classes of TIMEFLIPS (see Fig. 3):

(i) **Installation.** A new TCAM rule $S \rightarrow a$ is installed, effective from time T_0. A timed installation is a rule of the form $(s_u, \ldots, s_1, [T \geq T_0]) \rightarrow a$.

(ii) **Removal.** An existing TCAM rule $S \rightarrow a$ is scheduled to be deactivated at time T_0, using a rule of the form $(s_u, \ldots, s_1, [T < T_0]) \rightarrow a$.

(iii) **Rule update.** An existing rule $S \rightarrow a$ is modified to $S' \rightarrow a'$ at T_0. A rule update can simply be represented as a pair of rules, one for removal and one for installation.

(iv) **Action update.** The action of an existing TCAM rule is modified from a to a' at time T_0. Hence, prior to T_0 the management software *installs* a rule of the form $(s_u, \ldots, s_1, [T \geq T_0]) \rightarrow a'$ that precedes the existing $S \rightarrow a$. The first-match behavior of TCAMs implies that if $T \geq T_0$, the search matches the newly installed rule, and a' is performed, whereas at any time before T_0 the $S \rightarrow a$ rule prevails. After time T_0 the TCAM management software *removes* the excess rules: the rule $S \rightarrow a$ is deleted, and $(s_u, \ldots, s_1, [T \geq T_0]) \rightarrow a'$ is replaced by $(s_u, \ldots, s_1, *, \ldots, *) \rightarrow a'$, requiring a single entry. As shown in Fig. 3(vi), an alternative representation of the action update uses a rule that maps $T < T_0$ to a, and S to a'. Complementary encoding [23], [32], also known as *negative encoding*, allows in some cases to represent the rule more efficiently, as further discussed in Section IV.

Our analysis in this paper focuses on timed installations. The other three cases are derived from this analysis: timed *removal* is similar, except that it uses left time ranges of the form $T < T_0$. A *rule update* is simply implemented by a superposition of two rules (Fig. 3(iv)), one for installation and one for removal. In Section IV we discuss *action updates*, and show how complementary encoding can be used in this context, allowing more efficient TCAM usage.

C. Timed Installation: Formal Definition

Let R be a time range. Given a time-oblivious TCAM entry $S \rightarrow a$ with $S = (s_u, \ldots, s_1, *, \ldots, *)$, we define a ***timed installation*** of $S \rightarrow a$ over R to be a TCAM rule $S_R \rightarrow a$, such that $S_R := (s_u, \ldots, s_1, R)$. Hence, a is activated during the time range R.

We define the *expansion of a timed installation* over a time range R to be the expansion of the range R.

Since R is a time range, S_R is represented by one or more entries in the TCAM. We note that even if more than one entry is used to represent S_R, the excess entries are required only for a brief period of time; we assume that shortly after S_R is activated the TCAM management software performs a cleanup, leaving only a single entry, representing $S \rightarrow a$.

III. OPTIMAL TIME-BASED RULE INSTALLATION

A. Optimal Scheduling

It has been shown [14] that an extremal range of the form $[T_0, 2^W - 1]$ can be represented using at most W entries. However, we observe that a careful selection of the value T_0 can significantly reduce the number of entries required for representing this update. The update time T_0 may be tuned to an optimal value in the following scenarios:

(i) In Atomic Bundle updates, a network device is required to perform a set of changes atomically, without strict timing constraints, and hence it is flexible to select the time T_0 at which these changes are performed.

(ii) In network-wide coordinated updates, *optimal scheduling* can be enforced by a central entity that determines the update time, e.g., a Network Management Station (NMS) or an SDN controller. The central entity's goal is to find a value of T_0 (within a set of allowed values) that will minimize the timestamp range expansion; the underlying assumption is that all network devices use the same format to represent the timestamp, and the same TCAM range encoding scheme.

As depicted in Fig. 2, we assume that the value of T_0 is determined by a scheduling algorithm, subject to the constraint $T_{min} \leq T_0 \leq T_{max}$. We define the *scheduling tolerance*, denoted by TOL, to be $T_{max} - T_{min} + 1$. This is the number of allowable values for T_0.

We wish to study how T_0 should be selected. As a first step we learn the expansion of a specific extremal range $[T_0, 2^W - 1]$. Property 1, based on [14], shows that the expansion of this range is given by the number of '1'-s in a binary representation of $2^W - T_0$.

Property 1. *The expansion $r(T_0)$ of a right range $[T_0, 2^W - 1]$ is given by the number of '1'-s in a binary representation of the number of values in the range, $2^W - 1 - T_0 + 1 = 2^W - T_0$.*

Proof Outline. The property follows from the definition of the prefix encoding as defined in [14]. The encoding is composed of entries that consider disjoint sets of inputs. The cardinality of each set is a power of 2, and distinct sets have different

cardinalities. The sum of the cardinalities equals the number of values in the range. □

Example 1. *For $W = 4$, the range $[T_0, 2^W - 1] = [9, 15]$ includes $15-9+1 = 7$ values. The binary representation of 7 is 0111, which has three '1'-s and accordingly the range can be encoded using the three entries $(1001), (101*), (11**)$. Likewise, the range $[11, 15]$ has $15 - 11 + 1 = 5 = 0101$ values and can be encoded by the two entries $(1011), (11**)$. The range $[15, 15]$ has a single value ($1 = 0001$) and can be encoded by the single entry (1111).*

By symmetry, we can show that the expansion $\ell(T_0)$ of the range $[0, T_0-1]$ is given by the '1'-s in a binary representation of the number of values in the range, $T_0 - 1 - 0 + 1 = T_0$.

The next theorem relates the expansion of a right range $[T_0, 2^W - 1]$ and a left range $[0, T_0 - 1]$.

Theorem 1. *For $W \geq 1$ and $T_0 \in [1, 2^W - 1]$ the expansion $r(T_0)$ of the right range $[T_0, 2^W - 1]$ and the expansion $\ell(T_0)$ of the left range $[0, T_0 - 1]$ satisfy $r(T_0) + \ell(T_0) \leq W + 1$.*

Proof. Let $B_r = 2^W - 1 - T_0 + 1 = 2^W - T_0$ and $B_\ell = T_0 - 1 + 1 = T_0$ be the number of values in the right and the left ranges, respectively. Clearly, $B_r + B_\ell = 2^W$. Let $(b_{r,W}, \cdots, b_{r,1})$ and $(b_{\ell,W}, \cdots, b_{\ell,1})$ be the binary representations of B_r, B_ℓ. Let $n_r = \Sigma_{i\in[1,W]} b_{r,i}$ and $n_\ell = \Sigma_{i\in[1,W]} b_{\ell,i}$. By Property 1, we have that $r(T_0) = n_r$ and $\ell(T_0) = n_\ell$. We show the result by induction. First, if $W = 1$, we have that $T_0 = 1$. The right range $[1, 1]$ and the left range $[0, 0]$ can be both encoded in a single entry and $r(T_0) + \ell(T_0) = 1 + 1 = 2 \leq W + 1$. For a general W, we distinguish between the two following sub-cases. If $b_{r,1} = 1$, i.e. B_r is odd (and accordingly $b_{\ell,1} = 1$, i.e. B_ℓ is odd as well since $B_r + B_\ell = 2^W$), we have that $(2^W - 1) - B_r = B_\ell - 1$. Since $(2^W - 1)$ can be represented by a binary vector with W '1'-s, the number of '1'-s in $(2^W - 1) - B_r$ is $W - n_r$. By the last equality this equals $n_\ell - 1$, the number of '1'-s in $B_\ell - 1$. We then have that $W - n_r = n_\ell - 1$ and $r(T_0) + \ell(T_0) = n_r + n_\ell = W + 1$. In the second sub-case $b_{r,1} = b_{\ell,1} = 0$ and B_r, B_ℓ are even. Then, $r(T_0) = n_r = \Sigma_{i\in[1,W]} b_{r,i} = \Sigma_{i\in[2,W]} b_{r,i}$ and $\ell(T_0) = n_\ell = \Sigma_{i\in[1,W]} b_{\ell,i} = \Sigma_{i\in[2,W]} b_{\ell,i}$. We now consider $T_0' = 0.5 \cdot T_0$ with $W' = W - 1$ and examine the expansions $r(T_0')$ and $\ell(T_0')$ within the space $[0, 2^{W'} - 1]$. We have that $2^{W'} - T_0' = 0.5 \cdot (2^W - T_0)$ and the number of values in $[T_0', 2^{W'} - 1]$ is represented by $(b_{r,W}, \cdots, b_{r,2})$ and $r(T_0') = \Sigma_{i\in[2,W]} b_{r,i} = r(T_0)$. Likewise, since $T_0' = 0.5 \cdot T_0$ the number of values in $[0, T_0' - 1]$ is represented by $b_{\ell,W}, \cdots, b_{\ell,2}$ and $\ell(T_0') = \Sigma_{i\in[2,W]} b_{\ell,i} = \ell(T_0)$. Accordingly, $r(T_0) + \ell(T_0)$ (in W) equals the sum of the expansions $r(T_0') + \ell(T_0')$ in $W' = W - 1$. By the induction hypothesis we have that $r(T_0') + \ell(T_0') \leq W' + 1 = W - 1 + 1 = W \leq W + 1$ and the result follows. □

We can now introduce the SCHEDULE algorithm (Fig. 4), which computes an optimal value, T_{SCH}, for a given range $[T_{min}, T_{max}]$. Throughout the paper we use the notation T_{SCH}, defined by $T_{\text{SCH}} :=$ SCHEDULE(T_{min}, T_{max}, W), and the range R_{SCH}, defined by $R_{\text{SCH}} := [T_{\text{SCH}}, 2^W - 1]$.

```
SCHEDULE(T_min, T_max, W)
1  t_0 ← 0; i ← 0
2  while t_i ∉ [T_min, T_max]
3    i ← i + 1
4    if t_{i-1} < T_min
5      t_i ← t_{i-1} + 2^{W-i}
6    else
7      t_i ← t_{i-1} − 2^{W-i}
8  T_SCH ← t_i
9  return T_SCH
```

Fig. 4: Optimal scheduling algorithm; no other scheduling algorithm produces an extremal range with a lower expansion.

Intuitively, SCHEDULE performs a binary search over the range $[0, 2^W - 1]$, and returns the first value that falls within $[T_{min}, T_{max}]$. Notably, we shall see that due to the nature of the binary search, SCHEDULE returns the value T_{SCH} with the fewest '1'-s in the binary representation of $2^W - T_{\text{SCH}}$, and hence, by property 1 minimizes $r(T_{\text{SCH}})$. In terms of complexity, the number of iterations in SCHEDULE is bounded by W, as it is a binary search over a range of 2^W values.

The following theorem states that SCHEDULE is optimal, i.e., that no other scheduling algorithm produces an extremal range with a lower expansion.

Theorem 2. *If $T_{\text{SCH}} =$ SCHEDULE(T_{min}, T_{max}, W), then $r(T_{\text{SCH}}) \leq r(T)$ for all $T \in [T_{min}, T_{max}]$.*

Proof. Clearly, if $T_{min} = 0$, then $T_{\text{SCH}} = 0$, and the range $[T_{\text{SCH}}, 2^W - 1]$ can be represented by a single entry, $(*^W)$. For $T_{min} > 0$, without loss of generality, T_{SCH} is determined by SCHEDULE after m iterations, i.e., $i = m$ on line 8 of SCHEDULE. We prove the claim by induction on $m \geq 1$. Denote $2^W - T_{\text{SCH}}$ by B. By property 1, $r(T_{\text{SCH}})$ is equal to the number of '1'-s in the representation of B. For $m = 1$ we have that $T_{\text{SCH}} = 2^{W-1}$, and thus $B = 2^{W-1}$. Since the binary representation of B is $(10\ldots0)$, we have $r(T_{\text{SCH}}) = 1$, which is of course optimal. Now we assume the claim holds for every T'_{SCH} that is computed when SCHEDULE returns after m iterations. Let T_{SCH} be a value returned by SCHEDULE after $m + 1$ iterations. We distinguish between two cases: (i) $T_{\text{SCH}} > 2^{W-1}$: we now ignore the most significant bit of the timestamp field, and reexamine the algorithm's outcome. The algorithm SCHEDULE$(T_{min}^{(W-1)^-}, T_{max}^{(W-1)^-}, W-1)$ returns ${T_{\text{SCH}}}^{(W-1)^-}$ after m iterations, and thus by the induction hypothesis ${T_{\text{SCH}}}^{(W-1)^-}$ is optimal in $[0, 2^{W-1} - 1]$. Now assume by way of contradiction that there exists a time $T_{min} \leq T' \leq T_{max}$ such that $r(T') < r(T_{\text{SCH}})$. Thus, the range $[T', 2^W - 1]$ can be represented by fewer entries than the expansion of $[T_{\text{SCH}}, 2^W - 1]$, and by removing the most significant bit of the rule $[T', 2^W - 1]$ we get that $r(T'^{(W-1)^-}) < r({T_{\text{SCH}}}^{(W-1)^-})$, contradicting the induction

hypothesis. (ii) $T_{\text{SCH}} < 2^{W-1}$: similarly to the first case, by observing the range $[0, 2^{W-1}-1]$ we deduce that ${T_{\text{SCH}}}^{(W-1)^-}$ is obtained after m iterations, and is thus optimal. Assume by way of contradiction that there is a $T' \in [T_{min}, T_{max}]$ such that $r(T') < r(T_{\text{SCH}})$. Denote $2^W - T'$ by B'. Note that $[T_{min}, T_{max}] \subset [0, 2^{W-1}-1]$, since otherwise we would have $2^{W-1} \in [T_{min}, T_{max}]$, and SCHEDULE would terminate after one iteration. Thus, $T' < 2^{W-1}$. It follows that ${B'}^{1^+} = B^{1^+} = 1$. Since $r(T') < r(T_{\text{SCH}})$ in $[0, 2^{W-1}-1]$, by property 1 we have that the number of '1'-s in B' is smaller than the number of '1'-s in B. We conclude that there are less '1'-s in ${B'}^{(W-1)^-}$ than in $B^{(W-1)^-}$, yielding $r({T'}^{(W-1)^-}) < r({T_{\text{SCH}}}^{(W-1)^-})$, which is in contradiction to the induction hypothesis. □

An interesting property of SCHEDULE is presented in Lemma 3: the output of the algorithm, T_{SCH}, has a long sequence of least significant '0' bits. This property allows very efficient prefix encoding of extremal ranges of the form $T \geq T_{\text{SCH}}$.

Lemma 3. *The $\lfloor log_2(TOL) \rfloor$ least significant bits of T_{SCH} are all '0'.*

Proof. We denote $\lfloor log_2(TOL) \rfloor$ by X. For $T_{\text{SCH}} = 0$ we have W bits of '0', and the claim is satisfied. For $T_{\text{SCH}} > 0$, we prove this claim by induction on m, the number of iterations in SCHEDULE. For $m = 1$ we have $T_{\text{SCH}} = 2^{W-1}$ with $W-1$ least significant bits of '0', and since $TOL < 2^W$, we have $X \leq W-1$, and thus the claim is satisfied. We assume that the claim holds for $m-1$, and prove for m. We consider two distinct cases: (i) $T_{\text{SCH}} > 2^{W-1}$: in this case $[T_{min}, T_{max}] \subset [2^{W-1}, 2^W-1]$. By considering the $(W-1)$-bit shifted range $[2^{W-1}, 2^W-1]$, SCHEDULE produces ${T_{\text{SCH}}}^{(W-1)^-}$ after $m-1$ iterations, and thus by the induction hypothesis the X least significant bits are '0', which is true also for T_{SCH}. (ii) $T_{\text{SCH}} > 2^{W-1}$: this case is similar to (i), except that $[T_{min}, T_{max}] \subset [0, 2^{W-1}-1]$, and thus we can run SCHEDULE on $[0, 2^{W-1}-1]$, again concluding from the induction hypothesis that the X least significant bits are '0'. □

The following lemma presents an upper bound on the expansion of the range $T \geq T_{\text{SCH}}$, as a function of the scheduling tolerance. This captures a tradeoff between the scheduling tolerance and the time-based range expansion; it is possible to reduce the expansion of a timed installation by increasing the scheduling tolerance.

Lemma 4. *If $TOL < 2^W$ then $r(T_{\text{SCH}}) \leq W - \lfloor log_2(TOL) \rfloor$.*

Proof. Denote $2^W - T_{\text{SCH}}$ by B, and $\lfloor log_2(TOL) \rfloor$ by X. By Lemma 3, the X least significant bits of T_{SCH} are '0', and hence the X least significant bits of B are '0'. Thus, the number of '1'-s in the representation of B does not exceed $W - X$, and by Property 1 we have $r(T_{\text{SCH}}) \leq W - X$. □

B. Average Expansion

In this section we study the influence of the scheduling tolerance on the average expansion. We concentrate on the average expansion of the prefix-based encoding of ranges of the form $[T_0, 2^W - 1]$ for a given W. Intuitively, for a larger scheduling tolerance TOL within which T_0 should be selected, the flexibility is larger and the expansion for the best of the options is expected to be smaller.

Our model is the following. For a given W and $TOL \in [1, 2^W]$, we examine the possible $[T_{min}, T_{max}]$ values that enable TOL possible options. These are the $2^W - TOL + 1$ values $[0, TOL-1], [1, TOL], [2, TOL+1], \cdots, [2^W - TOL - 1, 2^W - 2], [2^W - TOL, 2^W - 1]$. As we described, for each $[T_{min}, T_{max}]$ we use the SCHEDULE algorithm to calculate a T_0 that has an expansion of $\min_{T_0 \in [T_{min}, T_{max}]} r(T_0)$. Based on Property 1, for $[T_{min}, T_{max}]$, T_{SCH} is the value that minimizes the number of '1'-s in a binary representation of $2^W - T_0 \in [2^W - T_{max}, 2^W - T_{min}]$. We calculate the average value of $r(T_{\text{SCH}})$ among the $2^W - TOL + 1$ possible values of $[T_{min}, T_{max}]$. We denote this value by $\rho(W, TOL)$.

Theorem 5. *For $W \geq 1$ and $TOL \in [1, 2^W]$, let $a = \lceil \log_2(TOL) \rceil$. The average value of the expansion of the easiest-to-encode range according to a set $[T_{min}, T_{max}]$ with TOL possible values is given by*

$$\rho(W, TOL) = \frac{1}{2^W - TOL + 1} \cdot \Big(1 - 2^{W-a} + 2^{W-a-1} \cdot (W - a + 2) \cdot (2^a - TOL + 1) + \Sigma_{i=0}^{W-a-1} 2^{i-1} \cdot (i+2) \cdot (TOL - 1)\Big).$$

Proof. Denote $2^W - T_{min}$ and $2^W - T_{max}$ by B_{min} and B_{max}, respectively. Likewise, let B_0 represent the value of $2^W - T_0$ for some T_0. Intuitively, a value of $[T_{min}, T_{max}]$ defines $[B_{max}, B_{min}]$ of which B_0 can be selected. Let $d \in [0, W]$ be the maximal number of most significant bits such that $B_{min}^{d^+} = B_{max}^{d^+}$. We distinguish between several cases according to the value of d. We first assume that $T_{min} \geq 1$ (and accordingly $B_{min} \leq 2^W - 1$). There are $TOL-1$ possible values of $[B_{max}, B_{min}]$ with $d = 0$. These are the values that include $2^{W-1} - 1$ and 2^{W-1}. The $TOL - 1$ options are $[2^{W-1} - TOL + 1, 2^{W-1}], \cdots, [2^{W-1} - 1, 2^{W-1} + TOL - 2]$. In each of these options we can select $B_0 = 2^{W-1}, T_0 = 2^W - B_0 = 2^{W-1}$ and encode the range $[T_0, 2^W - 1] = [2^{W-1}, 2^W - 1]$ by a single entry. There are $2 \cdot (TOL-1)$ values of $[B_{max}, B_{min}]$ for which $d = 1$. These are $[x + 2^{W-2} - TOL + 1, x + 2^{W-2}], \cdots, [x + 2^{W-1} - 1, x + 2^{W-1} + TOL - 2]$ for $x \in \{0, 2^{W-1}\}$. The $(TOL - 1)$ options with $x = 0$ can be encoded with a single entry while the $(TOL - 1)$ options with $x = 1$ require 2 entries. They require a total number of $1 \cdot (TOL-1) + 2 \cdot (TOL-1) = 2^d \cdot (TOL-1) \cdot (1 + d/2)$ entries. More generally, for $i \in [0, W-a-1]$ there are $2^i \cdot (TOL-1)$ values of $[B_{max}, B_{min}]$ with $d = i$ that require a total number of $2^i \cdot (TOL - 1) \cdot (1 + i/2) = 2^{i-1} \cdot (TOL - 1) \cdot (2 + i)$ entries. For $d = W - a$, there are $2^{W-a} \cdot (2^a - TOL + 1)$ values of $[B_{max}, B_{min}]$. In 2^{W-a} of them, B_0 can be selected such that it has a last bits of 0. Thus these 2^{W-a} values of $[B_{max}, B_{min}]$ require a total number of $2^{W-a} \cdot ((W-a)/2)$ entries. Similarly to the previous detailed cases, the other $2^{W-a} \cdot (2^a - TOL)$ are encoded each with $((W-a)/2 + 1)$

entries on average, requiring a total number of $2^{W-a} \cdot (2^a - TOL) \cdot ((W-a)/2+1) = 2^{W-a-1} \cdot (2^a - TOL) \cdot (W-a+2)$ entries. By summarizing these requirements together with the single entry required for the case of $T_{min} = 0$, we deduce the suggested average for the $2^W - TOL + 1$ possible values of $[T_{min}, T_{max}]$. □

Example 2. *Again, let $W = 4$. For $TOL = 2$, we consider the $2^W - TOL + 1 = 15$ possible ranges $[T_{min}, T_{max}]$: $[0,1], [1,2], [2,3], [3,4], [4,5], [5,6], [6,7], [7,8], [8,9], [9,10], [10,11], [11,12], [12,13], [13,14], [14,15]$. For $TOL = 2$, we have $a = \lceil \log_2(TOL) \rceil = 1$. By Theorem 5, we have that the average expansion here is $\rho(W, TOL) = \frac{1}{15} \cdot (1 - 2^3 + 2^2 \cdot (3+2) \cdot (2-2+1) + \Sigma_{i=0}^{2} 2^{i-1} \cdot (i+2) \cdot 1) = \frac{25}{15} = \frac{5}{3}$. Indeed, for the first of these 15 options, we can set $T_0 = 0$ and encode the range [0,15] by the single entry (****). For $[1,2], [2,3]$ we set $T_0 = 2$ and encode $[2,15]$ by the three entries $(001*), (01**), (1***)$. Likewise, for $[3,4], [4,5], [5,6], [6,7]$ two entries are required. For $[7,8], [8,9]$ we set $T_0 = 8$ and encode $[8,15]$ in a single range. For $[9,10], [10,11]$ two entries are required ($T_0 = 10$) while for $[11,12], [12,13], [13,14], [14,15]$ a single entry is required (for $T_0 = 12$ or $T_0 = 14$). This yields an average number of $\frac{1}{15} \cdot (1+2 \cdot 3+4 \cdot 2+2 \cdot 1+2 \cdot 2+4 \cdot 1) = \frac{25}{15} = \frac{5}{3} = \rho(W, TOL)$, as suggested by Theorem 5.*

C. Installation Bounds and Periodic Ranges

As noted in Section II-B, setting up a timed installation rule requires a two-step procedure (see Fig. 5); in the *insertion* step, the TCAM management software installs the timestamp-dependent TCAM rule representing the configuration that should take place starting at time T_0. In the *cleanup* step, the management software removes the timestamp dependency of the rules representing the new configuration, leaving a single time-oblivious TCAM entry. In this section we assume that the insertion and cleanup operations are performed within well-known *installation bounds*,[3] denoted by Δ, i.e., it is guaranteed that the time-based rule is inserted no sooner than Δ before T_0, and is cleaned up by time $T_0 + \Delta - 1$.

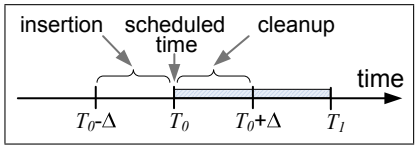


Fig. 5: Installation bounds

We shall show that guaranteed installation bounds significantly reduce the number of TCAM entries required for a TIMEFLIP; rather than defining a range $T \geq T_0$, one can define the range $[T_0, T_1]$ for some $T_1 \geq T_0 + \Delta - 1$, using fewer TCAM entries with effectively the same impact.

Two ranges, R_1 and R_2, are said to be ***Δ-similar***, denoted $R_1 \overset{\Delta}{\sim} R_2$, if there exists a value T_0 such that $R_1 \cap R_2 \supseteq [T_0, T_0+\Delta-1]$ and $(R_1 \cup R_2) \cap [T_0-\Delta, T_0-1] = \emptyset$. Given an extremal range $R_{T_0} = [T_0, 2^W-1]$, since R_{T_0} is only observed during the period $[T_0 - \Delta, T_0 + \Delta - 1]$, every range R that is Δ-similar to R_{T_0} produces the same TCAM match results during this period. Hence, every timed installation over R_{T_0} can be represented by an equivalent timed installation over R.

We define the ***2^V-periodic continuation*** of a time range $[T_0, T_1]$, denoted by $[T_0, T_1]^{V^-}$, to be the range defined by masking the $W - V$ most significant bits of the timestamp, i.e., $R_{pc} := \bigcup_{n=0}^{2^{W-V}-1} ([T_0^{V^-}, T_1^{V^-}] + n \cdot 2^V)$. Moreover, if $T_1^{V^-} < T_0^{V^-}$, then $R_{pc} = \bigcup_{n=0}^{2^{W-V}-1} (([T_0^{V^-}, 2^V-1] \cup [0, T_1^{V^-}]) + n \cdot 2^V)$. The expansion of a periodic range R_{pc} is the number of entries used for representing the range.

Intuitively, periodic ranges (Fig. 6) allow efficient representation of TIMEFLIPs. A 2^V-periodic range is encoded with *don't care* in its $W - V$ most significant bits, and thus the number of bits required to represent such a range is V.

We now introduce the BOUNDEDRANGE algorithm. Given a scheduling time, T_0, and an extremal range, $R_{T_0} = [T_0, 2^W - 1]$, the algorithm computes a periodic range $R_{BR} \overset{\Delta}{\sim} R_{T_0}$ that, for a sufficiently small Δ, has a smaller expansion than R_{T_0}.

Lemma 6. *If $R_{BR} = \text{BOUNDEDRANGE}(T_0, \Delta, W)$, then the expansion of every timed installation over R_{BR} is bounded by $\lceil log_2(2\Delta) \rceil$.*

Proof. We denote $\lceil log_2(2\Delta) \rceil$ by V. We analyze the periodic range $[T_0, T_0 + 2^{V-1} - 1]^{V^-}$, focusing on a single range of 2^V values, and distinguish between two cases:

(i) $T_0^{(W-V)^+} = (T_0 + 2^{V-1} - 1)^{(W-V)^+}$: in this case (depicted in Fig. 6(i)) we have a shifted V-bit range, $[T_0^{V^-}, (T_0 + 2^{V-1} - 1)^{V^-}]$. We shall show that this range has a worst-case expansion of V. We analyze the two sub-ranges $[T_0^{V^-}, 2^{V-1} - 1]$ and $[2^{V-1}, (T_0 + 2^{V-1} - 1)^{V^-}]$. Both sub-ranges are in fact $(V-1)$-bit shifted extremal ranges. The expansions of these two sub-ranges are $r(T_0^{(V-1)^-})$ and $\ell(T_0^{(V-1)^-})$, respectively. By Lemma 1, over a $(V-1)$-bit field we have $r(T) + \ell(T) \leq V$ for all T. It follows that the

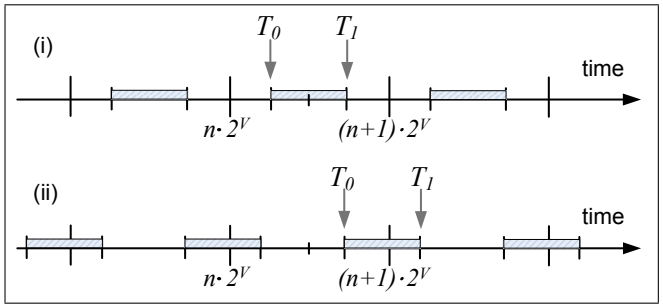


Fig. 6: Periodic ranges: the 2^V-periodic continuation of $[T_0, T_1]$. (i) For $T_2^{V^-} > T_1^{V^-}$. (ii) For $T_2^{V^-} < T_1^{V^-}$. In BOUNDEDRANGE $T_1 = T_0 + 2^{V-1} - 1$.

[3] In practice, the installation bounds may be high in some network devices. We further discuss how this affects TIMEFLIP in Section VI-B

BOUNDEDRANGE(T_0, Δ, W)
1 $V \leftarrow \lceil log_2(2\Delta) \rceil$
2 return $[T_0, T_0 + 2^{V-1} - 1]^{V^-}$

Fig. 7: Determining a range with installation bounds Δ.

expansion of the two sub-ranges is V.

(ii) $T_0^{(W-V)^+} \neq (T_0 + 2^{V-1} - 1)^{(W-V)^+}$: in this case (depicted in Fig. 6(ii)), by definition of a 2^V-periodic continuation we have a shifted V-bit range $[T_0^{V^-}, 2^V - 1] \cup [0, (T_0 + 2^{V-1} - 1)^{V^-}]$. As in case (i), we have two $(V-1)$-bit complementary extremal ranges, and thus the worst-case expansion of the two sub-ranges is V. □

The following theorem states that when the scheduling tolerance, TOL, is sufficiently large, a timed installation can be represented by a single TCAM entry.

Theorem 7. *If $TOL \geq 2^{\lceil log_2(\Delta) \rceil}$, then there exists a range R such that $R \overset{\Delta}{\sim} R_{\text{SCH}}$, and the expansion of every timed installation over R is 1.*

Proof. Since $TOL \geq 2^{V-1}$, it follows that $\lfloor log_2(TOL) \rfloor \geq 2^{V-1}$. There exists an integer n such that $T_{\text{SCH}} = n \cdot X$, since by Lemma 3 the X least significant bits of T_{SCH} are '0'. We define $R_{BR} := \text{BOUNDEDRANGE}(T_{\text{SCH}}, \Delta, W)$. By definition of BOUNDEDRANGE, $R_{BR} \overset{\Delta}{\sim} R_{\text{SCH}}$. Thus, at least one of the following must hold:

- There exists an integer n such that $T_{\text{SCH}} = n \cdot 2^V$. Using BOUNDEDRANGE we have $R_{BR} = [0, 2^{V-1}]^{V^-}$, which can be encoded by a single entry where the timestamp field has the value $*^{W-V}, 0, *^{V-1}$, i.e., the only unmasked bit is $t_V = 0$.
- There exists an integer n such that $T_{\text{SCH}} = (2n+1) \cdot 2^{V-1}$. Thus, ${T_{\text{SCH}}}^{V^-} = 2^{V-1}$. By using BOUNDEDRANGE we have $R_{BR} = [2^{V-1}, 2^V - 1]^{V^-}$, which can be encoded by the single entry $*^{W-V}, 0, *^{V-1}$, i.e., the only unmasked bit is $t_V = 1$.

□

The following theorem generalizes the observations about the scheduling tolerance and installation bounds, and provides the worst-case expansion as a function of TOL and Δ.

Theorem 8. *If $R_{BR} = \text{BOUNDEDRANGE}(T_{\text{SCH}}, \Delta, W)$, and $TOL < 2^{\lceil log_2(\Delta) \rceil}$, then the expansion of every timed installation over R_{BR} is bounded by $\lceil log_2(2\Delta) \rceil - \lfloor log_2(TOL) \rfloor$.*

Proof Outline. The proof is based on Lemma 4 and Lemma 6. □

D. Timestamp Field Size in Bits

In the analysis so far we have been assuming that the timestamp field is a W-bit field. This implies that in every TCAM that requires timed installations, W bits of every entry would be "wasted" on the timestamp field. In this section we analyze how the timestamp field can be significantly reduced, depending on the scheduling tolerance and installation bounds. We show that the size of the timestamp field (Fig. 8) is affected by two factors of the system: (i) If it is well-known that every TIMEFLIP is scheduled with a scheduling tolerance TOL, then the $X = \lfloor log_2(TOL) \rfloor$ least significant bits of the timestamp field are always *don't care*, and thus can be omitted from the timestamp field. (ii) If there are guaranteed installation bounds, Δ, the use of a 2^V-periodic range, for $V = \lceil log_2(2\Delta) \rceil$, allows the $W - V$ most significant bits to be omitted.

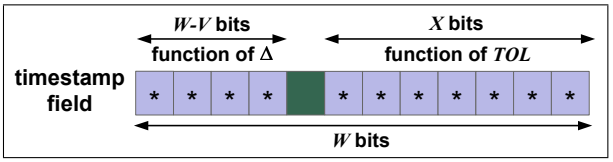


Fig. 8: Example of 1-bit timestamp, per Theorem 10.

Lemma 9. *If $TOL < 2^W$, then the range R_{SCH} can be represented by a timestamp field of $W - \lfloor log_2(TOL) \rfloor$ bits.*

Proof. We denote $\lfloor log_2(TOL) \rfloor$ by X. By Lemma 3 the X least significant bits of T_{SCH} are '0'. Thus, every prefix-based encoding of R_{SCH} has *don't care* on the X least significant bits. Hence, R_{SCH} can be represented by the $W - X$ most significant bits. □

Theorem 10. *If $TOL \geq 2^{\lceil log_2(\Delta) \rceil}$, then there exists a range R such that $R \overset{\Delta}{\sim} R_{\text{SCH}}$, and R can be represented by a timestamp field of a single bit.*

Proof. The proof is very similar to the proof of Theorem 7; the range R_{BR} satisfies $R_{BR} \overset{\Delta}{\sim} R_{\text{SCH}}$, and can be represented by a single rule, where the only unmasked bit is t_V, for $V = \lceil log_2(2\Delta) \rceil$. □

Theorem 11. *If $TOL < 2^W$, the installation bounds are given by Δ, and $TOL < 2^{\lceil log_2(\Delta) \rceil}$, then there exists a range R such that $R \overset{\Delta}{\sim} R_{\text{SCH}}$, and R can be represented by a timestamp field of $\lceil log_2(2\Delta) \rceil - \lfloor log_2(TOL) \rfloor$ bits.*

Proof. The range R_{BR} satisfies $R_{BR} \overset{\Delta}{\sim} R_{\text{SCH}}$. By definition of BOUNDEDRANGE, the most significant $W - V$ bits are masked, and can thus be omitted. By Lemma 9 the X least significant bits are masked and can thus be omitted. Hence, we are left with $V - X = \lceil log_2(2\Delta) \rceil - \lfloor log_2(TOL) \rfloor$ bits. □

It is well-known [14] that a W-bit extremal range has a worst-case expansion W, i.e., there is a tight coupling between the expansion of an extremal range and the number of bits used to represent it. Thus, it is not surprising that our results show that this coupling applies to time-based ranges as well, as seen in Theorems 8 and 11. Specifically, in a system that uses a 1-bit timestamp, per Theorem 10, every TIMEFLIP is represented by a single entry, as shown in Theorem 7.

IV. Optimal Time-based Action Updates

In the previous section we analyzed timed **installations** (Fig. 3(ii)). In this section we briefly discuss these results in the context of timed **action updates** (Fig. 3(v)), and show that the number of entries required to represent a time-based action update is, in the worst case, roughly half of the number of entries required to represent a timed installation.

Significantly, timed action updates can be represented either by positive encoding (Fig. 3(v)) or by negative encoding (Fig. 3(vi)). It was shown [23] that a W-bit extremal range can be represented by $\lceil \frac{W+1}{2} \rceil$ entries, by choosing the best of the positive or the negative encoding.

Let R be a time range, and define $R^c := [0, 2^W - 1] \setminus R$ to be the complementary range of R. Given a time-oblivious TCAM entry $S \rightarrow a$ with $S = (s_u, \ldots, s_1, *, \ldots, *)$, we define a ***timed action update*** of S over R as a pair of TCAM rules $(S_R \rightarrow a_R, S \rightarrow a)$, such that $S_R := (s_u, \ldots, s_1, R)$. Hence, a_R is activated during the time range R. Note that the order of the rules is of importance, since a TCAM lookup can match S only if it does not match S_R. Given a timed action update, $(S_R \rightarrow a_R, S \rightarrow a)$, we define its negative encoding as $(S_{R^c} \rightarrow a, S \rightarrow a_R)$, such that $S_{R^c} := (s_u, \ldots, s_1, R^c)$.

We define the *expansion* of a timed action update over a time range R, denoted by $e(R)$, as the expansion of R. In this context $e(R)$ is the minimum between the positive encoding of R and its negative encoding, given by R^c. Note that in both cases, positive and negative, the expansion does not include the time-oblivious entry, S. The following theorem defines an upper bound on the expansion of a timed action update over a W-bit extremal range.

Theorem 12. *If $e(R_{T_0,W})$ is the expansion of a timed action update over $R_{T_0,W} = [T_0, 2^W - 1]$, then $e(R_{T_0,W}) \leq \lfloor \frac{W+1}{2} \rfloor$.*

Proof Outline. The proof is based on the $\lceil \frac{W+1}{2} \rceil$ result of [23], with the exception that, in contrast to [23], our definition of timed action updates excludes the entry that assigns *don't care* to the timestamp field. Due to this minor difference, the expansion is $\lfloor \frac{W+1}{2} \rfloor$ rather than $\lceil \frac{W+1}{2} \rceil$. □

The following lemma presents the worst-case expansion of using both positive and negative encoding, as a function of the scheduling tolerance. The result generalizes Theorem 12.

Lemma 13. *If $e(R_{\text{SCH}})$ is the expansion of R_{SCH}, and $TOL < 2^W$, then $e(R_{\text{SCH}}) \leq \lfloor \frac{W - \lfloor log_2(TOL) \rfloor + 1}{2} \rfloor$.*

Proof Outline. The proof is similar to the proof of 4, but uses the result of Theorem 12 for the worst-case expansion when using both positive and negative encoding. □

V. Experimental Evaluation

Our evaluation is composed of two parts:

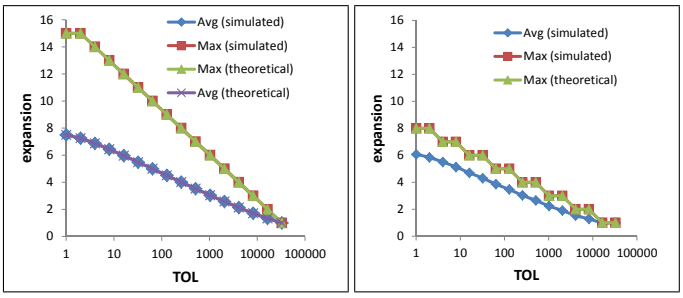
(i) A simulation-based analysis was used to evaluate the resources required for representing TimeFlips, and to verify our analytical results from the previous sections.

(ii) A microbenchmark using a commercial switch was used to evaluate the accuracy of timed updates using our approach.

A. Simulation-based Evaluation

We implemented the Schedule and BoundedRange algorithms, and computed the respective range expansion and the required timestamp bit size in various cases. All of our simulations were performed with $W = 16$.

We evaluated the expansion of an extremal range as a function of the scheduling tolerance, TOL. For each value of TOL we simulated all the possible values of T_{min}, and the graphs in Fig. 9 present both the worst-case expansion and the average expansion (as defined in Section III-B). Fig. 9a depicts the results for timed installation, i.e., $r(T_0)$, while Fig. 9b illustrates the results for timed *action updates*. It can be shown that the expansion of the latter is roughly half of the former, since timed action updates make use of both the positive and the negative encoding.



(a) Timed installation: expansion as a function of TOL. Theoretical: max based on Lemma 4, average based on Theorem 5.

(b) Timed action update: expansion as a function of TOL using both positive and negative encoding. Theoretical max based on Lemma 13.

Fig. 9: Expansion as a function of TOL

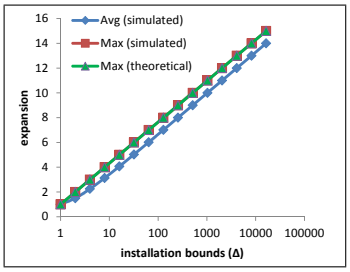
Fig. 10 depicts the effect of the installation bounds, Δ, on the time range expansion. Schedule was used for computing T_0, and BoundedRange was used for selecting the time range. Fig. 10a illustrates the expansion for $TOL = 1$, and includes both the simulated values and the analytical values, based on Lemma 6. Fig. 10b depicts the worst-case expansion for several values of TOL. The star-shaped markers indicate the points where $TOL = 2^{\lceil log_2(\Delta) \rceil}$, illustrating that, as stated in Theorem 7, if Δ is small enough, i.e., $TOL \geq 2^{\lceil log_2(\Delta) \rceil}$, the time range can be represented by a single entry.

Fig. 11 illustrates the effect of the scheduling tolerance and the installation bounds on the number of bits required to represent the timestamp field. Again, the star-shaped markers indicate the points where $TOL = 2^{\lceil log_2(\Delta) \rceil}$, and thus by Theorem 10, if Δ has a smaller value than the star-shaped marker, the timestamp field requires only a single bit.

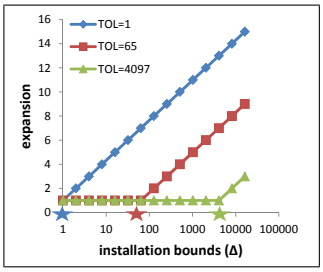
The simulations confirm our theoretical results, and demonstrate the tradeoff between the two parameters, TOL and Δ, and the TCAM resource consumption.

B. Microbenchmark

We performed an experiment in order to demonstrate that the method presented in this paper is applicable to real-life switches, and that the method can effectively provide a high



(a) Expansion as a function of Δ for $T_{max} = T_{min}$. Theoretical values based on Lemma 6.



(b) The simulated worst-case expansion as a function of Δ for various values of TOL. The star-shaped markers indicate the points where $TOL = 2^{\lceil log_2(\Delta) \rceil}$.

Fig. 10: Expansion as a function of Δ with BOUNDEDRANGE in a timed installation.

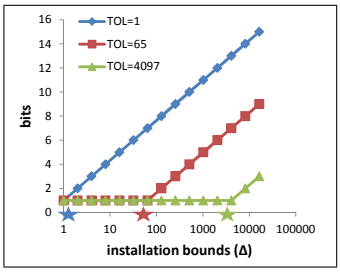
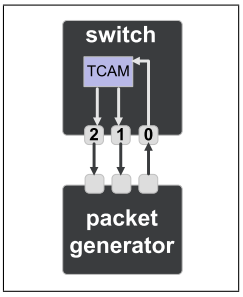


Fig. 11: The number of bits as a function of Δ for various values of TOL, using BOUNDEDRANGE in a timed installation. The star-shaped markers indicate the points where $TOL = 2^{\lceil log_2(\Delta) \rceil}$.

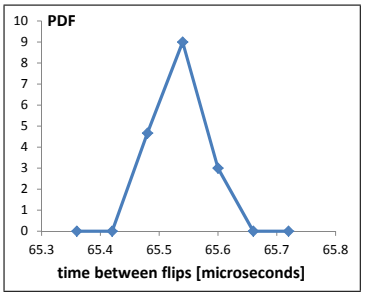
degree of accuracy. As mentioned above, when an update is scheduled for time T_0, it is performed in practice at some time $t \in [T_0 - \delta, T_0 + \delta]$. The scheduling accuracy, δ, is affected by two factors, the device's *clock accuracy*, which is the maximal offset between the clock value and the value of an accurate time reference, and the *execution accuracy*, which is a measure of how accurately the device can perform a timed update, given a clock that is perfectly synchronized to real time. The achievable clock accuracy strongly depends on the network size and topology, and on the clock synchronization method being used. For example, the achievable accuracy using the Precision Time Protocol [9] is typically on the order of 1 microsecond [10], [11]. Our microbenchmark is focused on the *execution accuracy* of time-based TCAM updates.

The experiment was performed using an evaluation board of the Marvell 98DX4251 [33] switch silicon. The reason we used a switch silicon evaluation board is that it provides flexible configuration options compared to an off-the-shelf pizza-box switch. Specifically, the evaluation board allows the flexibility to define the structure of the TCAM key, including an ingress timestamp field. It is important to emphasize that we used the switch as-is, without modifications or extensions.

The experiment setup is illustrated in Figure 12a. We used an IXIA XM12 packet generator, that was connected to ports



(a) Experiment setup



(b) Empirical PDF of the time between TIMEFLIPS

Fig. 12: Microbenchmark

0, 1, and 2 of the switch, and was configured to continuously transmit 64B-packets to port 0 of the switch at a full-wire-speed of 10 Gbps. Thus, a packet was transmitted to the switch every 67.2 ns (nanoseconds). The switch was configured to perform a TCAM lookup on all incoming packets, with the following two entries:

- $(in\_port = 0, T = (*\ldots*, 1, *^{15})) \rightarrow out\_port = 1$
- $(in\_port = 0, T = (*\ldots*)) \rightarrow out\_port = 2$

The only unmasked bit in the timestamp field of the first entry was $t_{16} = 1$. T is measured in nanoseconds, and therefore the 16th bit, t_{16}, represents 2^{16} ns. Consequently, the two rules produce periodic behavior where each rule is matched for a duration of 2^{15} ns; the first rule is matched for a duration of 2^{15} ns, and then the second rule is matched for 2^{15} ns, and so on.

In the context of this experiment, every TIMEFLIP between out_port=1 and out_port=2 is a timed action update. Our analysis focuses on the question how accurately the timed action updates occur. To answer this question, we measured the time between two consecutive TIMEFLIPS from out_port=1 to out_port=2. We repeated this measurement 50 times, and the empirical Probability Density Function (PDF) of these measurements is illustrated in Figure 12b.

The expected mean time interval between TIMEFLIPS was 2^{16} = 65536 ns. The precision of our measurements was affected by two factors: (i) the packet generator timestamped the incoming packets with a 10 ns resolution, and (ii) the packet generator transmitted a packet exactly every 67.2 ns. Due to these two factors, the precision of the measurement was on the order of tens of nanoseconds. Notably, since the measurement is performed by the packet generator, as a difference between two TIMEFLIP events, no synchronization is required between the packet generator and the switch.

As shown in Figure 12b, the timed action updates were all performed within tens of nanoseconds of the expected time, which is well within the margin of error of our measurement method. Hence, the *execution accuracy* in our experiment was no worse than tens of nanoseconds, which is negligible compared to the *clock accuracy* in a typical network, on the order of 1 microsecond. Thus, the microbenchmark indicates that using the method we present in this paper, updates can

be timed in a typical network with a microsecond accuracy.

VI. Discussion

A. The Scheduling Tradeoff

TimeFlip allows accurate scheduling while allowing efficient TCAM resource consumption. Notably, the execution accuracy of TimeFlips is not affected by the scheduling tolerance, TOL, and the installation bound, Δ, whereas the resource consumption, namely the number of bits per timestamp and the number of entries per TimeFlip is indeed affected by these two parameters.

Consider a scenario where n different updates need to be performed, all having the same constraint, $[T_{min}, T_{max}]$. The Schedule algorithm computes the same T_{SCH} for the n updates, resulting in two potential drawbacks: (i) If a large number of updates must be installed and removed in a short period of time around T_{SCH}, this load on the TCAM management software may affect the installation bound Δ. (ii) The total number of excess TCAM entries that are used for the n updates is on the order of n. To avoid this collision scenario, the scheduling constraints can be defined as n disjoint ranges, $[T^j_{min}, T^j_{max}]$ for $1 \leq j \leq n$, producing n different scheduling times T_j, thereby avoiding the two issues above. Intermediate approaches can be taken, where the n ranges partially overlap. Hence, TOL should be selected carefully; on the one hand, a low value of TOL yields higher TCAM resources per update (Theorem 8). On the other hand, if TOL is too large, the collision scenario above may occur. This tradeoff regarding the scheduling tolerance, TOL, shows that it should be carefully selected based on the system properties, Δ and n.

B. The Cost of High Installation Bounds

As discussed in Section III-C, guaranteed installation bounds, Δ, can help reduce the amount of TCAM resources for each TimeFlip. However, previous work [6] has demonstrated large fluctuations in TCAM rule installation latencies, varying from a few milliseconds to a few seconds. Does this make TimeFlip impractical? Fortunately, the answer is no.

High installation bounds may yield high resource consumption by each TimeFlip, but does not compromise their execution accuracy. A network device can reduce Δ by limiting the number of TCAM updates per second.

Interestingly, as shown in Theorems 8 and 11, even when Δ is very high, if TOL is sufficiently large, then TimeFlips can still be represented very efficiently. For example, if Δ is as high as 10 seconds, and TOL is also 10 seconds, then every TimeFlip can still be represented by a single TCAM entry, using a single timestamp bit. Hence, TimeFlip can perform well even in the presence of high installation bounds.

C. Timed Updates of Non-TCAM Memories

The concepts presented in this paper can be used for applying timed updates to non-TCAM lookup tables in network devices. We provide an example of performing a timed update in an IP routing table. Assume that at time T_0 a set of entries in the routing table should be updated to a new value. As shown in Fig. 13, a time-based TCAM range is used for defining the time range $T \geq T_0$, and the corresponding action is a *version* metadata field, indicating whether routing should be performed based on the old version or on the new one. The version value is then used to access the routing table, along with the destination IP address. This approach bears some resemblance to the version tag approach of [17], although our approach uses the version indication internally in the network device, and it is not added to the packet header as in [17].

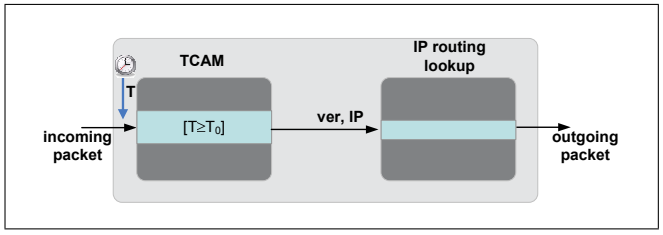


Fig. 13: Timed updates in non-TCAM lookups

VII. Conclusion

We introduced TimeFlip, a practical method of implementing accurate time-based network updates and a natural implementation of Atomic Bundles, using time-based TCAM ranges. At the heart of our analysis lie two properties that are unique to time-based TCAM ranges. First, by carefully choosing the scheduled update time, the range values can be selected to minimize the required TCAM resources. Second, if there is a known bound on the installation time of the TCAM entries, then by using periodic time ranges, the expansion of the time range can be significantly reduced. We have shown that TimeFlips work on existing network devices, making accurate time-based updates a viable tool for network management.

References

[1] T. Mizrahi, O. Rottenstreich, and Y. Moses, "TimeFlip: Scheduling network updates with timestamp-based TCAM ranges," in *IEEE INFOCOM, accepted*, 2015.

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. M. Parulkar, L. L. Peterson, J. Rexford, S. Shenker, and J. S. Turner, "Openflow: enabling innovation in campus networks," *ACM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[3] Open Networking Foundation, "Openflow switch specification," *Version 1.4.0*, 2013.

[4] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: scaling flow management for high-performance networks," in *ACM SIGCOMM*, 2011.

[5] J. Naous, D. Erickson, G. A. Covington, G. Appenzeller, and N. McKeown, "Implementing an OpenFlow switch on the NetFPGA platform," in *ACM/IEEE ANCS*, 2008.

[6] X. Jin, H. H. Liu, R. Gandhi, S. Kandula, R. Mahajan, J. Rexford, R. Wattenhofer, and M. Zhang, "Dionysus: Dynamic scheduling of network updates," in *ACM SIGCOMM*, 2014.

[7] T. Mizrahi and Y. Moses, "Time-based updates in software defined networks," in *ACM HotSDN*, 2013.

[8] ——, "Time-based Updates in OpenFlow: A Proposed Extension to the OpenFlow Protocol," technical report, CCIT Report #835, EE Pub No. 1792, 2013. [Online]. Available: http://tx.technion.ac.il/~dew/OFTimeTR.pdf

[9] IEEE TC 9, "1588 IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems Version 2," 2008.

[10] ITU-T G.8271/Y.1366, "Time and phase synchronization aspects of packet networks," 2012.

[11] IEEE Std C37.238, "IEEE Standard Profile for Use of IEEE 1588 Precision Time Protocol in Power System Applications," 2011.

[12] F. Long, Z. Sun, Z. Zhang, H. Chen, and L. Liao, "Research on TCAM-based OpenFlow switch platform," in *IEEE ICSAI*, 2012.

[13] Renesas, "R8A20410BG QUAD-Search TCAM," datasheet, 2010.

[14] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel, "Fast and scalable layer four switching," in *ACM SIGCOMM*, 1998.

[15] A. Bremler-Barr and D. Hendler, "Space-efficient TCAM-based classification using gray coding," *IEEE Trans. Computers*, vol. 61, no. 1, pp. 18–30, 2012.

[16] P. François and O. Bonaventure, "Avoiding transient loops during the convergence of link-state routing protocols," *IEEE/ACM Trans. Netw.*, vol. 15, no. 6, pp. 1280–1292, 2007.

[17] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, "Abstractions for network update," in *ACM SIGCOMM*, 2012.

[18] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven WAN," in *ACM SIGCOMM*, 2013.

[19] J. C. Corbett, J. Dean, M. Epstein *et al.*, "Spanner: Google's globally-distributed database," in *OSDI*, 2012.

[20] A. G. Greenberg, G. Hjálmtýsson, D. A. Maltz, A. Myers, J. Rexford, G. G. Xie, H. Yan, J. Zhan, and H. Zhang, "A clean slate 4D approach to network control and management," *ACM Computer Communication Review*, vol. 35, no. 5, pp. 41–54, 2005.

[21] A. Atlas, T. Nadeau, and D. Ward, "Interface to the routing system problem statement," IETF, draft-atlas-i2rs-problem-statement-01, work in progress, 2013.

[22] "Minutes for IETF87, SDNRG meeting," IETF, meeting minutes, 2013.

[23] O. Rottenstreich, R. Cohen, D. Raz, and I. Keslassy, "Exact worst case TCAM rule expansion," *IEEE Trans. Computers*, vol. 62, no. 6, pp. 1127–1140, 2013.

[24] H. Liu, "Efficient mapping of range classifier into ternary-cam," in *IEEE Hot Interconnects*, 2002.

[25] H. Che, Z. Wang, K. Zheng, and B. Liu, "DRES: Dynamic range encoding scheme for TCAM coprocessors," *IEEE Trans. Computers*, vol. 57, no. 7, pp. 902–915, 2008.

[26] C. R. Meiners, A. X. Liu, and E. Torng, "TCAM Razor: A systematic approach towards minimizing packet classifiers in TCAMs," in *IEEE ICNP*, 2007.

[27] A. Bremler-Barr, D. Hay, and D. Hendler, "Layered interval codes for TCAM-based classification," *Computer Networks*, vol. 56, no. 13, pp. 3023–3039, 2012.

[28] K. Kogan, S. I. Nikolenko, O. Rottenstreich, W. Culhane, and P. Eugster, "SAX-PAC (Scalable And eXpressive PAcket Classification)," in *ACM SIGCOMM*, 2014.

[29] E. Norige, A. X. Liu, and E. Torng, "A ternary unification framework for optimizing TCAM-based packet classification systems," in *ACM/IEEE ANCS*, 2013.

[30] C. R. Meiners, A. X. Liu, E. Torng, and J. Patel, "Split: Optimizing space, power, and throughput for TCAM-based classification," in *ACM/IEEE ANCS*, 2011.

[31] C. R. Meiners, A. X. Liu, and E. Torng, "Bit Weaving: A non-prefix approach to compressing packet classifiers in TCAMs," *IEEE/ACM Trans. Networking*, vol. 20, no. 2, pp. 488–500, 2012.

[32] O. Rottenstreich, I. Keslassy, A. Hassidim, H. Kaplan, and E. Porat, "On finding an optimal TCAM encoding scheme for packet classification," in *IEEE INFOCOM*, 2013.

[33] "Marvell Prestera 98DX4251 Product Brief," http://www.marvell.com/switching/assets/Marvell_Prestera_98DX4251-02_product_brief_final2.pdf, 2013.