

# Tiresias: Online Anomaly Detection for Hierarchical Operational Network Data

Chi-Yao Hong, Matthew Caesar  
University of Illinois at Urbana-Champaign  
{cyhong|caesar}@illinois.edu

Nick Duffield, Jia Wang  
AT&T Research Labs  
{duffield|jiawang}@research.att.com

## ABSTRACT

*Operational network data*, management data such as customer care call logs and equipment system logs, is a very important source of information for network operators to detect problems in their networks. Unfortunately, there is lack of efficient tools to *automatically* track and detect anomalous events on operational data, causing ISP operators to rely on manual inspection of this data. While anomaly detection has been widely studied in the context of network data, operational data presents several new challenges, including the volatility and sparseness of data, and the need to perform fast detection (complicating application of schemes that require offline processing or large/stable data sets to converge).

To address these challenges, we propose Tiresias, an automated approach to locating anomalous events on hierarchical operational data. Tiresias leverages the hierarchical structure of operational data to identify high-impact aggregates (e.g., locations in the network, failure modes) likely to be associated with anomalous events. To accommodate different kinds of operational network data, Tiresias consists of an online detection algorithm with low time and space complexity, while preserving high detection accuracy. We present results from two case studies using operational data collected at a large commercial IPTV network operated by a Tier-1 ISP: customer care calls log and set-top box crashes log. By comparing with a reference set verified by the ISP’s operational group, we validate that Tiresias can achieve  $> 94\%$  accuracy in locating anomalies. Tiresias also discovers several previously unknown anomalies in the ISP’s customer care cases, demonstrating its effectiveness.

## 1. INTRODUCTION

Network monitoring is becoming an increasingly critical requirement for protecting large-scale network infrastructure. Criminals and other Internet miscreants make use of increasingly advanced strategies to propagate spam, viruses, worms, and exercise DoS attacks against their targets [21, 22, 27]. However, the growing size and functionality of modern network deployments have led to an increasingly complex array of failure modes and anomalies that are hard to detect with traditional approaches [6, 16].

Detection of statistically anomalous behavior is an important line of defense for protecting large-scale network infrastructure. Such techniques work by analyzing *network traffic* (such as traffic volume, communication outdegree, and rate of routing updates), characterizing “typical” behavior, and extracting and reporting statistically rare events as alarms [4, 9, 14, 15, 29].

While these traditional forms of data are useful, the landscape of ISP management has become much broader. Commercial ISPs collect an increasingly rich variety of *operational data* including network trouble tickets, customer care cases, equipment system logs, and repair logs. Several properties of operational data make them highly useful for network monitoring. Operational data is often annotated with semantics. Customer care representatives annotate call logs with detailed characteristics on the end user’s experience of the outage, as well as information on the time and location of the event. Repair logs contain information about the underlying root cause of the event, and how the problem was repaired. This can enable us to gain more detailed information on the meaning and the type of event that occurred from a smaller amount of data. In addition, operational data is often described in an explicitly-described hierarchical format. While the underlying signal may be composed across multiple dimensions, correlations across those dimensions are explicitly described in the provided hierarchy, allowing us to detect anomalies at higher levels of the tree when information at the leaves is too sparse.

While the operational data can serve as rich sources of information to help ISPs manage network performance, there is still a lack of an effective mechanism to assist ISP operation teams in using these data sources to identify network anomalies. For example, currently a Tier-1 ISP operation group *manually* inspects  $> 300,000$  customer care calls in a working day in order to deduce network incidents or performance degradations. Unfortunately, the sheer volume of data makes it difficult to manually form a “big picture” understanding of the data, making it very challenging to locate root causes of performance problems accurately.

To deal with this, it would be desirable to *automatically* detect anomalies in operational data. Unfortun-

nately, several unique features of operational data complicate application of previous work in this space:

**Operational data is complex:** Operational data is *sparse* in the time domain, with a low rate of underlying events; it is composed of data spread over a large hierarchical space, reducing magnitude of the signal. Operational data is also *volatile*, being sourced from a data arrival rate that changes rapidly over time, complicating the ability to detect statistical variations. Unfortunately, the fact that user calls arrive on the order of minutes (as opposed to sub-millisecond reporting in traditional network traffic monitoring systems) means that the ISP may need to make a decision on when to react after a few calls.

**Operational data represents urgent concerns:** Operational data contains mission-critical failure information, often sourced *after* the problem has reached the customer. Operational data such as call logs reflects problems crucial enough to motivate the customer to contact the call center. Moreover, call logs that affect multiple customers typically reflect large-scale outages that need immediate attention. Unfortunately, the richness of operational data incurs significant memory / processing requirements, complicating use of traditional anomaly detection approaches (which rely on offline processing).

To address these challenges, we propose Tiresias, an automated approach to locating anomalous events on operational data. To leverage the hierarchical nature of operational data, we build upon previous work on *hierarchical heavy hitter*-based anomaly detection [9, 30]. These works identify heavy hitter positions in the hierarchy which have higher aggregated data counts. However, these works relied on offline processing for anomaly detection, which does not scale to operational data (refer to §8 for further details). We extend these works to perform anomaly detection in an *online* and *adaptive* fashion. To quickly detect anomalies without incurring high memory / processing requirements, we propose novel algorithms to maintain a time series characterization of data at each heavy hitter that is updated in both time and hierarchical domain dynamically without requiring storage of older data. To avoid redundant reports of the same anomaly, our approach also automatically discounts data from descendants that are themselves heavy hitters.

We first characterize two large-scale operational datasets (customer care calls and set-top boxes crash logs; §2) via measurement. Then we formally define our problem (§3). We present a high-level overview of our design and its limitations (§4). To achieve online detection, we propose novel algorithms to update the time series in both time and hierarchical domain dynamically (§5). We adopt time series analysis to derive the data seasonality and perform time series based anomaly detection (§6). We evaluate the efficiency and effectiveness

of Tiresias using large-scale operational datasets (§7). Our findings suggest that Tiresias can scale to these real datasets with sufficiently low computational resources (§7.1). We compare the set of anomalies detected by Tiresias on customer call cases with an existing approach used by a major US broadband provider (§7.2). Tiresias discovers > 94% of the anomalies found by the reference method as well as many previously unknown anomalies hidden in the lower levels. We present related work (§8) before we conclude (§9).

## 2. CHARACTERIZING OPERATIONAL DATA

It is arguably impossible to accurately detect anomalous events without understanding the underlying data. We first present characterization results of two operational datasets (§2.1). Then we conduct a measurement analysis to study the properties of the operational datasets (§2.2).

### 2.1 Operational Datasets

The data used in this study was obtained from a major broadband provider of bundled Internet, TV and VoIP services in the US. We present the results based on the datasets collected from May to September 2010. Qualitatively similar results were found in other time periods. In this study, all customer identifying information was hashed to a unique customer index, in order to protect privacy. Two distinct operational datasets were used, which we now describe.

**Customer Care Call Dataset (CCD):** The study used data derived from calls to the broadband service customer care center. A record is generated for each such call. Based on the conversation with the customer, the care representative assigns a category to the call, choosing from amongst a set of *predetermined hierarchical* classes. We focus on analyzing customer calls that are related to performance troubles (service disruptions/degradation, intermittent connectivity, etc.) and discard records unrelated to performance issues (e.g., calls about provisioning, billing, accounting). The resulting operator-assigned category constitutes a *trouble description*, which is drawn from a hierarchical tree with a height of 5 levels. These categories indicate the broad area of the trouble: IPTV, home wireless network, Internet, email, digital home phone, and a more specific detail within the category. For example, [Trouble Management]  $\Rightarrow$  [TV]  $\Rightarrow$  [TV No Service]  $\Rightarrow$  [No Pic No Sound]  $\Rightarrow$  [Dispatched to Premise]. We summarize the distribution of number of customer tickets for the first-level categories (Table 1).

Each record also specifies information concerning the *network path* to the customer premises, which is drawn from a hierarchical tree comprising 5 levels. The top (root) level is SHO (Super Head-end Office), which is the main source of national IPTV content. Under SHO, there are multiple VHOs (Video Head-end Office) in the

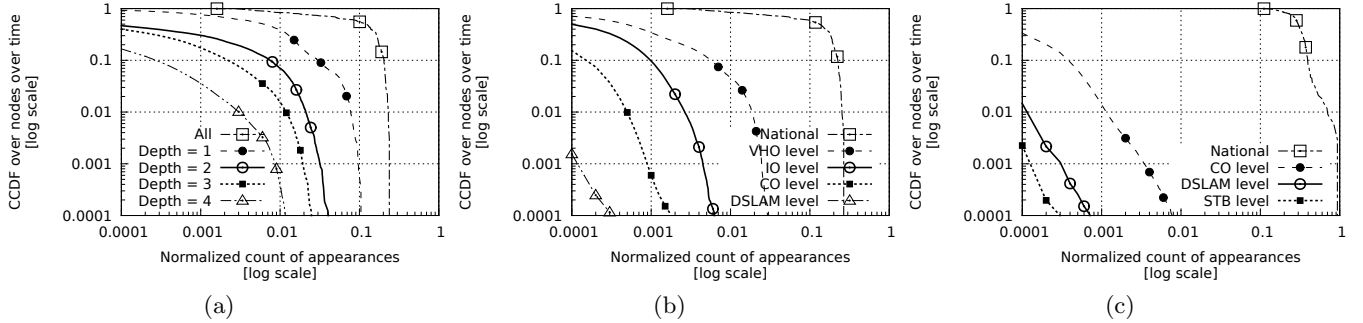


Figure 1: For each level in the hierarchy, the complementary cumulative distribution function of the normalized count of cases across nodes and timeunits. (a) trouble issues of CCD, (b) network locations of CCD, and (c) network locations of SCD. Note that both x- and y-axes are log scale.

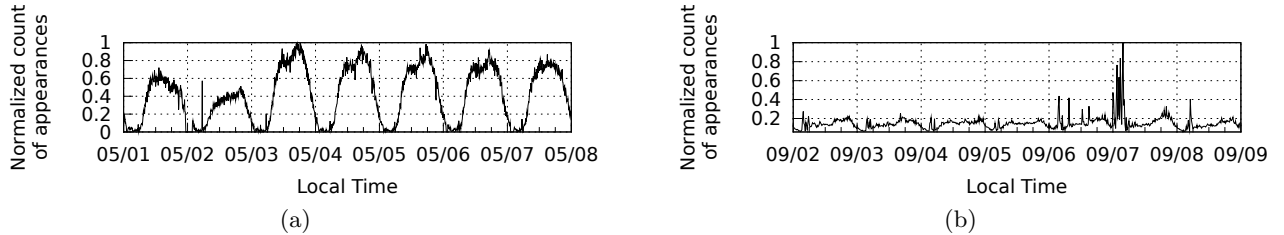


Figure 2: Two representative time series of the count of appearances in a 15-minute precision for the datasets (a) CCD and (b) SCD. The data count is normalized, by dividing it by the maximal count in the time series.

Ticket Types	Percentage(%)
TV	39.59
All Products	26.71
Internet	10.04
Wireless	9.26
Phone	8.46
Email	3.59
Remote Control	2.35

Table 1: CCD2 customer calls

Data	Type	Depth	Typical degree at $k$ th level			
			$k = 1$	$k = 2$	$k = 3$	$k = 4$
CCD	Trouble descr.	5	9	6	3	5
	Network path	5	61	5	6	24
SCD	Network path	4	2000	30	6	N/A

Table 2: Hierarchy properties

second level, each of which is responsible for the network service in a region. A VHO serves a number of IOs (intermediate offices), each of which serves a metropolitan area network. An IO serves multiple routers/switches called COs (central offices), and each CO serves multiple switches DSLAMs (digital subscriber line access multiplexers) before reaching a customer residence.

**STB Crash Dataset (SCD):** At the customer premises, a Residential Gateway serves as a modem and connects to one or more Set-Top Boxes (STBs) that serve TVs and enable user control of the service. Similar to other types of computer devices, an STB may crash, then subsequently reboot. This study used logs of STB crash events from the same broadband network, that are generated by the STB. By combining the STB identifier reported in the log, with network configuration information, network path information similar to that de-

scribed for CCD can be associated with each crash event. This information is represented in a hierarchical tree of 4 levels. The broad properties such as nodes degrees in different levels are summarized (Table 2).

## 2.2 Characteristics of Operational Data

**Sparsity:** We measure the case arrival rate for difference aggregates in the hierarchies (Fig. 1). The number of cases per node, especially for nodes in the lower levels, is very sparse over time in both data sets. For example, we observe that in about 93% (70%) timeunits we have no cases for nodes in the CO level in CCD (SCD). Anomaly detection over sparse data is usually inaccurate as it is hard to differentiate the anomalous and normal case [31]. However, larger localized bursts of calls relating to certain nodes do occur; these are obvious candidates for anomalies.

**Volatility:** In both datasets, we observed the number of cases varies significantly over time. For example, consider the root node of trouble issues in CCD (“All” in Fig. 1(a)), we observe that the case number at the 90<sup>th</sup> percentile of the appearance count distribution is  $\approx 35$  times higher than that at the 10<sup>th</sup> percentile. While we did not present in the figure, we observed that the sibling nodes in the hierarchy could have very different case arrival rates. The above observations imply that the set of heavy hitters could change significantly over time. Therefore, observing any fixed subset of nodes in the hierarchy could easily miss significant anomalies.

**Seasonality:** To observe the variation on arrival case counts, we measured the time series of the normalized count of appearances in 15-minute timeunits (Fig. 2). Overall, a diurnal pattern is clearly visible in both the datasets. This pattern is especially strong in CCD. In particular, the daily peak are usually around 4 PM, while the daily minimum is at around 4 AM. Moreover, the weekly pattern is observed in CCD where we have fewer cases in first two days (May 1 – 2, 2010), which are Saturday and Sunday (Fig. 2). Regardless of the differences observed in the two datasets, one common property is the presence of large spikes. Some spikes last only a short period (e.g., < 30 minutes at 5 AM, May 2nd in Fig. 2(a)), while some spikes last longer (e.g., > 5 hours at midnight, Sept 7th in Fig. 2(b)). These spikes are by no means unique over time – we observed a similar pattern in other time periods.

### 3. PROBLEM STATEMENT

In this section, we formally define the online anomaly detection problem. In particular, we first describe the operational data format, and then we define *hierarchical heavy hitter* which allows us to identify the high-impact aggregates. To detect if the event related to these heavy hitters is anomalous, we define *time series based anomaly detection*.

An input stream  $S = s_0, s_1, \dots$  is a sequence of operational network data  $s_i$ . Each data  $s_i = (k_i, t_i)$  consists of two components: *category*  $k_i$  provides classification information about the data; *time*  $t_i$  denotes the recorded date and time of  $s_i$ . The data can be classified into “timeunits” based on its time information  $t_i$ . The data category is drawn from a hierarchical domain (i.e., tree) where each node  $n$  in the tree is associated with a *weight* value  $A_n[k, t]$  for timeunit  $t$ . For leaf nodes, the weight  $A_n[k, t]$  is defined by the number of data items in  $S$  whose category is  $k$  and occurs in timeunit  $t$ . For non-leaf nodes (i.e., interior nodes), the weight  $A_n[k, t]$  is the summation of the weights of its children. We defined the *hierarchical heavy hitter (HHH)* as follows:

**DEFINITION 1.** (HHH; Hierarchical Heavy Hitter). Consider a hierarchical tree where each node  $n$  is weighted by  $A_n[k, t]$  for timeunit  $t$ . Then the set of HHH for timeunit  $t$  can be defined by  $\text{HHH}[\theta, t] = \{n | A_n[k, t] \geq \theta\}$  for a threshold  $\theta$ .

These heavy hitters (i.e., high-volume aggregates) are suspicious locations where anomalies (i.e., significant spikes) could happen at. Intuitively, if an aggregate has very little data in a timeunit, it is unlikely to see a huge spike there. We expect to capture most anomalies by setting a sufficient small heavy hitter threshold.

In Definition 1, the count of appearances of any leaf node will contribute to *all* its ancestors. Therefore, if we perform anomaly detection without taking the hierarchy into account, an anomaly in a leaf node may be reported

multiple times (at each of its ancestors). To address this, one could reduce the redundancy by removing a heavy hitter node  $n$  if its weight can be inferred from that of heavy hitter node  $n_d$  that is a descendant of  $n$ . To arrive at this more “compact” representation, we consider a more strict definition of HHHs to find nodes whose weight is no less than  $\theta$ , *after discounting the weights from descendants that are themselves HHHs*.

**DEFINITION 2.** (SHHH; Succinct Hierarchical Heavy Hitter). Consider a hierarchical tree where each node  $n$  is associated with a *modified weight* value  $W_n[k, t]$  for timeunit  $t$ . For leaf nodes, the modified weight  $W_n[k, t]$  is equal to the original weight  $A_n[k, t]$ . For non-leaf (internal) nodes, the modified weight  $W_n[k, t]$  is the summation of weights of its children that are themselves *not* a heavy hitter; In other words, given the set of SHHHs, denoted by  $\text{SHHH}[\theta, t]$ , the modified weight for non-leaf nodes can be derived by

$$W_n[k, t] = \sum_m W_m[k, t] : m \in \text{child}(n) \cap m \notin \text{SHHH}[\theta, t]$$

and the succinct hierarchical heavy hitter set is defined by  $\text{SHHH}[\theta, t] = \{n | W_n[k, t] \geq \theta\}$ .

Notice that both the set of SHHHs and the modified weights are unique, i.e., there can be only one heavy hitter set satisfying the above recursive definition<sup>1</sup>. An intuitive explanation is that the question of whether each node joins the set only depends on the weight of its descendants. It is not hard to see that a bottom-up traversal on the hierarchy would give the correct set. We will use this definition throughout this paper (for brevity, we will write “heavy hitter” instead of “succinct hierarchical heavy hitter” hereafter).

To detect anomalous events, we will perform forecasting models on heavy hitter nodes to detect anomalous events. As heavy hitter nodes could change over time instances, we need to construct the time series only for nodes which are heavy hitter in the *most recent* timeunit.

**DEFINITION 3.** (Time Series). For each timeunit  $t$ , consider a hierarchical tree where each node  $n$  is weighted by  $A_n[k, t]$ . Given the heavy hitter set in the *last* timeunit ( $t = 1$ ), i.e.,  $\text{SHHH}[\theta, 1]$ , the time series of any heavy hitter node  $n \in \text{SHHH}[\theta, 1]$  is denoted by  $T[n, t] = A_n[k, t] - \sum_m A_m[k, t] : m \in \text{child}(n) \cap m \in \text{SHHH}[\theta, 1]$ , for  $1 \leq t \leq \ell$ , where  $\ell$  is the length of time series.

<sup>1</sup> Suppose that more than one solution exists. Then there must exist one node  $n$  who is in one set  $\text{SHHH}^{(1)}$  but not in another set  $\text{SHHH}^{(2)}$ . Then we know  $W_n^{(1)} \neq W_n^{(2)}$ . By definition, there must exist a node  $m \in \text{child}(n)$  where  $W_m^{(1)} \neq W_m^{(2)}$ . Repeating the above procedures, we end up having a leaf node  $p$  where  $W_p^{(1)} \neq W_p^{(2)}$ , which violates the definition of the modified weight for leaf nodes.

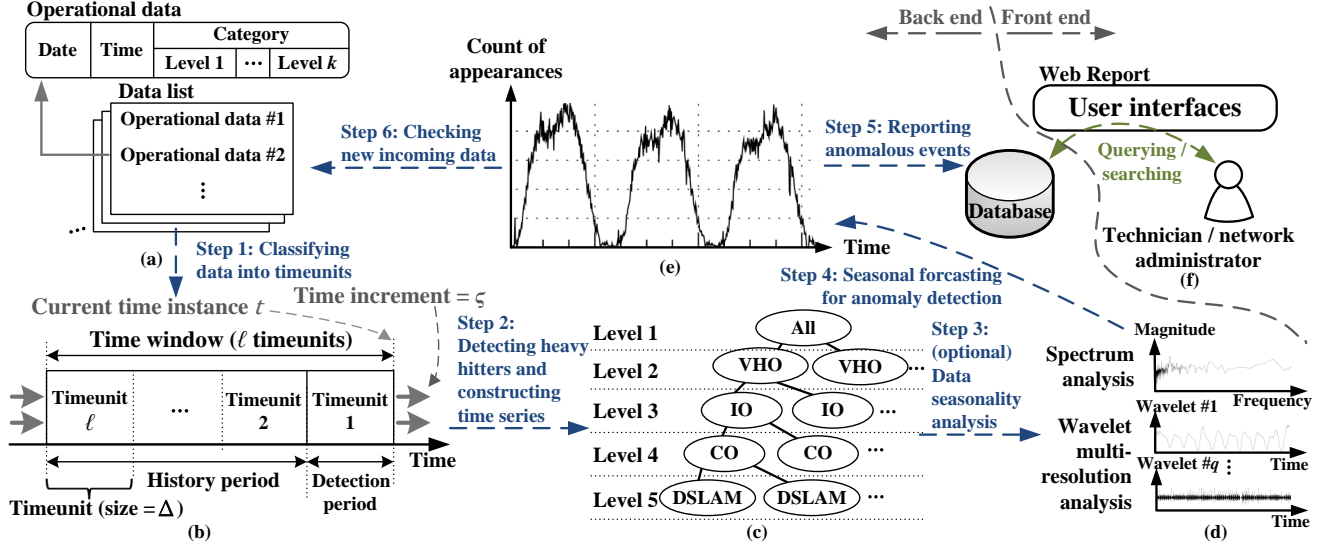


Figure 3: System diagram: (a) abstraction of streaming operational data, (b) a window-based structure to classify input data into timeunits, (c) heavy hitter detection and time series construction in a hierarchical domain, (d) data seasonality analysis, (e) seasonal forecasting model for anomaly detection, and (f) a front-end system provides user interfaces to query anomalous events. For details, refer to §4.1.

DEFINITION 4. (Anomaly). Given a time series  $T[n, t]$  of any node  $n$ ,  $1 \leq t \leq \ell$ , the forecast traffic of the latest time period  $T[n, 1]$  is given by  $F[n, 1]$ , deriving by some forecasting model. Then there is an anomalous event happening at node  $n$  in the latest time period iff  $T[n, 1]/F[n, 1] > RT$  and  $T[n, 1] - F[n, 1] > DT$ , where  $RT$  and  $DT$  are sensitivity thresholds<sup>2</sup>.

## 4. TIRESIAS SYSTEM OVERVIEW

We describe the building blocks of Tiresias (§4.1) and its limitations (§4.2).

### 4.1 System Components

We present our system diagram in Fig. 3. For each successive time period we receive a data list consisting of a set of newly arrived operational data (Fig. 3(a)).

**Step 1:** We maintain a sliding time window to group the input streaming data into “timeunits” based on its time information  $t_i$  (Fig. 3(b)). In particular, a time window consists of  $\ell$  units, each of which is of size  $\Delta$ . When new data lists arrived, we simply *shift* the time instance / window (by a configurable offset  $\varsigma$ ) to keep track of newly available data. For each time instance / window, we focus on detecting anomalous event in the *last* timeunit (the detection period in Fig. 3(b)), while the data in other timeunits will be used as the history for forecasting (the history period in Fig. 3(b)).

**Step 2:** Based on the category information  $k_i$  on the data, we construct a classification tree (Fig. 3(c)), where each category  $k_i$  can be bijectively mapping to a *leaf node* in the tree. As nodes associated with higher ag-

gregated data counts in the last timeunit are suspicious about places where anomaly happens, we first detect the heavy hitter set in the classification tree (Fig. 3(c); Definition 2). Then we construct time series for these heavy hitters (Definition 3). We will present efficient algorithms to achieve these goals (§5).

**Step 3:** We perform data seasonality analysis (Fig. 3(d)) to automatically choose the seasonal factors. In particular, we will apply fast fourier transform and à-trous wavelet transform [24] to identify significant seasonal factors (§6.1). The results are used as the parameters for anomaly detection in the next step.

**Step 4:** Given the time series and its seasonalities, we apply time series based anomaly detection (Fig. 3(e); Definition 4) on heavy hitter nodes. In particular, we will apply Holt-Winters’ seasonal forecasting model [5] to detect anomalies (§6.2).

**Step 5:** We report anomalous events to a database. A web-based front-end system (Fig. 3(f)) is developed in JavaScript to allow SQL queries from a text database.

**Step 6:** As we consider an online problem where the data arrives in a certain time period, we keep checking if new incoming data is arrived.

### 4.2 Limitations

Naturally, the power of Tiresias is limited by the operational data we have. We use customer care call dataset to illustrate some of the fundamental limitations.

First, Tiresias can only detect network incidents that directly or indirectly affect customers. For example, Tiresias cannot detect upticks in memory utilization on aggregate routers if it has no sensible impact to customers. This will require a more powerful detector which

<sup>2</sup>We conduct a sensitivity test to select thresholds (§7).

has access to runtime memory usage.

Second, it is difficult to verify whether the customer call cases have been miscategorized. For example, Tiresias cannot locate an anomaly correctly if a majority of customers provide incorrect information about their geographical location. To minimize the human error, well-trained service representatives will classify the problem based on the *predefined* categories. We believe that the selection is much less prone to error than mining natural language.

Third, Tiresias can only process input data which is drawn from an *additive hierarchy*. We limit our focus on hierarchical domains as it reflects the nature of the operational datasets we had in practice (§2.1). The more general data representation (e.g., directed acyclic graph) is out scope of this paper and is left for future work.

Finally, Tiresias cannot detect anomalies where the data rate abruptly drops, for example, link outages in network traffic data [25]. We did not consider this particular type of anomalies because detecting spikes (i.e., unexpectedly increases) is much more interesting in operational data like customer care calls.

## 5. ONLINE HHH DETECTION AND TIME SERIES CONSTRUCTION

While we presented our system overview in the previous section, it is still unclear how to efficiently detect the heavy hitters and construct time series in an on-line fashion (Step 2 in Fig. 3). To motivate our design, we start by presenting a strawman proposal STA (§5.1). Since the heavy hitter set may vary over timeunits, STA reconstructs the time series for new heavy hitters by traversing each timeunit within the sliding window of history. This requires to maintain  $\ell$  trees at any time. Moreover, whenever the sliding window shifts forward, we need to traverse all of these  $\ell$  trees to construct time series for each heavy hitter. This is inefficient in time and space as the length of time window  $\ell$  (in terms of the timeunit size  $\Delta$ ) is usually a large number, resulting in an inefficient algorithm. While this overhead can be reduced by increasing  $\Delta$ , or reducing the amount of history, doing so also reduces the accuracy of inference. The typical value of  $\ell$  is 8,064 (a time window of 12 weeks with 15-minute timeunits). To solve this problem, we propose ADA, a novel algorithm which works by adaptively update previous time series' position in the hierarchy (§5.2). We will show the clear advantages of ADA when compared to STA (§7). Table 3 summarizes the notation.

### 5.1 A Strawman Proposal STA

We first describe a strawman algorithm STA (Fig. 4), which reuses the same data structure in different time instances, i.e., we destroy a tree iff it will not be used in the next time instance. While in the first iteration

Variable	Definition
$\theta$	threshold of heavy hitter (Def. 1 and Def. 2)
SHHH	set of succinct hierarchical heavy hitters (Def. 2)
$\Delta$	timeunit size (Def. 3)
$\varsigma$	time increment of time window (§3)
$\ell$	length of time window (§5)
$n_r$	root node (Fig. 5)
$S$	input stream (§3)
$RT$ and $DT$	sensitivity thresholds (Def. 4)
$\lambda$	time scale base (§ 5.2.6)
$\eta$	time scale exponent (§ 5.2.6)
$T[\cdot]$ and $F[\cdot]$	time series and forecast series (Def. 3)

Table 3: Summary of notation

```

1 for time instance  $t = t_1, t_1 + \Delta, t_1 + 2\Delta, \dots$  do
2   if  $(t == t_1)$  then  $\kappa = \ell$ ; else  $\kappa = 1$ ;
3   foreach timeunit  $1 \leq i \leq \kappa$  do
4     Read any data item within time period
        $[t - i\Delta, t - i\Delta + \Delta)$  from the input stream to construct
       a tree  $\text{Tree}[t - i\Delta]$  such that each node in the tree is
       associated with a count of appearances; Notice that we
       use  $\text{Tree}[\tau]$  to represent a tree constructed by the data
       within time period  $(\tau, \tau + \Delta)$ .
5   end
6   Do a bottom-up traversal on  $\text{Tree}[t - \Delta]$  to derive the
   SHHH $[\theta, t - \Delta]$ ;
7   foreach timeunit  $1 \leq i \leq \ell$  do
8     Given the fixed heavy hitter set SHHH $[\theta, t - \Delta]$ , do a
       bottom-up traversal on  $\text{Tree}[t - i\Delta]$  to compute time
       series  $T[n, i]$  for each node  $n \in \text{SHHH}[\theta, t - \Delta]$ ;
9   end
10 end

```

Figure 4: Strawman Algorithm STA

(i.e.,  $t = t_1$ ) we construct  $\ell$  trees, the time spent on the first time instance can be amortized to other time instances where we construct only one tree. However, the time complexity of STA is dominated by time series construction (lines 7-9). In particular, for each time instance we need to traverse  $\ell$  trees to construct time series for each heavy hitter.

### 5.2 A Low-Complexity Adaptive Scheme

In this section, we propose ADA, an adaptive scheme that greatly reduces the running time while simultaneously retaining high accuracy. First, we present the data structure and the pseudocode (§5.2.1) for ADA. In particular, ADA maintains only *one* tree (rather than  $\ell$  trees in STA), and only those heavy hitter nodes in the tree associated with a time series. However, the key challenge is that heavy hitters changes over time – we need efficient operations (instead of traversing  $\ell$  trees) to derive the time series for heavy hitters nodes in current time instance. To this end, we construct time series by *adapting* the time series generated in the previous time instance (§5.2.2). For example, consider a tree with only three nodes: a root node and its two children. Suppose the root node is a heavy hitter only in the current time instance, while the two children are heavy hitters only in the last time instance. In this case, we can derive root node's time series by merging time series from its two children. Although the adaptation process is very intuitive in the above example, the problem becomes more complicated as tree height increases. We will show the proposed algorithm always finds the correct heavy hitter set (§5.2.3). While the positions of heavy hitters are perfectly correct, the corresponding time series might be

biased after performing *split* function. To address this problem, we will present two heuristics (§5.2.4–§5.2.5) to improve the accuracy.

Without loss of generality we assume that the time instance will keep increasing by a constant  $\varsigma \leq \Delta$ , where  $\Delta$  is the timeunit size. We can always map the problem with time increment  $\varsigma > \Delta$  to another equivalent problem with using a smaller time increment  $\varsigma' \leq \Delta$  such that  $\varsigma'$  is a divisor of  $\varsigma$ . For simplicity, we describe our algorithm by assuming  $\varsigma = \Delta$ . We then extend to consider any  $\varsigma \leq \Delta$  where  $\varsigma$  is a divisor of  $\Delta$  (§5.2.6).

### 5.2.1 Data structure and Pseudocode

We maintain a tree-based data structure as follows. Each node  $n$  in the tree is associated with a type field  $n.type$ . Fields  $n.child[i]$  and  $n.parent$  point to the  $i$ -th child and parent of the node,  $n.depth$  represents the depth of the node, and  $n.weight$  maintains the modified weight of the node (after discounting the weight from other descendants that are themselves heavy hitters; Definition 2). Array  $n.actual$  stores the time series of the modified weights for the node  $n$ , and Array  $n.forecast$  stores the time series of the forecasted values for the node  $n$ .

```
typedef struct {
    string      type;           // the type of this node
    vector<node*> child;       // child[i] points to the i-th child
    node*      parent;        // points to the parent
    int        depth;         // the depth of this node
    int        weight;        // the weight after discounting
    bool       ishh;          // this node is a heavy hitter
    bool       washh;         // this node was a heavy hitter
                                // in the previous time instance
    bool       tosplit;       // the node will "split" if possible
    vector<vol_type> actual;   // the time series of this node
    vector<vol_type> forecast; // the forecast time series of this node
} node;
```

Now we describe the core algorithm of ADA (Fig. 5). We maintain a tree structure *Tree* and a set of heavy hitter nodes, denoted by *SHHH* over time instances. For the first time instance (lines 2-5), ADA performs the same operations as STA. However, starting from the second time instance (lines 6-29), ADA consists of the following three stages.

**Initialization** (lines 6-12): We reset variables  $n.washh$ ,  $n.ishh$ , and  $n.weight$ . Subsequently, we recursively call a function *Update-Ishh-and-Weight*( $n$ ) (Fig. 6) to calculate the  $n.ishh$  and  $n.weight$  according to the Definition 2.

**SHHH and Time Series Adaptations** (lines 13-25): Unlike STA, where we explicitly update *SHHH* using a bottom-up traversal, in this stage we present a novel way to derive the set of *SHHH*. We first perform an *inverse level order traversal* (bottom-up) to check which node should “split” its time series to the children ( $n.tosplit$ ). Then we perform a top-down level order traversal to split (§5.2.2) the time series of any node  $n$  with  $n.tosplit$  to its children. We then do a bottom-up level order traversal to check if we should “merge” (§5.2.2) the

```
1 for time instance  $t = t_1, t_1 + \Delta, t_1 + 2\Delta, \dots$  do
2   if ( $t == t_1$ ) then
3     Perform the same operations as lines 2-9 in STA. This
       gives us a tree Tree and a heavy hitter set SHHH; We
       set  $n.ishh$  by true if  $n \in SHHH$ . Otherwise, we set
        $n.ishh$  by false. Each heavy hitter node  $n \in SHHH$  also
       has a time series of actual values  $n.actual$  and a time
       series of forecast values  $n.forecast$ ;
       continue;
4   end
5   foreach node  $n \in Tree$  do
6      $n.washh \leftarrow n.ishh$ ;  $n.weight \leftarrow 0$ ;  $n.tosplit \leftarrow 0$ ;
7   end
8   foreach streaming item ( $key, time$ ) such that
        $t - \Delta \leq time < t$  do
9     Find the node  $n$  such that  $n.type == key$ . Increase
        $n.weight$  by 1;
10  end
11  Update-Ishh-and-Weight( $n_{root}$ ); // Update  $n.ishh$  and
        $n.weight$ 
12  foreach node  $n \in Tree$  (bottom-up level order traversal) do
13    if ( $n.ishh \vee n.tosplit \wedge n \notin SHHH$ ) then
14       $n.parent.tosplit \leftarrow true$ ;
15    end
16  end
17  foreach node  $n \in Tree$  (top-down level order traversal) do
18    if ( $(n \in SHHH \vee n.depth == 1) \wedge n.tosplit$ ) then
19      SPLIT( $n$ );
20  end
21  foreach node  $n \in Tree$  (bottom-up level order traversal) do
22    if ( $n \in SHHH \wedge \neg n.ishh$ ) then MERGE( $n$ );
23  end
24  if ( $n_{root}.weight < \theta$ ) then  $SHHH \leftarrow SHHH \setminus n_{root}$ ;
25  else  $SHHH \leftarrow SHHH \cup n_{root}$ ;
26  foreach node  $n \in SHHH$  do
27    Append  $n.weight$  to  $n.actual$ ; Remove the first element
       from  $n.actual$  and that from  $n.forecast$ ;
28    Derive the newest forecast value and append it to
        $n.forecast$ ;
29  end
30 end
```

**Figure 5: Core Algorithm of ADA.** This algorithm calls three functions: *Update-Ishh-and-Weight* (Fig. 6), *SPLIT* (Fig. 7) and *MERGE* (Fig. 8).

```
int Update-Ishh-and-Weight(node  $n$ )
1 foreach node  $n_c = n.child$  do
2    $n.weight \leftarrow n.weight + Update-Ishh-and-Weight(n_c)$ ;
3 if ( $n.weight \geq \theta$ ) then
4    $n.ishh \leftarrow 1$ ; return 0;
5 else
6    $n.ishh \leftarrow 0$ ; return  $n.weight$ ;
```

**Figure 6: Updating fields  $n.ishh$  and  $n.weight$ .** This function recursively updates  $n.ishh$  and  $n.weight$  in a bottom-up fashion

```
SPLIT(node  $n$ )
1  $C_n \leftarrow \{n_c | n_c \in n.children[\cdot], n_c \notin SHHH\}$ ;
2 if ( $\exists n_c \in C_n$  s.t.  $n_c.weight \geq \theta$ ) then
3   foreach node  $n_c \in C_n$  do
4     Split the time series from  $n$  to  $n_c$  with a scale ratio of
        $F(n_c, C_n)$ ; (The function  $F(\cdot)$  is defined in §5.2.4)
5   Clear  $n.weight$ ;
6    $SHHH \leftarrow (SHHH \cup C_n) \setminus \{n\}$ ;
```

**Figure 7: Splitting time series to children**

```
MERGE(node  $n$ )
1 if ( $n.weight < \theta$ ) then
2    $n_p \leftarrow n.parent$ ;  $C_n \leftarrow \{n_c | n_c \in \{n_p\} \cup n_p.children[\cdot], n_c \in SHHH, n_c.weight < \theta\}$ ;
3    $n_p.weight \leftarrow \sum_{n_c \in C_n} n_c.weight$ ;
4    $SHHH \leftarrow (SHHH \setminus C_n) \cup \{n_p\}$ ;
5   Merge the time series from every  $n_c \in C_n$  to  $n$ ;
```

**Figure 8: Merging time series to parent**

time series from some neighboring nodes to the parent. Throughout these split and merge operations, we keep updating SHHH so that any node  $n \in \text{SHHH}$  iff  $n$  has time series  $n.\text{actual}$  and  $n.\text{forecast}$  (except for the root node). Finally, we simply add/remove the root node to/from SHHH based on its weight. We will show that SHHH satisfies the Definition 2 after performing this stage (§5.2.3).

**Time Series Update** (lines 26-29). We update the time series for each heavy hitter node in SHHH. Unlike STA, the time series can be updated in constant time.

### 5.2.2 Split and Merge

This section presents two functions, split and merge, to *adapt* the position of heavy hitter nodes. Before the adaptation, heavy hitter nodes are placed in the location given in the previous timeunit. The goal is to move these heavy hitter nodes (with the binding time series and forecast) to their new positions according to the data in the current timeunit, to satisfy the Definition 2.

We leverage a *Split*( $n$ ) function (Fig. 7) to linearly decompose the time series from node  $n$  into its non heavy hitter children. In other words, a heavy hitter node  $n$  will be split into one or more heavy hitters in one level down. In this function, we first check if there exists any child node  $n_c \notin \text{SHHH}$  with weight  $\geq \theta$ . If so, we split the time series of  $n$  with a scale ratio of  $F(n_c, C_n)$  for each non-heavy-hitter-child  $n_c$  to approximate the time series of its children. We will discuss different heuristics to estimate scale ratio  $F(n_c, C_n)$  (§5.2.4).

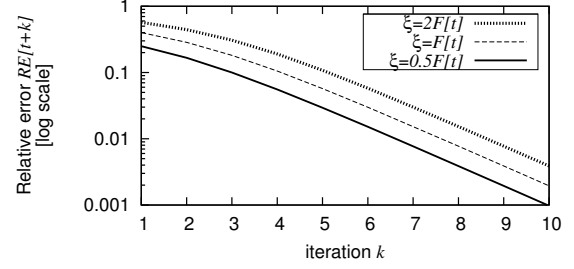
We present a *Merge*( $n$ ) function (Fig. 8) to combine the time series from one or more lower level nodes to a higher level node. In this function, we first check if the weight of the node  $n$  is smaller than  $\theta$ . If so, we merge the time series of  $n$  and its siblings and parent who meet the same requirement as  $n$ .

### 5.2.3 Correctness of Heavy Hitter Set

It suffices to show that Tiresias always selects the correct heavy hitter set after performing the stage “SHHH and Time Series Adaptation” (lines 13-21), i.e., Definition 2 will hold. The high-level intuition is that top-down split operations ensure that there is no heavy hitter hidden in the lower levels, while bottom-up merge operations ensure all non-heavy-hitter nodes propagate the weight to its heavy hitter ancestor correctly. This argument clearly holds for the first time instance because there is no adaptation. We show it by induction on time instance, and the proof is given in Appendix.

### 5.2.4 Split Rules and Error Estimation

Recall that in *SPLIT*( $n$ ) function (Fig. 7), we split the time series from node  $n$  to each non-heavy-hitter children  $n_c \in C_n$  with a scale ratio of  $F(n_c, C_n)$ , where  $C_n = \{n_c | n_c \in n.\text{children}[\cdot], n_c \notin \text{SHHH}\}$ . The scale



**Figure 9: Relative error  $RE[t+k]$  after  $k$  iterations. (y-axis is  $\log_{10}$  scale;  $\alpha = 0.5$ ;  $T[i] = 1$ )**

ratio is derived by

$$F(n_c, C_n) = \frac{X_{n_c}}{\sum_{m \in C_n} X_m} \quad (1)$$

where  $X_n$  is a weight-related property of node  $n$ . Below we consider four split rules for deriving the scale ratio.

**Uniform:** each element in the time series of  $n$  is uniformly split to all its children  $n_c \notin \text{SHHH}$ , i.e.,  $X_n = 1$ .

**Last-Time-Unit:** we assign  $X_n$  by the weight of node  $n$  in the previous timeunit. This assumes the weight is similar in the current timeunit and previous one.

**Long-Term-History:** we assign  $X_n$  by the total weight seen by the node across previous timeunits.

**EWMA:** The **Last-Time-Unit** only captures the distribution in the last timeunit, while the **Long-Term-History** treats past observations equally. This heuristic takes a different approach – the  $X_n$  is assigned by an exponentially smoothed weight.

We next discuss the estimation error of the “split” operation over time. For simplicity, we illustrate this by considering a EWMA-based forecast model  $F[t] = \alpha T[i-1] + (1-\alpha)F[t-1]$ , where  $T[t]$  is the actual time series. Suppose we did a split for any time series  $T$  at time  $t$ . Also, suppose the forecast  $F[t]$  is biased by  $\xi$ , i.e., the estimated forecast  $F_E[t] = F[t] + \xi$ . Then after  $k$  iterations, the estimated forecast can be derived by

$$F_E[t+k] = \alpha \left\{ \sum_{j=1}^k (1-\alpha)^{j-1} T[t+k-j] \right\} + (1-\alpha)^{k-1} (F[t] + \xi) \quad (2)$$

for any  $k > 0$ . Then we have relative error

$$RE[t+k] = \frac{|F_E[t+k] - F[t+k]|}{F[t+k]} \quad (3)$$

As the proposed algorithm derive the forecast value in a recursive way, the inaccuracy will remain in the future forecast. However, the forecast error will exponentially decrease over time, and this is evident in Figure 9.

### 5.2.5 Reference Time Series

To further reduce the time series errors induced by the split operations, we propose a simple add-on to ADA by adding a set of *reference time series*. Con-

sider a node  $n$  with category  $k$  in the hierarchical domain, the reference time series are time series of its *unmodified* weight  $A_n[k, t]$  (instead of  $W_n[k, t]$  which discounts weights from heavy hitter descendants) and the time series of its forecast values. Now we illustrate how to use reference time series to reduce the estimation error induced by the split operations. Suppose a node  $n$  splits the time series into its children using some split rule (§5.2.4). As split operations can be inaccurate, we assume that a children node  $n_c$  received a “biased” time series  $T[n_c, i]$ ,  $1 \leq i \leq \ell$ . If  $n_c$  has a reference time series  $T_{\text{REF}}[n_c, i]$ , we will replace the  $T[n_c, i]$  by  $T_{\text{REF}}[n_c, i] - \sum_{n^* \in \mathbf{N}^*} T[n^*, i]$ ,  $1 \leq i \leq \ell$ , where  $\mathbf{N}^*$  is the set of  $n_c$ ’s heavy hitter descendants, i.e.,  $\mathbf{N}^* = \{n^* | n^* \in \text{SHHH} \wedge n^* \in \text{desc}(n_c)\}$ .

We maintain reference time series only for nodes in the top  $h$  levels in the hierarchy. We will discuss the memory and detection accuracy tradeoffs (§7.1).

### 5.2.6 Multiple Time Scales

We now show that ADA can support any case where  $\Delta$  is a multiple of  $\varsigma$  by generalizing ADA to support multiple time scales: consider a vector of  $\eta$  time scales where the  $i$ -th time scale is  $\lambda^{i-1}\Delta$ ,  $1 \leq i \leq \eta$  for any two given positive integers  $\lambda$  and  $\eta$ .

To this end, we need to maintain time series in multiple time scales. We use two-dimension dynamic arrays (i.e., `vector<vector<vol_type>>`) for fields `n.actual` and `n.forecast`, where the first dimension represents time scales. Recall that we append the new value to the tail of time series in the single time scale case (line 27 in Fig. 5). To consider multiple timescales, we defined a recursive function `UPDATE_TS( $n, w, i$ )` to append a new value  $w$  to the time series of node  $n$  for the  $i$ -th time scale (Fig. 10). In particular, when the size of any time series in the  $i$ -th time scale is a multiple of  $\lambda$ , we sum up the last  $\lambda$  elements and recursively update the weight to the  $(i + 1)$ -th time scale. It is simple to see that `UPDATE_TS( $n, w, i$ )` will be called for every  $\lambda^i$  timeunits, for  $1 \leq i \leq \eta$ . Then for  $\kappa$  timeunits, we will call `UPDATE_TS( $\cdot$ )`  $\sum_{i=1}^{\eta} \kappa/\lambda^i \leq 2\kappa$  times. Therefore, the update operation remains  $\Theta(1)$  amortized time for each time instance.

With multiple time scales, it suffices to show that ADA can support any case where  $\Delta$  is a multiple of  $\varsigma$ . Consider a problem  $P_A$  with parameters  $\Delta_A$  and  $\varsigma_A$ . It is not hard to see that  $P_A$  is equivalent to another problem  $P_B$  with parameters  $\Delta_B = \varsigma_B = \varsigma_A$  by assigning  $\lambda_B = \Delta_A/\varsigma_A$  and  $\eta_B = 1$ .

## 6. TIME SERIES ANALYSIS

As we observed strong periodicities in the data sets, simple forecasting models like exponential weighted moving average (EWMA) [11] will be very inaccurate. To this end, we first apply time series analysis techniques to quantify the data seasonality (Step 3 in Fig. 3; pre-

```

UPDATE_TS(node  $n$ , int  $w$ , int  $i$ )
1  $n$ .forecast[ $i$ ].push_back( $\alpha \times w + (1 - \alpha) \times n$ .forecast[ $i$ ].last());
2  $n$ .actual[ $i$ ].push_back( $w$ );
3  $s \leftarrow n$ .actual[ $i$ ].size();
4 if ( $i < \eta \wedge s = 0 \pmod{\lambda}$ ) then
5    $w' \leftarrow \sum_{1 \leq j \leq \lambda} n$ .actual[ $i$ ][ $s - j$ ];
6   UPDATE_TS( $n$ ,  $w'$ ,  $i + 1$ );
7   if ( $s = \ell + \lambda$ ) then
8     Do  $n$ .actual[ $i$ ].pop_head()  $\lambda$  times.

```

**Figure 10: Update time series in multiple time scales**

sented in §6.1). The results will be used as the parameters in the forecasting model. For time series based anomaly detection, we choose additive Holt-Winters’ seasonal forecasting model, which captures the data seasonalities while preserving low overhead (Step 4 in Fig. 3; presented in §6.2).

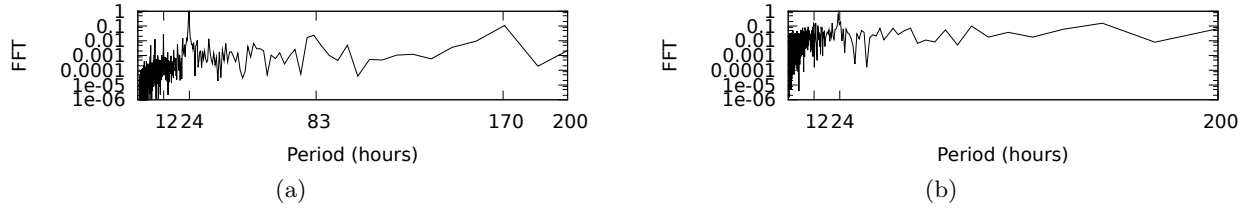
### 6.1 Data Seasonality

To study the seasonality, we apply Fast Fourier Transform (FFT) to obtain the count of cases in frequency spectrum (Fig. 11). We observe that the most significant period is 24 hours (i.e., day period) in the both datasets, while 170 hours (i.e., close to the weekly period) is also noticeable in CCD. To validate this, we also perform the *à-trous wavelet transform* [24] to convert the time series of our datasets to “smoothed” approximations at different timescales. To avoid phase shifting, we use the low-pass  $B_3$  spline filter ( $\frac{1}{16}, \frac{1}{4}, \frac{3}{8}, \frac{1}{4}, \frac{1}{16}$ ) as in previous work done on time series smoothing [20]. Given the approximated time series  $c_j(t)$  at timescale  $j$ , the *detail* of  $c(t)$  at time scale  $t$  can be derived by  $d_j(t) = c_{j-1}(t) - c_j(t)$ . Then we have the *energy* of the detail signal  $d_j(t) = \sum_{\forall t} d_j(t)$ , which indicates the strength of the fluctuations at timescale  $j$ . For both datasets, we apply this to multiple timescales (ranging from 5 seconds to 4 weeks), and the significant periodicities we found are consistent with that given by FFT. The results clearly suggest us using time series model with *multiple seasonalities* for CCD.

We found that the periodicities of operational datasets we had are fairly stable across time. Therefore, in our current implementation we perform the data seasonality analysis in an offline fashion, i.e., only in the first time instance.

### 6.2 Data Forecasting for Anomaly Detection

To distinguish anomalous behavior, we characterize typical behavior by applying forecasting model on data time series. As we observed strong periodicities in the data sets, simple forecasting models like exponential weighted moving average (EWMA) [11] will be very inaccurate. To address this, we adopt the additive Holt-Winters’ seasonal forecasting model [5], which decouples a time series  $T[t]$  into three factors: level  $L[t]$ , trend  $B[t]$



**Figure 11: The Fast Fourier Transform for the datasets (a) CCD and (b) SCD. The log-scale y axis is normalized by the maximal magnitude.**

and seasonal  $S[t]$ :

$$\begin{aligned} L[t] &= \alpha(T[t] - S[t - v]) + (1 - \alpha)(L[t - 1] + B[t - 1]) \\ B[t] &= \beta(L[t] - L[t - 1]) + (1 - \beta)(B[t - 1]) \\ S[t] &= \gamma(T[t] - L[t]) + (1 - \gamma)(S[t - v]) \end{aligned}$$

where the season length is denoted by  $v$ . Then the forecast  $G[t]$  is simply adding up these three factors, i.e.,  $G[t] = L[t - 1] + B[t - 1] + S[t - v]$ . To initialize the seasonality, we assume there are at least two seasonal cycles  $T[t - 1], T[t - 2], \dots, T[t - 2v]$ . The starting values  $L[t - 1], B[t - 1]$  and  $S[t - 1], \dots, S[t - 2v]$  can be derived by

$$\begin{aligned} L[t - 1] &= \frac{1}{2v} \sum_{j=t-1}^{t-2v} T[j] \\ B[t - 1] &= \frac{1}{2v} \left[ \left( \sum_{j=t-1}^v T[j] \right) - \left( \sum_{j=t-v-1}^{2v} T[j] \right) \right] \\ S[t - j] &= T[t - j] - L[t - 1], j = 1, \dots, 2v \end{aligned}$$

The use of Holt-Winters’ seasonal model does not compromise the update cost. Given the factors in the previous timeunits, the forecast can be computed in constant time. Moreover, we choose the additive Holt-Winters’ seasonal model because of its linearity – there is no need to recalculate the forecast value after merge / split. The proof is available in [1].

## 7. EVALUATION

To evaluate the performance of our algorithms, we build a C++ implementation of Tiresias, including ADA and STA. All our experimental results were conducted on a Solaris cluster with 2.4GHz processors using single core. In our evaluation, we first study Tiresias’s system performance (§7.1). Comparing it with the strawman proposal STA, the adaptive algorithm ADA provides (i) lower running time (by a factor of 14.2), (ii) lower memory requirement (by a factor of 2), (iii) time series with <1% absolute error, and (iv) identified anomalies with 99.7% accuracy. We then compare Tiresias with a current best practice used by an ISP operational team (§7.2). Tiresias not only detects >94% of the anomalies found by the reference method but also many previously unknown anomalies hidden in the lower levels.

**System parameters:** For SCD, we use the Holt-

Winters’ seasonal model with single seasonal factor  $S_{day}[t]$ .

To capture both diurnal and weekly patterns observed in CCD, we consider *two* seasonal factors  $S_{day}[t]$  and  $S_{week}[t]$  for CCD. To incorporate these two seasonal factors in the Holt-Winters’ model, we linearly combine the factors, i.e., the seasonal factor is defined as  $S = \xi S_{day}[t] + (1 - \xi) S_{week}[t]$ . We choose the weight  $\xi = FFT_{day} / FFT_{week} = 0.76$  according to the measured magnitude  $FFT_t$  of frequency  $1/t$  (§6.1). To find the other parameters for the Holt-Winters’ forecasting model ( $\alpha, \beta$  and  $\gamma$  in §6.2), we measure the mean squared error of the forecast values ( $\propto \sum (actual - forecast)^2$ ) to choose the parameters in an offline fashion. The parameters with the minimal mean squared error are chosen for the evaluation throughout this section. We choose a small heavy hitter threshold  $\theta$ , which gives us around 125 (5) heavy hitters in the “busy” (“quiet”) period in CCD, and 500 heavy hitters in SCD. We perform a sensitivity test and chose sensitivity thresholds  $RT = 2.8$  and  $DT = 8$ , which gave reasonable performance in comparison with the reference method (§7.2).

## 7.1 System Performance

We first evaluate the performance of Tiresias using algorithms ADA and STA. We first present the results for CCD, and we summarize the results from SCD.

**Computational Time:** We use Long-Term-History as a representative heuristic as we observed very similar running time for these heuristics. In Table 4, we summarize the running time of Tiresias using CCD in May 2010. We observe that ADA with 15-minute (1-hour) timeunits reduces the overall running time by a factor of 14.2 (5.4) as compared with STA. If we subtract the running time for reading traces (which is arguably necessary for any algorithm), we observed a factor of 49.5 (41.3) improvement. To better understand why ADA has better running time, we divide the program into four stages: Reading Traces, Updating Hierarchies, Creating Time Series and Detecting Anomalies. Among these stages, we observed that Creating Time Series is clearly a bottleneck in STA, which contributes 83.1% (93.9%) time of the total time of all stages for timeunits of size 15-minute (1-hour). As the time complexity of this stage for STA is proportional to  $\ell$ , we expect the running time of this stage would be inversely pro-

Timeunit size ( $\Delta$ )		15 Minutes		1 Hour	
Algorithm		ADA	STA	ADA	STA
Reading Traces	Mean	18.6 (74.0%)	19.592 (5.4%)	19.096 (89.0%)	18.885 (16.2%)
	Variance	12.493	11.493	15.802	12.801
Updating Hierarchies	Mean	4.762 (18.9%)	0.198 (0.1%)	1.699 (7.9%)	0.273 (0.2%)
	Variance	1.082	0.082	0.287	0.116
Creating Time Series	Mean		339.216 (93.9%)		96.943 (83.1%)
	Variance		83.114		86.518
Detecting Anomalies	Mean	1.785 (7.1%)	2.232 (0.6%)	0.67 (3.1%)	0.546 (0.5%)
	Variance	0.912	2.311	0.373	0.483
Sum		25.507 (100.0%)	361.238 (100.0%)	21.465 (100.0%)	116.647 (100.0%)

**Table 4: Summary of running time of Tiresias in minutes with different algorithms. The running time for STA is 5-14 times more than that for ADA. The gap increases with the decrease of timeunit size.**

portional to the timeunit size<sup>3</sup>. Fortunately, we also observe that the “per timeunit” running time of this stage with 15-minute timeunits is slightly smaller (with a factor of 0.87) than that with 1-hour time units. This is because the trace we used is sparse, and therefore we have a smaller tree size when using smaller timeunits.

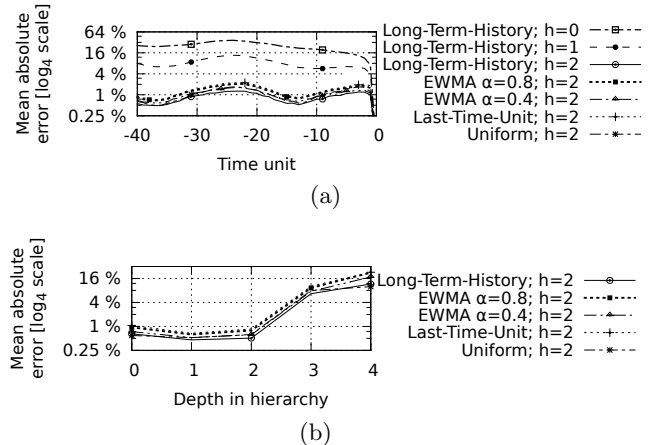
**Memory Requirement:** We evaluate memory costs for ADA (again, different split rules have very similar memory costs) and STA (Table 5). To eliminate cold-start effects, we focus on measuring space costs after a long run. We observe the memory cost of ADA is  $\approx 36\%$  of that of STA. We maintain the reference time series for any node in the top  $h$  levels in the hierarchy (the root level is not included). With two levels of reference nodes, the memory cost of ADA is only  $\approx 43\%$  of that of STA.

Algorithm	# ref. levels ( $h$ )	Normalized Space
STA	N/A	744.3
ADA	0	271.1
	1	280.2
	2	322.6

**Table 5: Normalized memory costs of Tiresias with different algorithms. (Normalized memory cost = total memory cost / average number of nodes in the tree / the memory cost of each node.)**

**Time Series Accuracy:** As the split operations might introduce inaccuracy, we next examine how much distortion of time series we have. We conduct the experiment by leveraging the results generated by STA as the ground truth. In particular, for each timeunit  $t$  in any time series  $T[t]$ , the absolute error is derived by  $|T_{ADA}[t] - T_{STA}[t]|$ . We first consider the impact of adding reference time series (Fig. 12). We observe the use of the reference time series significantly reduces the error rate. Consider Tiresias with reference time series for nodes in the top  $h = (1, 2, 3, 4)$  levels, we have (61, 322, 1926, 45479) reference time series, respectively. We observe that a large number of reference time series is not necessary as two layers of reference time series ( $h = 2$ ) already gives us a very accurate time series estimation (Fig. 12). For  $h = 2$ , we maintain reference time series for only 322/45479  $\approx 0.7\%$  of total nodes. More-

<sup>3</sup>For a time series with fixed length, we have  $\ell = 1/\Delta$ .



**Figure 12: The absolute error of time series for different heuristics versus (a) timeunits and (b) depths. The error is averaged over different timeunits in the time series and over the time series of heavy hitters. The reference time series are not included. The timeunit 0 is the most recent time for any time instance. We use  $h$  and  $\alpha$  to denote the number of reference levels (§5.2.5) and the smoothing rate of EWMA, respectively.**

over, we compare the accuracy for different split rules. We observe that only Long-Term-History have slightly better accuracy, while the accuracy results of the other heuristics are fairly close. Besides, we observe the accuracy is very stable over different timeunits. While for brevity we did not present the accuracy results for every timeunit in the figure, we observe very similar results for older timeunits.

**Anomaly Detection Accuracy:** To quantify the inaccuracy introduced by split operation, we compare the anomalies detected by ADA and STA. We use STA as the ground truth as it reconstructs accurate time series for each time instance. In Table 6, we summarize the accuracy metrics for different heuristics of ADA. The results are collected across 100 time instances between May 16-21, 2010. Overall, we found that ADA was able to detect anomalies with an accuracy of 97 – 99% over a variety of heuristics. However, there is some tradeoff between split rules. We observe that EWMA (with rate  $\alpha = 0.4$ ) has the highest precision, while Uniform has the best result on sensitivity. We also observe Long-Term-History has good results on all metrics under investigation.

Split rule	# ref levels ( $h$ )	Accuracy	Precision	Sensitivity	Specificity
		$= \frac{\#TPs + \#TNs}{\#Cases}$	$= \frac{\#TPs}{\#TPs + \#FPs}$	$= \frac{\#TPs}{\#TPs + \#FNs}$	$= \frac{\#TNs}{\#TNs + \#FPs}$
Long-Term-History	0	97.2%	37.8%	41.8%	98.5%
	1	98.6%	78.6%	49.3%	99.7%
	2	99.6%	94.4%	88.1%	99.9%
EWMA (rate = 0.8)	2	99.6%	96.5%	82.8%	99.9%
EWMA (rate = 0.6)	2	99.6%	96.6%	84.3%	99.9%
EWMA (rate = 0.4)	2	99.7%	96.7%	87.3%	99.9%
Last-Time-Unit	2	99.6%	96.5%	82.8%	99.9%
Uniform	2	99.3%	79.2%	94.0%	99.5%

Table 6: Anomaly detection accuracy of Algorithm ADA (Compared with STA)

Performance metric	Value
Type 1 (Accuracy) $= \frac{\#TPs + \#TNs}{\#Cases}$	94.1%
Type 2 $= \frac{\#TPs}{\#TPs + \#MAs}$	90.9%
Type 3 $= \frac{\#TNs}{\#TNs + \#NAs}$	94.1%
Type 4 $= \frac{\#TPs}{\#TPs + \#NAs}$	5.1%

Table 7: A comparison of ADA against the reference method

**Results for SCD:** Compared with CCD, it takes longer running time (with a factor of 3) for reading traces from SCD. Because of this, we observe the overall running time of ADA for SCD is slightly increased (with a factor of 1.3). However, the overall running time of STA for SCD is greatly increased by a factor of 7.4. In particular, the running time for Creating Time Series is increased by a factor of 6.4 because of the increase in the size of the hierarchy. Besides, we also observe an increased memory consumption for both ADA and STA in a factor of 2 because of the larger hierarchical space. However, with  $h = 0$  ( $h = 1$ ) level of reference nodes, the memory requirement of ADA is 43% (46%) of that of STA. For the time series accuracy of ADA, we observed an average absolute error of only 0.8% with  $h = 1$ . The primary reason of high accuracy for SCD is that the split operations are performed less frequently because of the smaller variance of the number of cases over time in SCD (Fig. 1(c)). Consequently, we observe a very good accuracy of anomaly detection for SCD. In particular, we have no false positive and very few false negatives (in only about 0.13% of all negative cases with  $h = 1$ ).

## 7.2 Comparison with Current Practice

To evaluate the performance of Tiresias in identifying live network problems, we compared the set of anomalies detected by our methods from CCD with a set of reference anomalies identified using an existing approach based on applying control charts to time series of aggregates at the first network level (the VHO level). This existing approach is currently used by an operational team of a major US broadband provider.

For the purposes of this study we treat the reference anomalies as the ground truth against which we evaluate the performance of our method. Unfortunately, the reference anomaly set is *not complete* – it only looks at the anomalies happened in the first network level (VHO). Because of this, we cannot simply use the standard terms {True|False} {Positive|Negative} in describing the anomalies found by our method relative to the

reference set. To undertake the comparison, we define the following comparison metrics. Let  $A_{OP}$  and  $A_{Tiresias}$  denote the set of reference anomalies and those found by Tiresias, respectively. Clearly, each anomaly  $a$  can be one-to-one mapping to a certain network location  $L(a)$  and timeunit  $T(a)$ . In particular, for each anomaly  $a_{OP} \in A_{OP}$ , we have a *true alarm* (TA) case if there exists an anomaly  $a_{Tiresias} \in A_{Tiresias}$  such that  $T(a_{OP}) = T(a_{Tiresias})$  and  $L(a_{OP}) \supseteq L(a_{Tiresias})$ , where  $L_1 \supseteq L_2$  iff  $L_1$  is equal to or an ancestor of  $L_2$ . In this case, Tiresias is able to locate the anomaly with finer granularity. For each anomaly in  $A_{OP}$ , we have a *missed anomaly* (MA) case if there does not exist any anomaly satisfying the above requirement. Moreover, let  $A_{Tiresias}^I \subset A_{Tiresias}$  denote the set of Tiresias’s anomalies which is not related to any anomaly  $a_{OP} \in A_{OP}$ . Specifically,  $A_{Tiresias}^I = \mathcal{G}(A_{Tiresias}, A_{OP}) = A_{Tiresias} \setminus \{a_{Tiresias} \in A_{Tiresias} | \exists a_{OP} \in A_{OP}, T(a_{OP}) = T(a_{Tiresias}), L(a_{OP}) \supseteq L(a_{Tiresias})\}$ . We say there is a *new anomaly* (NA) case for each element in  $A_{Tiresias}^I$ . Let  $\hat{A}_{Tiresias}$  denote the set of heavy hitters which are not reported as anomalies by Tiresias. For each element in  $\hat{A}_{Tiresias} \in \hat{A}_{Tiresias}$ , we have a *true negative* (TN) case if  $\hat{a}_{Tiresias}$  is unrelated to any anomaly in  $A_{OP}$ . In particular, the set of TN cases is defined by  $\mathcal{G}(\hat{A}_{Tiresias}, A_{OP})$ , where the function  $\mathcal{G}(\cdot)$  is defined in the NA case.

We compare the reported anomalies using CCD in the period of 14 – 24th, September 2010. In Table 7, we observe that ADA effectively detects most of the anomalies found by the reference method with an accuracy of 94%, i.e., we have a large number of true alarms and true negatives, while the number of missed anomaly is very small.

Furthermore, the results suggest that Tiresias is promising to find previous unknown anomalies. This is likely attributable to the fact that Tiresias is able to adapt heavy hitter position to detect anomalies below the first level, whereas the reference method is only able to trace the first network level nodes. We perform a simple data aggregation of the NAs to remove any redundant anomalies which are an ancestor of other anomalies. The fraction of the resulting cases in the (VHO, IO, CO, DSLAM) level is (5%, 56.3%, 29.3%, 9.4%). The majority (95%) of NA cases are localized below VHO level; These are inherently harder to detect with the reference dataset due to their VHO-level aggregation with non-anomalous call loads.

## 8. RELATED WORK

Given the increasing severity of DoS, scanning, and other malicious traffic, traffic anomaly detection in large networks is gaining increased attention. To address this, several works detect statistical variations in traffic matrices constructed using summarizations of traffic feature distributions [8, 10]. Statistical methods used in these works include sketch-based approach [13], ARIMA modeling [3], Kalman filter [26] and wavelet-based methods [2]. ASTUTE [25] was recently proposed to achieve high detection accuracy for correlated anomalous flows. Lakhina *et al.* [14, 15] showed that Principal Component Analysis (PCA) can improve scaling by reducing the dimensionality over which these statistical methods operate. Huang *et al.* [12] proposed a novel approximation scheme for PCA which greatly reduce the communication overheads in a distributed monitoring setting. More recently, Xu *et al.* [28] also applied PCA to mine abnormal feature vectors in console logs. However, it is unclear how to transform hierarchical domain in operational data to multiple dimensions for PCA. While entropy of data distribution might be a useful measure as suggested in previous studies [15, 19], Ringberg *et al.* [23] showed that the choice of aggregation can significantly impact the detection accuracy of PCA.

There are several works on hierarchical heavy hitter detection [7, 9, 17, 30]. Among these, the hierarchical heavy hitter detection scheme (HHD) [30] proposed by Zhang *et al.* is the closest in spirit to Tiresias. There are two key challenges prevent us to adopt this novel approach to our problem: **i)** HHD performs *offline* anomaly detection on identified heavy hitter nodes. To consider online anomaly detection, STA can be seen as a nature extension of HHD where the time series are re-computed from scratch in every timeunit. **ii)** HHD considers *cash-register* model [18], where the input stream elements cannot be deleted [7]. Because discounting old data is not allowed in the cash-register model, HHD is more useful to detect long-term heavy hitter over a coarser time granularity.

## 9. CONCLUSION

The ability to detect and identify network anomalies would benefit greatly from efficient analysis of operational data. Towards this goal, we develop Tiresias to detect anomalies designed toward the unique characteristics of operational data, including a novel adaptive scheme to accurately locate the anomalies using hierarchical heavy hitters. To the best of our knowledge, our design provides the first online mechanism for heavy hitter based anomaly detection. We present a measurement study on two large-scale operational datasets, as well as a performance evaluation of an implementation. Using the operational data collected at a tier-1 ISP, our evaluation suggests that Tiresias maintains accuracy and

computational efficiency that scales to large networks.

The current implementation serves as a promising building block for network anomaly detection in large network operations. To provide timely support to network operation of more localized network problems, we deploy our implementation inside a tier-1 ISP, and our major next step is to optimize system parameters based on the feedbacks from ISP operational teams.

## 10. REFERENCES

- [1] Proof of Lemma. <http://cyhong.projects.cs.illinois.edu/pub/proof.pdf>.
- [2] P. Barford and D. Plonka. Characteristics of network traffic flow anomalies. In *SIGCOMM Workshop on Internet Measurement*, 2001.
- [3] G. E. P. Box and G. Jenkins. *Time series analysis, forecasting and control*. Prentice Hall, 1994.
- [4] D. Brauckhoff, X. Dimitropoulos, A. Wagner, and K. Salamatian. Anomaly extraction in backbone networks using association rules. In *IMC*, 2009.
- [5] J. D. Brutlag. Aberrant behavior detection in time series for network monitoring. In *USENIX Conference on System Administration*, 2000.
- [6] D. R. Choffnes, F. E. Bustamante, and Z. Ge. Crowdsourcing service-level network event monitoring. In *SIGCOMM*, 2010.
- [7] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava. Diamond in the rough: Finding hierarchical heavy hitters in multi-dimensional data. In *SIGMOD*, 2004.
- [8] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. 2005.
- [9] C. Estan, S. Savage, and G. Varghese. Automatically inferring patterns of resource consumption in network traffic. In *SIGCOMM*, pages 137–148, New York, NY, USA, 2003. ACM.
- [10] C. Estan and G. Varghese. New directions in traffic measurement and accounting. In *SIGCOMM*, 2002.
- [11] C. C. Holt. Forecasting seasonals and trends by exponentially weighted moving averages. In *International Journal of Forecasting*, 2004.
- [12] L. Huang, X. Nguyen, M. Garofalakis, J. Hellerstein, A. D. Joseph, M. Jordan, and N. Taft. Communication-efficient online detection of network-wide anomalies. In *INFOCOM*, 2007.
- [13] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen. Sketch-based change detection: Methods, evaluation, and applications. In *IMC*, 2003.
- [14] A. Lakhina, M. Crovella, and C. Diot. Diagnosing network-wide traffic anomalies. In *SIGCOMM*, 2004.
- [15] A. Lakhina, M. Crovella, and C. Diot. Mining anomalies using traffic feature distributions. In

*SIGCOMM*, 2005.

- [16] A. Mahimkar, J. Yates, Y. Zhang, A. Shaikh, J. Wang, Z. Ge, and C. T. Ee. Troubleshooting chronic conditions in large ip networks. In *CoNEXT*, 2008.
- [17] A. A. Mahimkar, Z. Ge, A. Shaikh, J. Wang, J. Yates, Y. Zhang, and Q. Zhao. Towards automated performance diagnosis in a large IPTV network. In *SIGCOMM*, 2009.
- [18] S. Muthukrishnan. Data streams: Algorithms and applications. In *SODA*, 2003.
- [19] G. Nychis, V. Sekar, D. G. Andersen, H. Kim, and H. Zhang. An empirical evaluation of entropy-based traffic anomaly detection. In *IMC*, 2008.
- [20] K. Papagiannaki, N. Taft, Z.-L. Zhang, and C. Diot. Long-term forecasting of Internet backbone traffic. In *INFOCOM*, 2003.
- [21] P. Porras, H. Saidi, and V. Yegneswaran. A foray into Conficker’s logic and rendezvous points. In *LEET*, 2009.
- [22] Z. Qian, Z. M. Mao, Y. Xie, and F. Yu. Investigation of triangular spamming: A stealthy and efficient spamming technique. In *IEEE Symposium on Security and Privacy*, 2010.
- [23] H. Ringberg, A. Soule, J. Rexford, and C. Diot. Sensitivity of PCA for traffic anomaly detection. In *SIGMETRICS*, 2007.
- [24] M. Shensa. The discrete wavelet transform: Wedding the à trous and mallat algorithms. In *IEEE Transactions on Signal Processing*, 1992.
- [25] F. Silveira, C. Diot, N. Taft, and R. Govindan. ASTUTE: Detecting a different class of traffic anomalies. In *SIGCOMM*, 2010.
- [26] A. Soule, K. Salamatian, and N. Taft. Combining filtering and statistical methods for anomaly detection. In *IMC*, 2005.
- [27] S. Stover, D. Dittrich, J. Hernandez, and S. Dietrich. Analysis of the Storm and Nugache trojans: P2P is here. In *USENIX Magazine ;login.*, 2007.
- [28] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan. Detecting large-scale system problems by mining console logs. In *SOSP*, 2009.
- [29] J. Zhang, J. Rexford, and J. Feigenbaum. Learning-based anomaly detection in BGP updates. In *SIGCOMM Workshop on Mining Network Data*, 2005.
- [30] Y. Zhang, S. Singh, S. Sen, N. Duffield, and C. Lund. Online identification of hierarchical heavy hitters: algorithms, evaluation, and applications. In *IMC*, 2004.
- [31] Y. Zhao, Y. Xie, F. Yu, Q. Ke, Y. Yu, Y. Chen, and E. Gillum. BotGraph: large scale spamming botnet detection. In *NSDI*, 2009.

## Appendix: Proof of Correctness of ADA

LEMMA 1. For each time instance, after the adaptations we have  $n.ishh$  iff  $n \in SHHH$  for any node  $n \in Tree$ .

PROOF. We prove by induction on time instance. It is clear the argument will hold for the first time instance as there is no adaptation. Now we show the proof will hold for any time instance. For the “if” direction, suppose (by contradiction) there exists a node  $n \in SHHH$  such that  $n.ishh$  is *false*. Then we have  $n.weight < \theta$  because the function Update-Ishh-and-Weight() (Fig. 6) enforces  $n.ishh$  iff  $n.weight \geq \theta$ . Then by the merge rules (line 22 in Fig. 5 and line 1 in Fig. 8), node  $n$  will be merged to its parent so we have  $n \notin SHHH$ . The only exception is the root node  $n_{root}$  which has no parent to merge. However, we check this special case (lines 24-25 in Fig. 5) to enforce  $n_{root}.ishh$  iff  $n_{root} \in SHHH$ .

For the “only if” direction, suppose (by contradiction) there exists a node  $n$  such that  $n.ishh$  is true and  $n \notin SHHH$ . Similarly, we have  $n.weight \geq \theta$ . Because  $n.ishh$  is true,  $n.parent.tosplit$  is also true (by lines 13-17 in Fig. 5). We consider the following two cases. (i)  $n.parent.washh$  is true. In this case, by the induction hypothesis we know  $n.parent$  was in SHHH before any adaptation in this time instance, and therefore  $n.parent$  will split its time series to  $n$  (line 15 in Fig. 5 and line 2 in Fig. 7). If  $n$  has no child, then we have  $n \in SHHH$ . Otherwise, if  $n$  splits its time series, there must exist a child  $n_c$  such that  $n_c.weight < \theta$ . If there does not exist such a child  $n_c$ , then we have  $n.weight = 0$  which contradicts our previous inference  $n.weight \geq \theta > 0$ . Because we have  $n_c.weight < \theta$ , by the merge rules (line 22 in Fig. 5 and line 1 in Fig. 8)  $n_c$  will merge back the time series to  $n$ . Therefore, we have  $n \in SHHH$  after the adaptation stage. If the time series of  $n_c$  is further split into some children, similarly, there must exist a child  $n_{c'}$  of  $n_c$  such that  $n_{c'}.weight \leq n_c.weight < \theta$ . Then  $n_{c'}$  will merge its time series to  $n_c$ , and then  $n_c$  will merge back to  $n$ . It is not hard to see we can apply this argument multiple times to make sure that  $n \in SHHH$  after the adaptation stage. Consider another case (ii)  $n.parent.washh$  is false. Then we have  $n.parent \notin SHHH$  before the adaptation stage. Because  $n \notin SHHH$ , we have  $n.parent.weight \geq n.weight \geq \theta$ . If  $n.parent.parent.washh$  is true, then  $n.parent.parent$  will split its time series to  $n.parent$ , and then  $n.parent$  will split its time series to  $n$  such that  $n \in SHHH$  after the adaptation. Otherwise, if  $n.parent.parent.washh$  is false, we can repeatedly apply the above argument, and finally we have  $n_{root}.weight \geq \theta$  and  $\neg n_{root}.washh$ . Therefore,  $n_{root}$  was not in SHHH before the any adaptation in this time instance, and this violates the our enforcement on root node (lines 24-25 in Fig. 5) in the previous time instance. This implies the induction hypothesis is false, and we reach the contradiction.  $\square$

LEMMA 2. (Holt-Winters' Linearity). Given  $k$  Holt-Winters' forecasts  $G_1[t], G_2[t], \dots, G_k[t]$ , where  $G_i[t] = L_i[t - 1] + B_i[t - 1] + S_i[t - v]$ . Consider a time series  $T^*[t] = \sum_{i=1}^k T_i[t]$  and its Holt-Winters' forecast  $G^*[t] = L^*[t - 1] + B^*[t - 1] + S^*[t - v]$ . Then  $G^*[t] = \sum_{i=1}^k G_i[t]$ .

PROOF. We show  $L^*[t] = \sum_{i=1}^k L_i[t]$ ,  $B^*[t] = \sum_{i=1}^k B_i[t]$  and  $S^*[t] = \sum_{i=1}^k S_i[t]$  by induction on  $t$ . For the initialization, we have

$$\begin{aligned} L^*[t-1] &= \frac{1}{2v} \sum_{j=t-1}^{t-2v} T^*[j] \\ &= \frac{1}{2v} \sum_{j=t-1}^{t-2v} \sum_{i=1}^k T_i[j] \\ &= \sum_{i=1}^k \left( \frac{1}{2v} \sum_{j=t-1}^{t-2v} T_i[j] \right) \\ &= \sum_{i=1}^k L_i[t-1] \end{aligned} \quad (4)$$

$$\begin{aligned} B^*[t-1] &= \frac{1}{2v} \left[ \left( \sum_{j=t-1}^v T^*[j] \right) - \left( \sum_{j=t-v-1}^{2v} T^*[j] \right) \right] \\ &= \frac{1}{2v} \left[ \left( \sum_{j=t-1}^v \sum_{i=1}^k T_i[j] \right) - \left( \sum_{j=t-v-1}^{2v} \sum_{i=1}^k T_i[j] \right) \right] \\ &= \sum_{i=1}^k \left\{ \frac{1}{2v} \left[ \left( \sum_{j=t-1}^v T_i[j] \right) - \left( \sum_{j=t-v-1}^{2v} T_i[j] \right) \right] \right\} \\ &= \sum_{i=1}^k B_i[t-1] \end{aligned} \quad (5)$$

For  $j = 1, \dots, 2v$ , we have

$$\begin{aligned} S^*[t-j] &= T^*[t-j] - L^*[t-1] \\ &= \left( \sum_{i=1}^k T_i[t-j] \right) - \left( \sum_{i=1}^k L_i[t-1] \right) \quad (\text{by (4)}) \\ &= \sum_{i=1}^k (T_i[t-j] - L_i[t-1]) \\ &= \sum_{i=1}^k S_i[t-j] \end{aligned} \quad (6)$$

Then we have following induction hypotheses for  $j =$

$t-1, t-2, \dots, t-2v$ .

$$L^*[j] = \sum_{i=1}^k L_i[j] \quad (7)$$

$$B^*[j] = \sum_{i=1}^k B_i[j] \quad (8)$$

$$S^*[j] = \sum_{i=1}^k S_i[j] \quad (9)$$

Given the induction hypotheses, it suffices to prove the theorem when  $j = t$ .

$$\begin{aligned} L^*[t] &= \alpha(T^*[t] - S^*[t-v]) \\ &\quad + (1-\alpha)(L^*[t-1] + B^*[t-1]) \\ &= \alpha \left[ \left( \sum_{i=1}^k T_i[t] \right) - \left( \sum_{i=1}^k S_i[t-v] \right) \right] + (1-\alpha) \\ &\quad \times \left[ \left( \sum_{i=1}^k L_i[t-1] \right) + \left( \sum_{i=1}^k B_i[t-1] \right) \right] \\ &= \sum_{i=1}^k \{ \alpha(T_i[t] - S_i[t-v]) \\ &\quad + (1-\alpha)(L_i[t-1] + B_i[t-1]) \} \\ &= \sum_{i=1}^k L_i[t] \end{aligned} \quad (10)$$

$$\begin{aligned} S^*[t] &= \gamma(T^*[t] - L^*[t]) + (1-\gamma)(S^*[t-v]) \\ &= \gamma \left[ \left( \sum_{i=1}^k T_i[t] \right) - \left( \sum_{i=1}^k L_i[t] \right) \right] \\ &\quad + (1-\gamma) \left( \sum_{i=1}^k S_i[t-v] \right) \quad (\text{by (9), (10)}) \\ &= \sum_{i=1}^k \{ \gamma(T_i[t] - L_i[t]) + (1-\gamma)(S_i[t-v]) \} \\ &= \sum_{i=1}^k S_i[t] \end{aligned} \quad (11)$$

□