IFIP Networking 2019 1570511846

Balancing Performance Characteristics of Hierarchical Heavy Hitters

Hauke Heseding, Johannes Glöckle*, Robert Bauer, Martina Zitterbart
Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
{hauke.heseding, robert.bauer, zitterbart}@kit.edu, johannes-gloeckle@t-online.de

***Abstract*—Hierarchical Heavy Hitters (HHHs) identify frequent items in streaming data. Finding these items has several applications to network monitoring, particularly in distributed denial-of-service (DDoS) mitigation and anomaly detection. Several algorithms are available to compute HHHs, each with different performance characteristics in terms of resource consumption, speed and accuracy. These characteristics determine which HHH algorithm may be best suited for a given network situation (e.g., because it offers sufficient accuracy for fine-grained traffic analysis). However, since the situation can evolve over time, the best choice for an HHH algorithm may also change. Simply replacing a chosen HHH algorithm has the drawback of losing all previously acquired monitoring information.**

This paper introduces the novel concept of HHH-transitions that transfer monitoring information between HHH variants and consequently allows it to adopt new performance characteristics by switching algorithms at runtime. For example, this enables a DDoS mitigation system to adapt to evolving network situations and therefore increase overall Return-on-Mitigation. We present explicit transition rules for common one-dimensional HHH variants and evaluate our approach based on real traffic from MAWILab. Results indicate that trade-offs between performance characteristics can be realized at runtime and that it is possible to increase overall post-transition accuracy by retaining monitoring information.

***Index Terms*—HHH, monitoring, DDoS defense, HHH-transitions**

I. Introduction

Efficient processing of large traffic volumes is a key challenge in network monitoring, specifically in the field of DDoS mitigation [1], [2]. To simplify this task, data streams are often preprocessed to aggregate relevant characteristics into a compact but faithful representation. One way to obtain such a representation is through HHH algorithms, which exploit the structure of hierarchically organized items (e.g., IP addresses) to calculate their most frequently occurring manifestations in streaming data. This information is of particular interest in DDoS mitigation, since it can be used to identify malicious traffic sources and to generate appropriate perimeter blacklists. The effectiveness of such an approach critically depends on the choice of HHH algorithms, since their performance characteristics differ significantly with respect to speed, accuracy and overall resource consumption. A mitigation system should balance these key factors in accordance with the actual demand necessitated by an ongoing attack instead of over-emphasizing any single one of them.

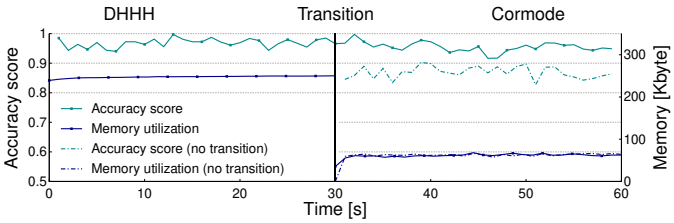

Fig. 1. Average memory consumption and accuracy score before and after an HHH-transition from the DHHH algorithm to the Cormode algorithm.

While a certain HHH algorithm may be best suited for a specific network situation, it can constitute a suboptimal choice if the situation changes over time. Several conditions may render the choice of an HHH algorithm obsolete. The memory consumption of certain HHH algorithms is proportional to the amount of different items in a data stream, such that available memory may be exhausted when items become more distinct over time. Other algorithms require only a fixed amount of memory, however, this can limit their maximum accuracy as well as their query and update times. These algorithms can prove insufficient when a demand for fine-grained traffic analysis arises or traffic volume increases. Furthermore, DDoS mitigation is often performed in a cloud environment to scale resources in accordance with attack intensity. When the mitigation uses an HHH algorithm for traffic analysis, the only option may be to frequently terminate and restart HHH instances to allow them to scale their resources. These conditions may occur unpredictably, hence, it may not be possible to select the best-suited algorithm a-priori.

In these cases, it would be preferable to switch over to a different HHH variant with more suitable performance characteristics. However, simply replacing the initially chosen algorithm results in the complete loss of all previously acquired monitoring information and a new instance would have to start ”from scratch”. It would require some time to reach a stable state again before HHH computation can resume with sufficient accuracy. Moreover, information on long-term behavior of traffic could no longer be retained. However, such information may be valuable in certain applications. For example, long-term monitoring information on benign traffic allows to build whitelists in the case of DDoS mitigation.

*Portions of this research were done while the author was a student at KIT.

ISBN 978-3-903176-16-4

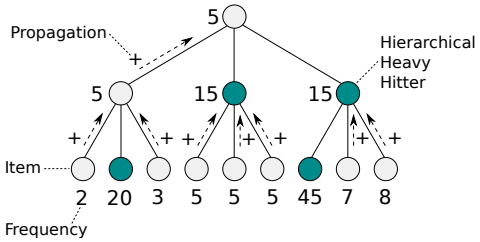

Fig. 2. Frequency propagation during calculation of HHHs for threshold parameter $\phi \cdot N = 10$ in a ternary prefix trie.

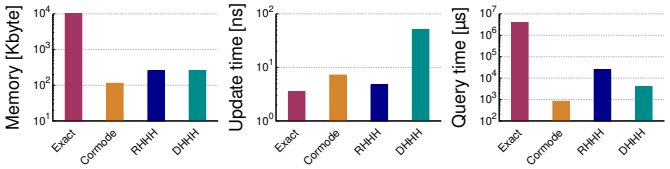

Fig. 3. Resource consumption w.r.t. maximum query/update time and memory utilization of Exact, Cormode, RHHH and DHHH measured over 30 different network traces with 60-second duration.

To address the problem of information loss, we define a set of so-called HHH-transitions that transfer the internal state of an HHH algorithm instance into that of a target variant, effectively retaining most of the already available monitoring information. We illustrate the effect with a transition from the (one-dimensional) sketch-based deterministic HHH algorithm (DHHH) to the Cormode-variant [3], [4] while processing 30 network traces of 60-second duration as described in Sec. IV-B using identical parameters for both algorithms. Fig. 1 shows the average accuracy of computed HHHs and memory consumption when starting with DHHH (left) and transitioning to Cormode (right) after 30 seconds. As a result, the typically lower memory footprint of the Cormode algorithm can be adopted. Transferring the already available monitoring information from DHHH to Cormode with a transition offers the additional benefit of increased accuracy. For comparison, the dotted green line in the right part shows the accuracy when Cormode is started at the 30-second mark without retaining the monitoring information.

The previous example shows, that a monitoring system can adapt its performance characteristics at runtime in terms of memory consumption and accuracy. With an additional transition in opposite direction, i.e., from DHHH to Cormode, memory consumption and accuracy become virtually interchangeable. To allow further trade-offs, we define transitions to and from the two previous algorithms, as well as the randomized HHH (RHHH) variant and the exact HHH algorithm (Exact). Nine transitions can be performed directly between two HHH algorithms, while the remaining three transitions use an intermediate algorithm. Despite this indirection, the memory overhead of all transitions is negligible, since entries in an algorithms data structure are immediately deleted after the transfer to another HHH instance.

To summarize, the contributions provided by HHH-transitions are as follows:

- The capability to balance resource-utilization, accuracy and speed of HHH algorithms as necessitated by evolving network behavior.
- The capability to retain acquired monitoring information when a change of HHH algorithms is desirable.
- Negligible increase in memory footprint while transferring information between HHH algorithm instances.

II. Background

Heavy Hitters (HHs) are items in a data stream whose frequency, i.e., the number of occurrences, exceeds a certain fraction ϕ of the total count of data stream items N. HH algorithms can identify the most frequently occurring items during online processing of streaming data. HHHs build upon this concept to provide a more accurate description of the distribution of items whose values are taken from a hierarchically structured domain. For each layer in the hierarchy of that domain the frequencies of items are aggregated into that of a higher-level item, i.e. a prefix, if their combined sum does not yield an HH in itself. Through repeated aggregation the distribution of data steam items can be represented as a hierarchically organized data structure as illustrated in Fig. 2.

The Exact algorithm computes HHHs in an error-free way by following the inductive definition of HHHs (cf. [3])

$$HHH_l := HHH_{l-1} + \{p : |p| = l \wedge F_p \geq \phi \cdot N\}, \qquad (1)$$

where $|p|$ is the length of prefix p, HHH_0 denotes the set of HHs derived from prefixes with maximum length, i.e., actual data stream items, and F_p denotes the accumulated frequencies of all items generalizable to p that are not captured by an HHH of lower level. When the maximum length of a prefix is L, HHH_L constitutes the (final) set of HHHs.

While the Exact algorithm stores item frequencies in separate entries, the Cormode algorithm [4] uses a trie data structure and repeatedly prunes prefixes with frequencies below a predefined threshold ϵN ($\epsilon \in (0, 1]$). This introduces an error in the frequency estimation of data stream items. To avoid a computation of inaccurate HHHs, the maximum incurred error Δ_p for a given prefix p is recorded during the pruning process. Through this, Cormode guarantees that the inequality

$$|f_p - f_p^\star| \leq \epsilon N \qquad (2)$$

holds for all prefixes p, where f_p and $f_p^\star$ denote the true and estimated frequencies.

The DHHH and RHHH [5] algorithms use a sketch data structure with multiple instances of HH algorithms to track prefix frequencies across different levels of the hierarchy. Both algorithms guarantee the same bounds for the error in frequency estimation as the Cormode algorithm (w.r.t. parameters ϵ and N). However, eq. 2 holds only with probability $\pi \geq 1-\delta$ in the case of RHHH for some chosen value $\delta \in [0, 1]$ due to randomized updates of item frequencies in the HH instances.

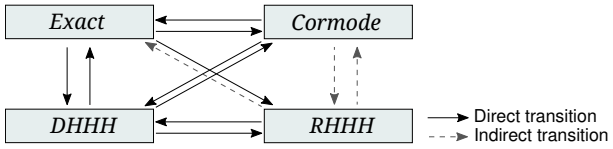

Fig. 4. Direct and indirect transitions between the four HHH algorithms Exact, DHHH, RHHH and Cormode

Throughout this paper, we consider RHHH and DHHH to be used in conjunction with the SpaceSaving HH algorithm [6], [7]. SpaceSaving provides a (fixed) number of n counters to store frequencies for individual data stream items. When the maximum number n of different counters has been exhausted and a new item p occurs in a data stream, the item q with lowest recorded frequency f_q is displaced by q. Specifically, the identifier of q is replaced with p and p inherits the frequency f_q. SpaceSaving then stores the error estimation $\Delta_p = f_q$ for p and subsequently increments the counter value f_p. For efficiency SpaceSaving stores items with identical frequency in a common bucket. Tracking Δ_p allows the SpaceSaving algorithm to ensure bounds on the error estimation as given in eq. 2 for every item processed by the HH instance.

Fig. 3 gives an example of maximum resource consumption for the four HHH algorithms. Memory utilization, query and update time differ by an order of magnitude, depending on the choice of the HHH algorithm. This is due to different data structure sizes, update strategies and HHH computations.

III. HHH-Transitions

Depending on how network situations evolve, the performance characteristics of a chosen HHH algorithm can become unsuitable over time. For example, consider a monitoring system that utilizes the DHHH algorithm. When a volumetric DDoS attack emerges, the low speed of the update operation of DHHH can easily become a bottleneck, since a large amount of items has to be processed in a short amount of time. Under these circumstances, it would be preferable to switch to the RHHH algorithm that performs update operations in considerably less time. The accumulated monitoring information stored in the DHHH instance would be lost, however. To prevent a significant loss of information, an HHH-transition, denoted as $S \rightarrow T$, transfers the internal state from a *source instance* of an HHH algorithm S to a *target instance* of algorithm T. For this, is it necessary to transform the internal data structures of S into those of T.

In the following, we outline a set of algorithms that allow it to transition between the one-dimensional versions of the four algorithms Exact, Cormode, DHHH and RHHH. For convenience, Table I summarizes terms commonly used in the context of HHH-transitions. Nine out of the twelve possible transitions are direct transitions between two algorithms. The remaining three are constructed as a chain of transitions that uses a third algorithm as an intermediate instance I – so called indirect transitions, denoted as $S \rightarrow I \rightarrow T$. Fig. 4 gives an overview of the direct and indirect transitions between the four algorithms.

TABLE I
Notations used in the context of transitions

Term	Meaning
ϵ	Threshold parameter indicating permissible pruning of prefixes
$\|p\|$	Length of prefix p
$\langle p \rangle$	Set of prefixes generalizable to prefix p
$\mathsf{par}(p)$	Parent of prefix p (i.e., the longest prefix to which p can be generalized
$\mathsf{E}, \mathsf{C}, \mathsf{D}$	An instance of the Exact, Cormode and DHHH algorithms (resp.)
X	Any one of the instances $\mathsf{E}, \mathsf{C}, \mathsf{D}$
$\mathsf{X}.N$	The total sum of item frequencies processed by X
$\mathsf{X}.H$	The maximum length of a prefix stored in X
$\mathsf{X}^{(l)}$	The set of prefixes with length l stored in X
X_p	The frequency of prefix p stored in X w.r.t. the prefix length $\|p\|$
$\Delta_{\mathsf{X},p}$	The error estimation of instance X for prefix p w.r.t. the prefix length $\|p\|$

To maintain the accuracy of computed HHHs when performing a transition, the error estimation of item frequencies in the source instance should correspond to that of the target instance. However, when performing a transition, the error estimation cannot be improved beyond the weaker bounds imposed by either the source or target instance. For example, RHHH takes random samples when processing data streams, resulting in a potential loss of information if a transition is performed before RHHH reaches its stable state. This loss cannot be recovered when transitioning to a more accurate algorithm. Hence, the new instance created by the transition can (at best) have the same bounded error as the RHHH instance.

For efficiency, we make a slight modification to the update operations of HH and HHH algorithms. Normally, this operation processes individual items, such that item frequencies are always incremented by 1. However, at the time of transitioning, the HHH instances contain aggregated frequencies and error estimations with a potentially much larger value. Consequently, we assume the algorithms to support update operations with values larger than 1 and the ability to explicitly set the error estimation associated with a given prefix. Otherwise, the execution time of a transition would depend on the sum of all frequencies instead of the much smaller number of distinct items in a data stream.

a) DHHH $\rightarrow$ RHHH and RHHH $\rightarrow$ DHHH: The transitions between DHHH and RHHH are the most straightforward of the twelve transitions. The difference in the estimated frequencies stored in DHHH and RHHH instances are the result of the randomized selection of HH instances for updates in the RHHH algorithm. RHHH updates layers only with a certain probability π. The transitions between DHHH and RHHH can be realized by multiplying frequency and error estimation of prefixes by $1/\pi$ and π when transitioning from RHHH and DHHH. This restores the expected values in the corresponding target instances. Algorithm 1 outlines the transition from DHHH to RHHH. It makes use of the modified update operation (line 5), which allows it to explicitly set the error estimation of a prefix in a target SpaceSaving instance.

Algorithm 1 HHH-transition from DHHH to RHHH

▷ $\pi \in [0,1]$ denotes the probability of invoking the update operation of any single HH instance of the RHHH algorithm

```
1: procedure DHHH_TO_RHHH(D, π)
2:     R ← new RHHH(D.ϵ, s)          ▷ Same ϵ for R and D
3:     for l = D.H downto 1 do
4:         for p ∈ D^(l) do
5:             R^(l).update(p, π · D_p, π · Δ_{D,p})
6:         end for
7:     end for
8:     return R
9: end procedure
```

Using this operation ensures that prefixes are linked to the correct bucket, whereas direct multiplication of bucket counters would lead to multiple buckets with the same associated frequencies due to rounding errors. Note, that none of the SpaceSaving algorithms in the RHHH instance will prune any transferred values, since they provide enough counters for all elements at the same layer of the DHHH instance.

The transition in opposite direction can be realized by multiplying with $1/\pi$ instead of π.

b) Exact → Cormode: The transition from an Exact instance to a new instance of Cormode is outlined in Algorithm 2. The Exact algorithm tracks the frequency of data stream items without error in a separate counter for each individual item. The transition from Exact to Cormode transfers these counters directly to the Cormode instance by initializing the frequency of the corresponding items with maximum prefix length, i.e., at be bottom of the hierarchy, with those of the Exact instance. This could be achieved with the update operation of the Cormode algorithm, since it supports updates with a count greater than 1. However, the update operation would periodically prune items from the Cormode trie while the counters are transferred. This would lead to suboptimal results, since pruning depends on the order in which items are inserted into the internal hierarchical data structure.

Consider, for example, two items a, b and their common parent p whose frequencies satisfy the following conditions before the update function is invoked: $f_p + f_a < \epsilon N$ and $f_p + f_a + f_b \geq \epsilon N$. If a compression occurs after both update operations for a and b have been performed, the two items are aggregated into p, but p is not aggregated any further, since $f_p + f_a + f_b \geq \epsilon N$. However, if the update operation for a is invoked first and a compression occurs immediately afterwards, a would be aggregated into p and p itself would be aggregated into one of its ancestor nodes (since $f_p + f_a < \epsilon N$). The additional aggregation steps can degrade the error estimation of the Cormode instance by increasing the error estimation of ancestor nodes that are visited while propagating p through the hierarchy. As a result, the accuracy of the calculated HHHs decreases unnecessarily, since enough information was available in the Exact instance

Algorithm 2 HHH-transition from Exact to Cormode

```
1: procedure EXACT_TO_CORMODE(E, ϵ)
2:     C ← new Cormode(ϵ)
3:     for p ∈ E do          ▷ Move items to lowest layer of C
4:         C_p ← E_p
5:         delete p from E
6:     end for
7:     for l = E.H downto 1 do          ▷ Build Cormode trie
8:         for p ∈ C^(l) do
9:             if par(p) ∉ C^(l−1) then
10:                C_p ← 0
11:            end if
12:            if C_p < ϵN then
13:                C_par(p) ← C_par(p) + C_p
14:                delete p from C^(l)
15:            end if
16:        end for
17:    end for
18:    return C
19: end procedure
```

to prevent this, as indicated by the first case. Hence, to avoid introducing unnecessary errors, the data structure of the Cormode instance is not compressed while a transition is in progress. Furthermore, after iterating over each counter stored in the Exact instance, its value is never again used during a transition, such that the memory occupied by an item can be freed immediately after it has been transferred (Line 5 of Algorithm 2). Insofar, the transition from Cormode to Exact is performed in-place with minimal memory overhead. Once the elements at the bottom of the hierarchy have been initialized, the hierarchical data structure of the Cormode instance can be compressed as normal, except that missing parent nodes in the data structure are created and initialized as needed (Line 9-11). The Cormode algorithm can then be executed as normal using the resulting data structure.

c) Exact → DHHH and RHHH: The transition from Exact to DHHH is outlined in Algorithm 3. It transfers the information from an Exact instance E by updating every layer of a target DHHH instance D. For this, we compute the l-th generalization of each element in E and invoke the update operation of the HH instance of D at layer l with the respective item count using modified update operations (Lines 7 and 8 of Algorithm 3). Once all update operations for an item are finished, it can be removed from E to avoid unnecessary memory consumption.

We use the SpaceSaving algorithm as HH algorithm in the various DHHH layers. The effect of the update operation of SpaceSaving on its internal data structures depends on the order in which items are processed. This is usually predetermined by the ordering of items in a data stream. However, that information is no longer available at the time of transitioning, such that the order in which items are transferred by a transition is inherently arbitrary. It can be chosen in a way that minimizes the error estimation of the SpaceSaving algorithm.

Algorithm 3 HHH-transition from Exact to DHHH

▷ sort(X) sorts the set of items X by their frequency in ascending order

```
1:  procedure EXACT_TO_DHHH(E, ε)
2:      D ← new DHHH(E.H, ε)
3:      E ← sort(E)
4:      for p ∈ E do
5:          x ← p
6:          for l = 1 to E.H do
7:              x ← par(x)          ▷ The l-th generalizaton of p
8:              D^(l).update(x, f_p)
9:          end for
10:         delete e from E
11:     end for
12:     return D
13: end procedure
```

When the update operation of a SpaceSaving instance S is invoked on an item x with frequency f_x that is currently not tracked by S, x replaces the item y with lowest frequency f_y in S. The frequency of y then serves as the error estimation Δ_x of the new item x to compensate for the removal of a previously tracked item. Clearly, when the item order can be chosen arbitrarily and $f_x < f_y$, it is preferable to insert x first and replace it with y later, since the resulting error estimation $\Delta_y = f_x$ of y is lower than $\Delta_x = f_y$ of x would be, if the sequence of update operations was reversed. To ensure a beneficial ordering of update operations, Algorithm 3 sorts items tracked in the Exact instance prior to transferring them to the target DHHH instance (Line 3).

Similar to DHHH → RHHH the transition Exact → RHHH can be realized by multiplying item frequencies and error estimations with the update probability of HH instances in the RHHH algorithm.

d) DHHH and RHHH → Cormode: While transferring items Algorithm 4 reconstructs the trie data structure of Cormode. It iterates over all layers of the DHHH instance, beginning with longest prefixes. The frequency estimations for prefixes stored in the DHHH instance must be adjusted in two ways before transferring them to Cormode. First, DHHH stores the total accumulated frequencies of all descendants of a prefix, while Cormode summarizes frequencies only as long as the aggregation threshold ϵN is not exceeded. Consequently, the frequency f_p of a prefix p stored in DHHH is reduced by the accumulated frequencies of all descendant prefixes $\langle p \rangle - p$ before it is transferred to the Cormode instance. Second, the frequency estimation of the SpaceSaving algorithm includes the error estimation for an item, while Cormodes estimation does not. Therefore, the transition adjusts f_p accordingly to match the representation in the Cormode instance (line 5).

The error estimation for entries in Cormode can be taken directly from the DHHH instance, since both algorithms yield the same bounds on the maximum overestimation of item frequencies when given the same threshold parameter ϵ. The

Algorithm 4 HHH-transition from DHHH to Cormode

▷ completeTree(C) adds missing inner nodes to the trie data structure of a Cormode instance C

```
1:  procedure DHHH_TO_CORMODE(D)
2:      C ← new Cormode(D.ε)          ▷ Same ε for C and D
3:      for l = D.H downto 1 do
4:          for p ∈ D^(l) do
5:              x ← D_p − Δ_{D,p} − Σ_{q∈⟨p⟩−p} C_q
6:              C_p ← x
7:              Δ_{C,p} ← Δ_{D,p}
8:              C.N ← C.N + x
9:              delete p from D^(l)
10:         end for
11:     end for
12:     completeTree(C)
13:     C.compress()                  ▷ Optional compression
14:     return C
15: end procedure
```

Algorithm 5 HHH-transition from Cormode to DHHH

```
1:  procedure CORMODE_TO_DHHH(C)
2:      C ← new DHHH(C.ε)             ▷ Same ε for D and C
3:      for l = C.H downto 1 do
4:          for p ∈ C^(l) do
5:              D^(l).update(p, C_p + Δ_{C,p}, Δ_{C,p})
6:              C_{par(p)} ← C_{par(p)} + C_p
7:          end for
8:      end for
9:      return D
10: end procedure
```

transition algorithm continuously adjusts the total frequency count of the Cormode instance and deletes items from SpaceSaving instances once they are transferred. The transfer of individual items does not necessarily result in a complete trie structure, since some inner nodes may not be created during this process. The missing nodes are subsequently added, once all items have been transferred. A final, optional compression step can be performed on the Cormode instance, to reduce its memory footprint and adopt its normal performance characteristics more quickly.

With the transitions RHHH → DHHH and DHHH → Cormode given, the transition from the RHHH algorithm can be constructed as the (indirect) chain RHHH → DHHH → Cormode.

e) Cormode → DHHH and RHHH: Transitions to the DHHH and RHHH algorithms require the reconstruction of the various HH instances. The transition to DHHH outlined in Algorithm 5 transfers counters stored in the Cormode trie data structure layer by layer to the target algorithms HH instances. For a given prefix p of length l and a DHHH instance with H layers, the frequency of f_p without error is $\sum_{q \in \langle p \rangle, |q|=H} f_q$. This sum is recomputed by summarizing frequencies stored in

the Cormode instance while traversing the trie from bottom to top (line 6). It is subsequently incremented by the error stored in the corresponding node in the Cormode trie and transferred to the DHHH instance by invoking the (modified) update operation of the HH instance at layer l (line 5). Recall that the error estimation of Cormode is calculated as the maximum error incurred by pruning and recreating nodes in the trie. This error is not necessarily identical to that computed by an HH instance of the DHHH algorithm. However, the HH instance is required to provide enough counters to output all items p with an estimated frequency $f_p^\star \geq \epsilon N$ when its query operation is invoked with parameter ϵ (instead of arbitrary ϕ). This is guaranteed by the SpaceSaving algorithm. Consequently, all items in the corresponding layer of the Cormode trie with frequency $f_p^\star \geq \epsilon N$ can be inserted into the HH instance without displacing any previously transferred item.

The transition from Cormode to RHHH can be realized with the indirect transition Cormode $\rightarrow$ DHHH $\rightarrow$ RHHH.

f) Cormode, DHHH and RHHH $\rightarrow$ Exact: Pruning entries from the data structures of a Cormode, DHHH or RHHH instance incurs a loss of information that cannot be recovered. As a result, the counters of an Exact instance E cannot be fully reconstructed. However, E can take the (partial) information available in an instance of either Cormode, DHHH or RHHH into account, when performing a query operation. For this, we extend E with an auxiliary data structure T composed of multiple layers. At each layer l, T stores the frequencies f_p for prefixes p of length $|p| = l$. When transitioning from an instance X of Cormode or DHHH, we first compute the result H of the query operation of X with parameter $\phi = \epsilon$. Recall that ϵ is the lowest possible choice for ϕ, hence, yielding the most accurate HHHs. The frequencies f_p of prefixes $p \in H$ with maximum length are then stored directly in the normal counters of E. Shorter prefixes are transferred to T, which stores the entry $T_p^{(l)} = f_p$ at layer $l = |p|$.

To take the additional information in T into account, we modify the query operation of E slightly: when computing the HHHs HHH_l with prefix length l, the value F_p in eq. 1 is incremented if T has a corresponding entry $T_p^{(l)}$:

$$HHH_l = HHH_{l-1} + \{p : |p| = l \wedge F_p + T_p^{(l)} \geq \phi \cdot N\}.$$

With this modification, E and X compute identical HHHs immediately after a transition and eq. 2 holds for entries in E. However, this bound improves as E processes additional items: let N be the number of items processed until the time of transition and $N' = \lambda N$ ($\lambda > 1$) the total number of items processed at a later point in time. Since the Exact algorithm does not introduce any additional error, the upper bound ϵN in eq. 2 remains constant. Consequently, the error estimation $|f_p - f_p^\star| \leq \epsilon N = \epsilon \lambda^{-1} N'$ improves relative to N', since the factor $\epsilon' = \epsilon \lambda^{-1}$ decreases as N' increases. Hence, E converges to the error-free processing of streaming data that is normally performed by the Exact algorithm. The overhead in memory consumption incurred by storing the auxiliary data structure T is negligible, since the memory footprint of the Exact algorithm is typically several orders of magnitude higher than that of Cormode, DHHH and RHHH.

TABLE II
EVALUATION PARAMETERS FOR HHH ALGORITHMS

Parameter	Value	Effect
ϵ	0.007	Threshold parameter indicating permissible pruning of prefixes
ϕ	0.01	Threshold for minimum HHH frequency
δ	0.1	Probability of frequency overestimation when using the RHHH algorithm
s	1.0	Sampling rate of the RHHH algorithm

The transition from RHHH to Exact can finally be realized with the indirect transition RHHH $\rightarrow$ DHHH $\rightarrow$ Exact.

IV. EVALUATION

We evaluate the feasibility of HHH-transitions through trace-driven emulation based on authentic network traffic taken from MAWILab [8]. Our experiments focus on adopting performance characteristics of target HHH algorithms through transitions and the impact that retaining monitoring information has on post-transition accuracy.

A. Implementation and Execution Environment

We implemented the one-dimensional version of the four HHH algorithms in C++, making extensive use of containers from the Standard Template Library (STL) to implement algorithm-specific data structures. The evaluation was performed on a single server with an Intel(R) XEON(R) CPU L5420 @ 2.5GHz and 32 GB RAM running Ubuntu 16.04-LTS. Update and query operations of different algorithms were executed in strictly sequential order to avoid interference of I/O-operations with timing-measurements.

B. Network Traffic

For our evaluation, we selected two different 15-minute network traces from the MAWILab. These traces were recorded by a large Japanese ISP on November, 23rd and 24th 2017 and contain about 200 million packets in total. Each trace was segmented into one-minute network traces of approximately the same size, yielding a total of 30 network traces that were independently processed by several instances of HHH algorithms as outlined below. The same choice of parameters given in Table II was applied to all algorithms (where applicable). The HHH instances monitor the frequencies of IP source addresses occuring in the network traces at bit granularity.

C. Measurement of Performance Characteristics

a) Computation time: We measure the time required for the query and update operations of the HHH algorithms. In general, both timings are non-constant when processing streaming data, but rather depend on the internal state of an HHH instance. To provide accurate measurements, we use the `high_resolution_clock`-class of the C++ Standard Template Library (STL), which offers nanosecond accuracy on current Linux systems. To avoid side effects from I/O-operations and preprocessing steps, time samples are collected

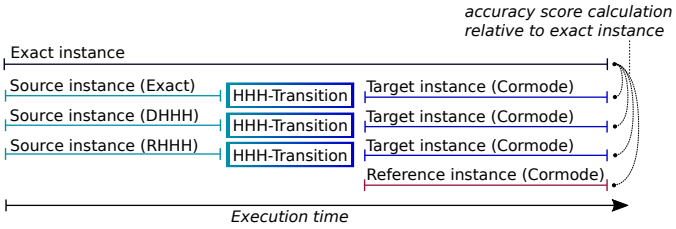

Fig. 5. Comparative execution of exact, source, target and reference HHH instances for accuracy score calculation.

immediately before and after the execution of the update and query operations. Furthermore, the operations of different HHH instances are executed strictly sequentially to avoid any mutual interference.

b) Memory consumption: The amount of used memory is calculated directly from the size and number of entries in the data structures of an HHH instance. Our C++-implementation deletes unused data structures immediately. This avoids side effects caused by asynchronous garbage collection with respect to the calculation of utilized memory and interference with time-measurements of subsequent computations.

c) Accuracy score: To give a measure for the accuracy of the set of HHHs computed by each algorithm, we use the *Optimistic Genealogy Measure (OGM)* [9]. For two HHH sets H_1 and H_2, the OGM calculates a score $s \in [0, 1]$ measuring the similarity of H_1 and H_2 with respect to their hierarchical structure. The OGM itself is asymmetric. Symmetry can be achieved with the *symmetric OGM (sOGM)* score that is defined as the average of two asymmetric OGM scores.

Since the calculation of HHHs by an Exact instance E is error-free, it serves as a baseline against which the results of any other HHH algorithm X can be compared. That is, when E and X process the same streaming data, the sOGM allows the comparison of the HHHs H_X and H_E computed by the query operations of X and E at any given time. Hence, the *accuracy score* of the result of a single algorithm H_X can be expressed as its sOGM score in relation to the Exact algorithm:

$$AccuracyScore(\mathsf{H}_\mathsf{X}) \coloneqq sOGM(\mathsf{H}_\mathsf{X}, \mathsf{H}_\mathsf{E}).$$

The accuracy score does not depend on which of the two sOGM-parameters is substituted by H_E, since sOGM is symmetric.

D. Experimental Results

To compare performance characteristics before and after performing an HHH-transition to a target algorithm, our test setup sequentially processes all 30 network traces through the following set of HHH instances:

- An instance of the Exact algorithm (*exact instance*) that processes streaming data over the entire duration of a network trace. This instance operates error-free and preserves monitoring information during a transition. It serves as the baseline for the calculation of the accuracy score (cf. section IV-C).

TABLE III
POST-TRANSITION RESOURCE UTILIZATION

		Exact	Cormode	DHHH	RHHH
Reference instance	Q	$7.91 \cdot 10^8$	$4.46 \cdot 10^5$	$3.45 \cdot 10^6$	$5.77 \cdot 10^6$
	U	$1.83 \cdot 10^0$	$4.82 \cdot 10^0$	$5.03 \cdot 10^1$	$4.27 \cdot 10^0$
	M	$2.81 \cdot 10^6$	$6.29 \cdot 10^4$	$2.48 \cdot 10^5$	$2.33 \cdot 10^5$
Source instance		Target instance: Exact	Cormode	DHHH	RHHH
Exact	Q		$5.50 \cdot 10^5$	$3.35 \cdot 10^6$	$2.05 \cdot 10^6$
	U		$4.90 \cdot 10^0$	$4.87 \cdot 10^1$	$4.22 \cdot 10^0$
	M		$6.18 \cdot 10^4$	$2.49 \cdot 10^5$	$2.42 \cdot 10^5$
Cormode	Q	$7.91 \cdot 10^8$		$3.42 \cdot 10^6$	$2.32 \cdot 10^6$
	U	$1.76 \cdot 10^0$		$4.99 \cdot 10^1$	$4.24 \cdot 10^0$
	M	$2.81 \cdot 10^6$		$2.51 \cdot 10^5$	$2.41 \cdot 10^5$
DHHH	Q	$7.92 \cdot 10^8$	$3.80 \cdot 10^5$		$2.39 \cdot 10^6$
	U	$1.74 \cdot 10^0$	$4.68 \cdot 10^0$		$4.23 \cdot 10^0$
	M	$2.81 \cdot 10^6$	$6.19 \cdot 10^4$		$2.43 \cdot 10^5$
RHHH	Q	$7.94 \cdot 10^8$	$6.25 \cdot 10^5$	$3.55 \cdot 10^6$	
	U	$1.74 \cdot 10^0$	$4.22 \cdot 10^0$	$4.98 \cdot 10^1$	
	M	$2.83 \cdot 10^6$	$1.03 \cdot 10^5$	$2.51 \cdot 10^5$	

Q: average query time [ns]
U: average update time [ns]
M: average memory consumption [Byte]

- A *source instance* for every algorithm other than the target algorithm. Source instances process streaming data until a transition is executed. The acquired monitoring information is then transferred to a new target instance (see below) by a transition.
- A *target instance* for each distinct source instance. These instances of the target algorithm are created by the transitions.
- A *reference instance*, which is an instance of the target algorithm that starts processing streaming data immediately after a transition. A reference instance represents the normal execution of a target algorithm, i.e., without retaining monitoring information.

Fig. 5 illustrates these instances for an HHH-transition to the Cormode algorithm. Measurement samples are obtained from the various instances every second while processing network traces, except for the duration of a transition.

a) Resource-oriented performance characteristics: In this section we present the evaluation results for resource-oriented performance characteristics realized by the target instances of HHH-transitions. Table III summarizes the average resource utilization with respect to CPU-time (query and update time) as well as memory consumption for each transition from a source instance to a target instance. For comparison, we include measurements obtained from the execution of reference instances for each target algorithm, i.e., the normal processing of the second half of the network traces.

The results indicate, that the target instances of transitions to Exact, DHHH and Cormode adopt the performance characteristics of the corresponding target algorithm, showing only slight deviations from the reference instance. Deviations can be attributed to different initial states of the target instances

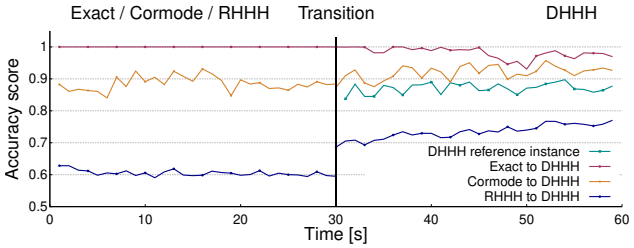

Fig. 6. Accuracy score of the source, target and reference instances of a transition from Exact, Cormode and RHHH to DHHH.

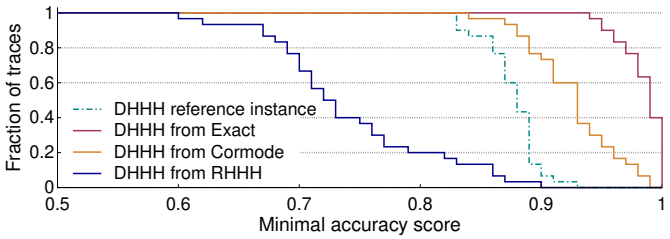

Fig. 7. CDF of minimal achieved accuracy score after transitioning to DHHH.

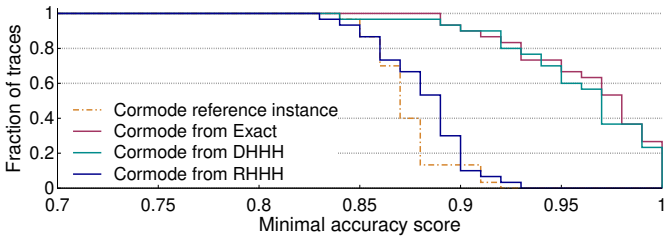

Fig. 8. CDF of minimal achieved accuracy score after transitioning to Cormode.

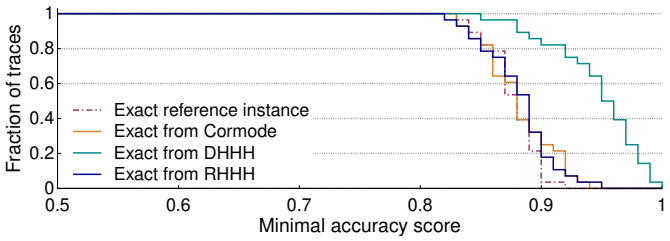

Fig. 9. CDF of minimal achieved accuracy score after transitioning to Exact.

(resulting from transitions or a clean start) and dissimilar subsequent aggregation decisions of the target algorithm. Transitions to the RHHH algorithm realize its performance characteristics with respect to update time and memory consumption. However, the reference instance of RHHH has a significantly higher query time than the target instances of a transition. This is the result of RHHH requiring a certain amount of processed items before converging to a sufficiently accurate computation of HHHs. Initially RHHH overestimates item frequencies, resulting in an elevated HHH count, which in turn slows down the query operation. The transfer of monitoring data essentially skips the convergence phase of the RHHH algorithm, resulting in decreased query time.

b) Post-transition accuracy score: When transitioning to another HHH instance, retaining previously acquired monitoring information influences the accuracy score of subsequent HHH computations. Complementary, the effect that the loss of monitoring information has on accuracy can be measured by a direct comparison of the accuracy scores obtained from a target and reference instance. (Recall that these scores are relative to an exact instance that is error-free and retains all monitoring information.) To measure this effect, we calculate the average accuracy scores of the source, target and reference instances taken over all network traces.

As an example, Fig 6 outlines their development over time for all HHH instances (except the exact instance) when transitioning to the DHHH algorithm. Compared to the reference instance, which has no knowledge of pre-transition monitoring information, the accuracy score of the target instances is increased when the transition originates from the Exact or Cormode algorithms. Furthermore, increased accuracy can be maintained over the entire second half of the network traces. However, when processing additional data stream items, the target DHHH instance reverts to its normal behavior. This can be observed in the loss of accuracy over time of the instance that originated from the Exact algorithm.

To quantify the increase in accuracy after transitioning to the DHHH algorithm we calculate the arithmetic mean of the accuracy scores calculated during the last 30 seconds of each individual network trace, i.e., after performing the transitions. Fig. 7 depicts the cumulative distribution function (CDF) of the minimum accuracy score achieved by each fraction of network traces. Comparing the reference instance and the target instance originating from the Exact algorithm shows an increase of 0.11 for 80% of all network traces. Transitioning from the Cormode instance, which supplies less precise monitoring information than the Exact algorithm, still yields an increase of 0.04 for 60% of the network traces.

However, Fig. 7 indicates that a transition from RHHH to DHHH reduces overall post-transition accuracy. The minimum accuracy score achieved after the transition drops by 0.16 for 60% of the network traces (compared to the reference instance). This can also be observed in Fig. 6, which indicates that the introduced error persists over time. Consequently, using a completely new instance of the DHHH algorithm is preferable to a transition that retains highly erroneous monitoring data.

Fig. 8 shows the same CDF for a transition to the Cormode algorithm. The accuracy score is increased by about 0.07 and 0.08 for 80% of all network traces when transitioning from DHHH and Exact (resp.). In this case, transitioning from RHHH does not incur a comparable loss of accuracy, but instead an increase in post-transition accuracy is realized (about 0.01 for 60% of all network traces). This is the result of Cormode typically operating with an accuracy that is more comparable to RHHH than DHHH.

Finally, the CDF of accuracy scores for transitions to the Exact algorithm is given in Fig. 9. Transitioning from Cormode or RHHH shows a slight increase (0.01 to 0.03) in the accuracy score of a lower fraction of network traces ($\leq 60\%$). However, this increase is more significant when transitioning from DHHH: the accuracy score of the target instance increases by about 0.06 for 80% of all network traces.

We omit the results for transitions to the RHHH algorithm due to space constraints.

V. Related Work

Computing HHHs directly in the data plane can increase the efficiency of network monitoring systems. DREAM [2] utilizes TCAM rules in a switch to identify HHHs. It allocates the available TCAM resources dynamically, adjusting the number of rules as needed to maintain a predetermined accuracy threshold throughout the monitoring process. The approach effectively balances two of the previously mentioned performance characteristics (memory and accuracy). The work of [10] performs network-wide heavy hitter detection through a combination of (P4-based) switches and a common coordinator. The coordinator repeatedly calculates adaptive thresholds to determine the amount of monitoring information that is sent by the switches, effectively trading accuracy of distributed HH computation against communication overhead. It is unclear if this balance could benefit from HHH-transitions when extending the approach to network-wide computation of HHHs.

Complementary to retaining monitoring information, Memento [11] uses sliding windows to discard outdated traffic statistics (as opposed to epoch-based approaches). Specifically, it utilizes RHHH in conjunction with the HH algorithm WCSS [12] to approach line speed. Since the sampling rate of RHHH can be adapted, a trade-off between accuracy and update speed can be realized at runtime. However, the sliding window discards monitoring information regardless of its origin. In a DDoS scenario, the combination of retaining monitoring information collected from benign traffic (to maintain service availability) and frequently discarding statistics of malicious traffic (for fast adaptation to emerging attack patterns) may improve overall Return-on-Mitigation.

VI. Conclusion and Future Work

Update time, query time, memory consumption and accuracy are important performance characteristics of HHH algorithms. By exchanging HHH algorithms, a monitoring system can adapt these characteristics to emerging situations in a network. Replacing the HHH algorithms usually incurs the loss of previously acquired monitoring information and a reduced accuracy. This paper introduced HHH-transitions to preserve already available information by transferring it between two instances of different HHH algorithms. We presented transitions for four one-dimensional HHH algorithms (Exact, Cormode, DHHH, RHHH) and how they can increase average post-transition accuracy.

Future work includes a deeper theoretical analysis of HHH transitions, their extension to multi-dimensional HHH variants, state synchronization between distributed instances of different HHH algorithms and to apply the concept of transferring state information to a broader range of algorithm classes (e.g., different kinds of neural networks).

Acknowledgment

This work was supported by the German Federal Ministry of Education and Research within the framework of the project KASTEL_SVI in the Competence Center for Applied Security Technology (KASTEL).

References

[1] V. Sekar, N. G. Duffield, O. Spatscheck, J. E. van der Merwe, and H. Zhang, "Lads: Large-scale automated ddos detection system." in *USENIX Annual Technical Conference, General Track*, 2006, pp. 171–184.

[2] M. Moshref, M. Yu, R. Govindan, and A. Vahdat, "Dream: Dynamic resource allocation for software-defined measurement," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, ser. SIGCOMM '14. New York, NY, USA: ACM, 2014, pp. 419–430. [Online]. Available: http://doi.acm.org/10.1145/2619239.2626291

[3] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava, "Finding hierarchical heavy hitters in streaming data," *ACM Trans. Knowl. Discov. Data*, vol. 1, no. 4, pp. 2:1–2:48, Feb. 2008. [Online]. Available: http://doi.acm.org/10.1145/1324172.1324174

[4] ——, "Finding hierarchical heavy hitters in data streams," in *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29*, ser. VLDB '03. VLDB Endowment, 2003, pp. 464–475. [Online]. Available: http://dl.acm.org/citation.cfm?id=1315451.1315492

[5] R. Ben Basat, G. Einziger, R. Friedman, M. C. Luizelli, and E. Waisbard, "Constant time updates in hierarchical heavy hitters," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17. New York, NY, USA: ACM, 2017, pp. 127–140. [Online]. Available: http://doi.acm.org/10.1145/3098822.3098832

[6] A. Metwally, D. Agrawal, and A. El Abbadi, "Efficient computation of frequent and top-k elements in data streams," in *Proceedings of the 10th International Conference on Database Theory*, ser. ICDT'05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 398–412. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-30570-5_27

[7] M. Mitzenmacher, T. Steinke, and J. Thaler, "Hierarchical heavy hitters with the space saving algorithm," in *Proceedings of the Meeting on Algorithm Engineering & Expermiments*, ser. ALENEX '12. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2012, pp. 160–174. [Online]. Available: http://dl.acm.org/citation.cfm?id=2790265.2790281

[8] R. Fontugne, P. Borgnat, P. Abry, and K. Fukuda, "Mawilab: Combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking," in *Proceedings of the 6th International Conference*, ser. Co-NEXT '10. New York, NY, USA: ACM, 2010, pp. 8:1–8:12. [Online]. Available: http://doi.acm.org/10.1145/1921168.1921179

[9] P. Ganesan, H. Garcia-Molina, and J. Widom, "Exploiting hierarchical domain structure to compute similarity," *ACM Trans. Inf. Syst.*, vol. 21, no. 1, pp. 64–93, Jan. 2003. [Online]. Available: http://doi.acm.org/10.1145/635484.635487

[10] R. Harrison, Q. Cai, A. Gupta, and J. Rexford, "Network-wide heavy hitter detection with commodity switches," in *Proceedings of the Symposium on SDN Research*, ser. SOSR '18. New York, NY, USA: ACM, 2018, pp. 8:1–8:7. [Online]. Available: http://doi.acm.org/10.1145/3185467.3185476

[11] R. B. Basat, G. Einziger, I. Keslassy, A. Orda, S. Vargaftik, and E. Waisbard, "Memento," *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies - CoNEXT '18*, 2018. [Online]. Available: http://dx.doi.org/10.1145/3281411.3281427

[12] R. Ben-Basat, G. Einziger, R. Friedman, and Y. Kassner, "Heavy hitters in streams and sliding windows." in *INFOCOM*, 2016, pp. 1–9.