Time Stamp Synchronization of Distributed Sensor Logs: Impossibility Results and Approximation Algorithms

By

THOMAS CHESTER RISTENPART
B.S. (University of California Davis) 2003

THESIS

Submitted in partial satisfaction of the requirements for the degree of

MASTERS OF SCIENCE

in

Computer Science

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

_____

_____

_____

Committee in Charge

2005

Time Stamp Synchronization of Distributed Sensor Logs:
Impossibility Results and Approximation Algorithms

Contents

Acknowledgements

Numerous people have contributed in countless ways to this thesis. First and foremost, I must acknowledge the guidance and support of my many and varied advisors at Davis: Matt Bishop, whose academic integrity, never-ending encouragement, and sharp insights were a model for a true scholar and gentleman; Phil Rogaway, who opened doors to intellectual pursuits I never even new existed and who taught me about formal writing; Hao Chen, whose zeal, determination, and work ethic I aspire to mimic daily; and Zhendong Su, who taught me much about analytical thinking during our brain-storming sessions. Of course, the many other teachers I have had both at Davis and before helped my intellectual growth at every turn. My numerous less official advisors deserve their credit: Dam Backer and Ash Morgan, two early mentors who taught me about the engineer in software engineering; Kim Reinking, not only for initially encouraging me to pursue graduate work, but for her seemingly endless dedication to the students she advises; my labmates and other fellow graduate students, who made possible the late night study sessions, long nights of research, and stimulating discussions that make graduate school such a valuable experience; and my family, for the invaluable support that only they can provide.

Lastly, I owe a huge debt to Promia, Inc., which funded this research, and the stellar researchers found there, particularly Steven Templeton, Mark Heckman, and Ray Granvold.

Abstract

Heterogenous, distributed sensor systems utilize individual sensors (e.g., IDSs, firewalls, and honeypots) that forward alerts to a central location where they are aggregated into log files. Included in alerts are time stamps that record when the sensors observed the alert-triggering behavior. When the system clocks of the sensors are not synchronized, temporal relationships among alerts cannot be directly determined from the time stamps. This impedes any useful analysis of the alert data since even simple temporal relationships such as order might be unrecoverable. Although practitioners have reported repeatedly dealing with such situations, no general solutions to this problem have been explored when a priori synchronization mechanisms (e.g., NTP) are unavailable or misconfigured. This work investigates a completely general mechanism for synchronizing the time stamps of collected alerts using only the data available in the log files. We show that general, precise a posteriori synchronization is impossible, but that simple approximation heuristics work well in realistic settings.

Chapter 1

Introduction

1.1 Distributed Sensor Logs

Increasingly, security auditors must analyze log files containing alerts from a large number of distributed, heterogeneous sensors. For example, enterprise-class organizations such as government entities or large corporations have assets on numerous, large networks. Ensuring coverage of such infrastructure requires a multitude of sensors, including Intrusion Detection Systems (IDS), firewalls, and honeypots. Aggregating alert data from multiple sensors provides a broader view of any malicious behaviors. In an even larger setting, sensors have been deployed across the Internet to create so-called 'Internet sensors'. Examples are the SANS Internet Storm Center [2], CAIDA [1], and Symantec's DeepSight [3]. These distributed Internet sensors collect alert data from numerous locations across the Internet to allow for a very broad view of potentially malicious activities.

Numerous analysis techniques and algorithms have been developed for sensor alert data [6, 7, 11, 18, 19, 28, 29]. Recent work generalizes these analysis techniques for distributed sensors [20, 26, 27]. These techniques and algorithms assist security auditors in the often overwhelming task of assessing the relevance of various alerts. Some techniques attempt to infer attack patterns or behaviors that span multiple alerts. Others correlate low-level alerts to create higher level incident reports, reducing the number of items an analyst must investigate. All such approaches assume that alerts include *synchronized time stamps*: the time stamps share a common view of time and can be used to determine temporal relationships (such as order). This assumption is trivially satisfied when alerts are from a single sensor. In multiple sensor settings, the individual system clocks used by each sensor may not have the same view of time. Clock drift (change in a clock's view of time) and the delays inherent in sending alert data across networks or the Internet complicate the determination of temporal relationships. When given unsynchronized time stamps, the algorithms and techniques utilized by security auditors might fail, or worse, yield misleading interpretations of alert data.

1.2 Possible approaches

Active mechanisms such as NTP [17] that perform a priori synchronization of system clocks can be used to ensure time stamps are synchronized. In practice, it is sometimes difficult to ensure such mechanisms are in use and correctly configured on all sensors. For example, security auditors who passively collect alert data might not have direct control over the configuration of sensors. It becomes an even more complicated issue of coordination when multiple large organizations contribute alerts to a common log file host (as with Internet sensors). In fact, this work was originally motivated by a large security vendor's recurring difficulty with several clients' inability to provide synchronized time stamps. It is therefore important to consider settings where a priori clock synchronization mechanisms are unavailable.

Another possible approach is to adapt individual analysis techniques or algorithms to compensate for unsynchronized time stamps. Serrano suggested an approach where the constraints used to determine temporal relationships are relaxed [26]. For example, multiple alert orderings (besides the one found in the aggregate log file) are considered when correlating alerts. Such an approach is not general; it requires updating individual algorithms. Additionally, it is difficult to determine how much to relax the constraints. If the constraints are relaxed too much, erroneous correlations might occur (false positives). If the constraints are not relaxed enough, correlations might be missed (false negatives).

A previously unexplored approach for this problem is a posteriori time stamp synchronization. With this method, time stamps are translated to a common time frame after the log file has been generated. This avoids the need for active system clock synchronization before time stamps are generated. Successful translation of all time stamps to a common time frame would allow accurate determination of temporal relationships. Because it requires no information beyond what is contained within the aggregate log file and is completely independent of what analysis tools are to be used, time stamp synchronization is a potentially powerful general approach for solving the synchronization problem.

1.3 Contributions

This work rigorously explores time stamp synchronization of distributed sensor logs. Our contributions include the following:

- We describe a model of aggregate log files and the temporal information in them. We focus on two types of aggregate log files. *Type-one log files* include only processing time stamps

(when the remote sensor observed the alert). *Type-two log files* additionally have arrival time stamps (when the log file host received the alert).

- We prove that precise time stamp synchronization is impossible for both type-one and type-two aggregate log files. This is primarily due to not having enough temporal information to bound message delay.

- We introduce two best-effort algorithms. For type-one log files, we give an interval-based algorithm which leverages the order of alerts. For type-two log files, we give an algorithm that synchronizes based on the temporal information of entries that are thought to have minimal delay. Through simulation we show that both of these algorithms perform well in realistic settings, providing accurate synchronization solutions for distributed log files.

Additionally, we note that our model and algorithms are general: they can be applied to time stamps generated by any many-to-one message passing system. Our results could be especially valuable for synchronizating event notifications in sensor networks, where typical a priori synchronization mechanisms utilize too much battery power [10, 13, 23].

1.4 Outline

In Chapter 2 we discuss in detail the settings assumed, a model of aggregate log files with temporal information, and impossibility results. In Chapter 3 we present algorithms for approximate synchronization of both type-one and type-two aggregate log files. In Chapter 4 we describe the simulations used to judge the efficacy of our algorithms and the results of these simulations. In Chapter 5 we present related work in more detail. In Chapter 6 we describe future work and briefly conclude.

Chapter 2

Formal Framework and Impossibility Results

In this chapter we develop a formal model of aggregate log files and use this model to reason about the limitations of time stamp synchronization. Our main result is in Section 2.2: we show that no log file has a precise time stamp synchronization solution. In Chapter 3 we utilize this model to guide the development of approximation algorithms.

2.1 Log Files with Temporal Data

The aggregate log files we consider consist of merged alert streams from multiple remote sensors. A central host receives alerts from the remote sensors and creates entries for each alert in a log file. We call this central host the log file host and notate it with L. We refer to the remote sensors interchangeably as 'remote sensors', 'remote hosts', or simply 'hosts'. Our model thus captures any log files created in a many-to-one message passing architecture.

2.1.1 Clocks and time stamps

Let $\mathcal{H}$ be the set of all remote sensors (hosts) of interest. Each host $h \in \mathcal{H}$ has a local system clock C_h. A simple and sufficient model of a system clock is the linear equation

$$C_h(t) \quad = \quad t + S_h + R_h t \tag{2.1}$$

where the constant S_h is the skew and the constant R_h is the drift [22]. Here t is a time taken from some global reference clock (we will refer to values taken from the global reference clock as the 'actual' time). *Skew* is therefore the initial offset from actual time at time zero. *Drift* is the

rate of change of the clock's skew.[1] Note that our model of system clocks implies that they are monotonically increasing (there are no clock resets). A *configuration* for a set of system clocks is simply a map Conf: $\mathcal{H} \to \mathbb{R} \times \mathbb{R}$ which assigns to each host $h \in \mathcal{H}$ a skew value $S_h \in \mathbb{R}$ and a drift value $0 \leq R_h \in \mathbb{R}$ (where $\mathbb{R}$ is the set of all real numbers).

System clocks are used to create time stamps to record the time of various events occuring within our sensor network. To simplify our treatment, we allow time stamps to be real numbers. (In implementations they are integral; restricting ourselves to integral time stamps does not change our results.) We consider the time stamps of two events: the processing of an alert at a remote host and the arrival of the alert data at the log file host. The time stamp $\overline{p}$ records the time at which some alert is processed at the remote host. The time stamp $\overline{a}$ records the time at which the alert data arrived at the log file host. Letting h be the host where the alert was processed and utilizing the definition of a system clock (Equation 2.1), we have that $\overline{p} = C_h(p) = (1 + R_h)p + S_h$ where p is the actual time the event was processed. Similarly the arrival time stamp is $\overline{a} = C_L(a) = (1 + R_L)a + S_L$ with actual arrival time a. Note that we floor the clock's time to ensure that time stamps are integral. time stamps are generally integral. Although time stamps are generally integral Between the processing of an alert at a remote host and the reception at the log file host occurs potential delays due to computation, queuing, and propagation. We define delay as $d \equiv a - p$.

Two subtelties of our model are worth mentioning. First, clocks in our model count abstract units of time. This framework can therefore be utilized to reason about log files with time stamps expressed in various granularities (e.g., seconds, microseconds, etc.). We worked with time stamps measured in seconds and point out that synchronization is more difficult for higher granularities. With higher granularities, drifts and other types of clock inaccuracy are larger relative to the units used in the time stamps (and thus have larger impact on synchronization attempts). Second, our model allows time stamps to be real numbers (infinitely precise). In implemented systems time stamps are generally restricted to integral values. Our results do not change under such a restriction and the abstraction to real numbers yields simplified exposition.

2.1.2 Log files

A log file is an ordered sequence of entries that correspond to alerts processed at remote hosts. Entries are ordered based on their arrival time. An entry will include relevant time stamps and an identifier for the remote host that generated the alert. In actual systems, there will be other information included in entries. For full generality we assume such information has no temporal

[1] Although our model has drift as a constant, in reality there are always slight deviations in drift for a variety of reasons (an example is temperature change when utilizing normal quartz oscillators). For our purposes these deviations are negligible.

significance.

For simplicity we assume all entries within a log file have the same number and variety of time stamps. We consider two types of log files corresponding to the availability of time stamps. The first type of log file has entries that include only the remote host processing time stamp $\overline{p}$ for each event. Formally, a *type-one log file* is a finite, ordered list of triples $(i, \overline{p}_i, h_i)$ where i is the rank of the triple in the list, $\overline{p}_i$ is the time stamp of the event, and h_i is the identifier of the host that sent the event.

The second type of log file has entries that include both $\overline{p}$ and the arrival time stamp $\overline{a}$ in each entry. Formally, a *type-two log file* is a finite, ordered list of four-tuples $(i, \overline{p}_i, \overline{a}_i, h_i)$ where i is the rank of the triple in the list, $\overline{p}_i$ is the processing time stamp of the event, $\overline{a}_i$ is the reception time stamp, and h_i is the identifier of the host that sent the event.

For convenience we let $H_F \subseteq \mathcal{H}$ be the set of all hosts that have at least one entry in a type-one or type-two log file F. Note that any type-two log file can be converted into a type-one log file simply by removing all arrival time stamps. Whenever we do not specify which type of log file, it is assumed that both types are being considered.

2.1.3 Causality

The events that are reported on by remote hosts in a distributed system may or may not have causal relationships. A simple example of a causal relationship is the happened-before relation [14]: an event A must have happened before an event B. Any causal relationship suggests constraints on the relative timings of events. In our example, the time at which event A occured must be before the time at which event B occured. For full generality, we assume no external knowledge about causal relationships between the alerts found in a log file (i.e., an alert of type B always follows an alert of type A).

On the other hand, we do make a few natural assumptions about causual relationships that exist between events related to the generation of the log file. Let $(i, \overline{p}_i, \overline{a}_i, h_i)$ and $(j, \overline{p}_j, \overline{a}_j, h_j)$ be any two entries in a log file such that $i < j$. Then the following statements hold:

1. $a_i \leq a_j$ (correct ordering of events),
2. $p_i \leq p_j$ whenever $h_i = h_j$ (order reflects processing times), and
3. $p_i \leq a_i$ (non-negative delay).

The first states that all entries are partially ordered by their arrival times; giving rise to the rank of each entry (ties are handled arbitrarily). The second states that entries from each host arrive in the

order of their processing times. Particularly, we assume for simplicity that event notifications are sent to the log file host in the order that they are processed and that no delays are significant enough to interfer with this ordering. Due to the fact that we assume system clocks are monotonically increasing, this assumption is equivalent to assuming that $\overline{p}_i \leq \overline{p}_j$ for entries $i < j$ with $h_i = h_j$. It is trivial to detect when this is violated, and such entries can then be easily ignored. The last statement ensures that delay cannot be negative—it takes at least zero time units to send an event notification to the log file host. Allowing delays of zero will be useful in our formalisms, and in some settings might not be a bad approximation for very small delays.

2.1.4 Time stamp synchronization

When all clocks involved are properly synchronized, temporal relationships can be accurately inferred from time stamps within a log file. However, in our setting we assume that the clocks are not synchronized a priori. Unsynchronized clocks may not agree due to a variety of reasons, incorrect configuration and differing time zones being two. Even if they are initially synchronized by hand, clock drift will cause their view of time to diverge.

The goal of a posteriori time stamp synchronization mechanisms is to modify all time stamps so that they accurately represent the temporal data of each alert as seen from a common time frame. In our setting the common time frame will be the clock of some distinguished remote host, which we will call the *reference host*. By translating all processing time stamps $\overline{p}$ to the common time frame of the reference host, correct temporal relationships between events can be derived. To accomplish this, any mechanism must calculate a *correction offset* for each processing time stamp in a log file. Consider an entry $(i, \overline{p}_i, \overline{a}_i, h_i)$ and an arbitrary reference host r. The precise correction offset to convert $\overline{p}_i$ to the reference host's clock C_r is $\mathit{off}_r(i) \equiv (R_r - R_{h_i})p_i + S_r - S_{h_i}$. Adding this correction offset to the time stamp yields

$$\begin{aligned}
\overline{p}_i + \mathit{off}_r(i) &= C_h(p_i) + (R_r - R_h)p_i + S_r - S_{h_i} \\
&= (1 + R_h)p_i + S_h + (R_r - R_h)p_i + S_r - S_{h_i} \\
&= (1 + R_r)p_i + S_r \\
&= C_r(p_i)
\end{aligned}$$

which is the the processing time stamp of the event relative to the reference host's clock. For a log file, we call the set of offsets that precisely translate each time stamp to any host r's time frame a *synchronization solution*.

2.2 Limitations and Impossibility Results

An intrinsic limitation for time stamp synchronization of aggregate log files occurs due to the interplay between skew and delay. Because messages (alert information) are sent in only one direction, there is no way to give an upper bound on the delay a message encountered. We can imagine two scenarios for a single message passed from a remote sensor. In the first scenario, we imagine that the message containing the alert encountered little delay. This implies that the alert was processed at the remote sensor recently. In the second scenario, we imagine that the message containing the alert encountered significant delay. This implies that the alert was processed at the remote sensor long ago. Now for the key point: the processing time stamp of the alert could be the same for both scenarios. Particularly, if the remote sensor's system clock has very small skew in the first scenario and has very large skew in the second scenario, then the resulting time stamps could be basically equal. Thus, in attempting a posteriori time stamp synchronization, we do not know whether an alert was generated in the first scenario or the second.

Generalizing on this thought experiment, we can show formally that the interplay between skew and delay allows for infinitely many 'scenarios'. We begin by defining a *correct configuration*. Recall that a configuration is simply an assignment of skews and drifts to each clock in the system. For a given log file, a correct configuration is one that could have been in use when the log file was generated, thus corresponding to the notion of a possible scenario.

Definition 1 *[Correct configuration] Let* $F = (1, \overline{p}_1, \overline{a}_1, h_1), \ldots, (n, \overline{p}_n, \overline{a}_n, h_n)$ *be a type-two log file. We say that a configuration* $\mathsf{Conf}\colon \mathcal{H} \to \mathbb{R} \times \mathbb{R}$ *for the system clocks represented in* F *is correct if and only if it yields a solution to the system of linear equations described by*

$$\overline{p}_i = p_i(1 + R_{h_i}) + S_{h_i} \qquad i \in [1..n] \tag{2.2}$$

$$\overline{a}_i = a_i(1 + R_L) + S_L \qquad i \in [1..n] \tag{2.3}$$

$$a_i \leq a_j \qquad i, j \in [1..n], i < j \tag{2.4}$$

$$p_i \leq p_j \qquad i, j \in [1..n], i < j, h_i = h_j \tag{2.5}$$

$$p_i \leq a_i \qquad i \in [1..n] \tag{2.6}$$

which is implied by the time stamps and the ordering of entries within the log file.

The existence of multiple correct configurations for any log file implies that there is an inherent ambiguity restricting the efficacy of time stamp synchronization mechanisms.

Theorem 1 *Any log file with one correct configuration has many correct configurations.*

Proof: Let $F = (1, \overline{p}_1, \overline{a}_1, h_1), \ldots, (n, \overline{p}_n, \overline{a}_n, h_n)$ be a type-two log file. We assume that it has a correct configuration $\mathsf{Conf}\colon \mathcal{H} \to \mathbb{R} \times \mathbb{R}$. We show how to construct another configuration Conf'

that is distinct from Conf and also correct. We define Conf′ as follows

$$\mathsf{Conf}'(h) \quad = \quad (S_h + \delta_h, R_h)$$

where δ_h is some positive real number (we choose one for each host $h \in H_F$). Now we must show that this configuration is correct, which amounts to showing that there is a solution to the equations listed in Definition 1. First we note that by assumption Conf was correct, and therefore has a solution to its corresponding equations. Let p_i, a_i for $i \in [1..n]$ be that solution. Let p'_i, a'_i be the solution we seek for Conf′. Then for $i \in [1..n]$ we let

$$p'_i = p_i - \frac{\delta_{h_i}}{1 + R_{h_i}} \quad \text{and} \quad a'_i = a_i \,.$$

Now we must show that these assignments actually solve the various equations. For each $i \in [1..n]$, we have that

$$\begin{aligned} \overline{p}_i &= p'_i(1 + R_{h_i}) + S_{h_i} + \delta_{h_i} \\ &= \left(p_i - \frac{\delta_{h_i}}{1 + R_{h_i}}\right)(1 + R_{h_i}) + S_{h_i} + \delta_{h_i} \\ &= p_i(1 + R_{h_i}) + S_{h_i} \end{aligned}$$

and thus all of the equations defined by Equation 2.2 are satisfied. It is trivial to verify that the equations related to Equations 2.3, 2.4, and 2.5 are satisfied. Finally, the equations related to Equation 2.6 are satisfied since we restricted δ_h to be positive for all hosts h. Since δ_h can take on any positive value, we have shown that there are an infinite amount of possible correct configurations. □

A straightforward corollary of Theorem 1 is that log files with a correct configuration have many synchronization solutions.

Corollary 1 *Any log file that has a correct configuration has many synchronization solutions.*

Proof: By Theorem 1 any log file with a correct configuration has many. For each time stamp $\overline{p}_i$ and for arbitrary reference host D, a precise correction offset is

$$(R_D - R_{h_i})t_i + S_D - S_{h_i}$$

where R_D, R_{h_i}, S_D, and S_{h_i} are given by any correct configuration. Since there are many possible correct configurations, there are many precise correction offsets. □

It is important to note that any log file created under our assumptions must have a correct configuration: the one that was actually in use during the generation of the log file. Thus, for any log file our model captures, there are many possible synchronization solutions. In turn, this implies that the best a general algorithm can do is approximately synchronize all the time stamps.

Chapter 3

Approximation Algorithms

In this chapter, we investigate algorithms for approximating synchronization solutions. Recall that a synchronization solution is a set of correction offsets that translate all time stamps in a log file to be on the time frame of a reference host's clock. Furthermore, these correction offsets are a function of drift, skew, and the actual time at which the time stamp was created. To simplify our task, we assume that drifts are negligible and thereby reduce correction offsets to be a function of (constant) skews. (Note that our impossibility result in the previous chapter still applies to a setting where we force restrictions on clock drift.) Consider some log file and arbitrary reference host r. For all hosts h assume that $R_h = 0$. Then, for each entry $(i, \overline{p}_i, \overline{a}_i, h)$ with $h \neq r$, we have that the correction offset is $(R_r - R_h)t_i + S_r - S_h = S_r - S_h$. We call any difference in skews a relative skew. We have thus reduced the problem of finding correction offsets to simply identifying the (constant) relative skews of hosts to a reference host.

This chapter therefore focuses on developing algorithms for approximating relative skews in both type-one and type-two log files with negligible drifts. We first investigate type-one log files. We show that under certain assumptions about delay, the ordering of events within a log file can be utilized to create bounds on relative skews. We show that it is possible to sometimes detect when these assumptions are violated and propose a heuristics-based approach for such situations. We then turn to type-two log files and show how to utilize the extra temporal information provided by the arrival time stamps. We propose two algorithms: a simple stastical one and one based on finding entries with minimal delay. Finally, we discuss some considerations concerning all the algorithms, particularly non-negligible drifts and the impact of reference host selection.

3.1 Type-one Log Files

3.1.1 Utilizing Order

In our setting the order of entries within a log file is temporal information that we can leverage to bound relative skews approximately. Particularly, the ordering gives us information about the relative values of the arrival times of all the events. In turn, this knowledge about the arrival times allows us to calculate lower and upper bounds on relative skews.

We consider a type-one log file $F = (1, \overline{p}_1, h_1), \ldots, (n, \overline{p}_n, h_n)$ with all drifts negligible ($R_h = 0$ for all $h \in H_F$). Let $(i, \overline{p}_i, g)$ and $(j, \overline{p}_j, h)$ be two entries such that $i < j$ and $g \neq h$. Then, we note that

$$\begin{aligned} \overline{p}_i - \overline{p}_j &= (1 + R_g)p_i + S_g - (1 + R_h)p_j - S_h \\ &= p_i - p_j + S_g - S_h \end{aligned} \tag{3.1}$$

where the second equation utilizes the negligible drift assumption. The definition of delay (see Section 2.1) tells us that $p_i = a_i - d_i$ and $p_j = a_j - d_j$. We can therefore cast Equation 3.1 in terms of arrival times and delays:

$$\begin{aligned} \overline{p}_i - \overline{p}_j &= a_i - d_i - a_j + d_j + S_g - S_h \\ &= a_i - a_j + (S_g - S_h) + (d_j - d_i)\,. \end{aligned} \tag{3.2}$$

Since $a_i \leq a_j$, we know that $\overline{p}_i - \overline{p}_j$ is a lower bound on $S_g - S_h + (d_j - d_l)$.

We can similarly derive an upper bound. Let $(k, \overline{p}_k, h)$ and $(l, \overline{p}_l, g)$ be two events in the log file such that $k < l$. Then,

$$\begin{aligned} \overline{p}_l - \overline{p}_k &= (1 + R_g)p_l + S_g - (1 + R_h)p_k - S_h \\ &= a_l - d_l - a_k + d_k + (S_g - S_h) \\ &= a_l - a_k + (S_g - S_h) + (d_k - d_l)\,. \end{aligned} \tag{3.3}$$

Here we have that $a_l \geq a_k$ which implies that $a_l - a_k \geq 0$. Therefore $\overline{p}_l - \overline{p}_k$ is an upper bound on $S_g - S_h + (d_k - d_l)$. Notice that deriving these bounds does not require explicit knowledge of the arrival times.

If we assume that the difference in delays for any two events is negligible, then we can utilize the bounds described to guide our selection of correction offsets. We therefore make the *negligible delay assumption*: for all events $(i, \overline{p}_i, g)$ and $(j, \overline{p}_j, h)$ we have $d_i - d_j = 0$. In other words, all hosts' delays were constant and equal when the log file was created.[1] Note that the closer

[1] In the next section we show how to detect when this assumption is violated and suggest approaches for handling such situations.

together arrivals are at the log file host (i.e., as the difference in arrival times approaches zero) the tighter these bounds become.

Every pair of events in a log file contributes a potential lower and upper bound. We scan a log file and produce two sets of bounds for each pair of hosts. For each distinct pair of hosts g and h in a log file let

$$B_{gh} = \{\, \overline{p}_i - \overline{p}_j \mid i < j \wedge (i, \overline{p}_i, g), (j, \overline{p}_j, h) \in F \,\}$$

and

$$T_{gh} = \{\, \overline{p}_j - \overline{p}_i \mid i < j \wedge (i, \overline{p}_i, h), (j, \overline{p}_j, g) \in F \,\}$$

be the sets of all lower bounds and upper bounds on the relative skew and delay. Then, by the negligible delay assumption we get that $\max\{B_{gh}\} \leq S_g - S_h \leq \min\{T_{gh}\}$. In other words, the interval $[\max\{B_{gh}\}, \min\{T_{gh}\}]$ is guaranteed to contain the relative skew. We can not naively pick values from these intervals as correction offsets, lest we end up with an inconsistent synchronization solution, in the following sense. Consider distinct $g, h, r \in H_F$ where r is the reference host. When we pick values $off_r(g) \in [\max\{B_{gr}\}, \min\{T_{gr}\}]$ and $off_r(h) \in [\max\{B_{hr}\}, \min\{T_{hr}\}]$, we should have that

$$\begin{aligned} off_r(g) - off_r(h) &\approx S_r - S_g - (S_r - S_h) \\ &= S_h - S_g \end{aligned}$$

and thus if our selections are good we should have that $\max\{B_{gh}\} \leq off_r(g) - off_r(h) \leq \min\{T_{gh}\}$.

3.1.2 Determining Tight Intervals

Our problem has a natural graph-theoretic representation that handles this consistency issue via shortest-path analysis. We view our problem as a Simple Temporal Problem (STP), as described by Dechter et al. [8]. An STP consists of a set of temporal variables with bounds on their differences. In our case the temporal variables are the skews of each host and the differences are the relative skews that we seek. Formally, given a log file F we create a directed, weighted graph $G = (V, E, w)$ for which

- $V = \{S_h \mid h \in \mathcal{H}\}$
- $E = \{(g, h) \mid h, g \in \mathcal{H}\}$
- $w(g, h) = \begin{cases} \min\{T_{gh}\} & \text{if } U_{gh} \neq \emptyset \\ \infty & \text{otherwise} \end{cases}$.

We call any such graph G a constraint graph. Each node represents a host's skew and an edge weight $w(g,h)$ corresponds to the bound $S_g - S_h \leq \min\{T_{gh}\}$.[2] We utilize infinite weightings to represent when there is insufficient entries to give us even a single data point for a pair of hosts. Note that a path $S_1, S_2, \ldots, S_k$ implies a constraint

$$S_1 - S_k \leq \sum_{h=1}^{k-1} w(S_h, S_{h+1})$$

which captures the constraints for a consistent solution. Thus, the tightest upper bound on the relative skews associated with two hosts g and h is the value of the shortest path from g to h. Let δ_{gh} represent the shortest path from node S_g to node S_h. Then,

$$-\delta_{hg} \leq S_g - S_h \leq \delta_{gh}$$

represents the bounds that allow for a consistent synchronization solution. These are minimal in the sense that they include the only possible choices of $S_g - S_h$ that do not violate some constraint. We will therefore utilize the graph $G' = (V, E, \delta)$ where we give each edge (g,h) the weight δ_{gh}.

We will not be able to create G' if there exists a negative-weight cycle in G [5]. A negative-weight cycle is simply a cycle $S_1, S_2, \ldots, S_k = v_1$ such that $\sum_{h=1}^{k-1} w(S_h, S_{h+1}) < 0$. Dechter et al. showed that a STP has a solution if and only if the graph G has no negative-weight cycles [8]. Furthermore, we can show that a negative-weight cycle will only occur when the negligible delay assumption has been violated.

Theorem 2 *Let $G = (V, E, w)$ be the constraint graph associated with some type-one log file F with which $R_h = 0$ for all $h \in H_F$. If the negligible delay assumption holds, then there are no negative-weight cycles in G.*

Proof: Assume the negligible delay assumption holds and that there exists a cycle $S_1, S_2, \ldots, S_k = S_1$ in G which has negative weight:

$$\sum_{h=1}^{k} w(S_h, S_{h+1}) < 0\,.$$

We now derive a contradiction. We have that

$$\begin{aligned} w(S_h, S_{h+1}) &= \overline{p}_j - \overline{p}_i \\ &= a_j - d_j + S_{h+1} - a_i + d_i - S_h \\ &= (a_j - a_i) + S_{h+1} - S_h \end{aligned}$$

[2] For clarity we slightly abuse notation and simply use the name of the host as input for the weighting function.

where $a_i \leq a_j$ are the arrival times associated with entries from host h and $h+1$, respectively. Thus, $a_j - a_i \geq 0$. Returning to our summation and utilizing this fact that all arrival time differences are greater than or equal to zero we have that

$$\begin{aligned} \sum_{h=1}^{k} w(S_h, S_{h+1}) &\geq \sum_{h=1}^{k-1} S_{h+1} - S_h \\ &= 0\,. \end{aligned}$$

This contradicts the assumption that the cycle's weight is negative. There can therefore be no negative-weight cycle in G. □

Thus, with the negligible delay assumption we are guaranteed that all constraints can be satisfied by some assignment of values to the relative skews. Finding such an assignment entails a simple backtrack-free search through the constraints given in G'. We first pick some host r to be the reference host, effectively setting $S_r = 0$. We then iterate over all other hosts h, choosing a value $S_r - S_h \in [-\delta_{rh}, \delta_{hr}]$ such that

$$-\delta_{hg} \leq S_r - S_h - (S_r - S_g) \leq \delta_{gh}$$

for all hosts g that have already had a value assigned to $S_r - S_g$. Dechter et al. show that when G contains no negative-weight cycles, such a backtrack-free search will always succeed.

3.1.3 Detecting Variable Delay

If remote hosts are spread across large geographical distances and connected to the log file host via the Internet, large variations in delay will almost certainly affect temporal data in a log file. Under these circumstances, the negligible delay assumption will not hold and the initial algorithm will probably fail. In this section we explore the effects of variable delay. Under certain conditions we show that variations in delay can be detected. With this in mind, we propose heuristics to assist the initial algorithm in finding correction offsets even in this difficult setting.

When the negligible delay assumption does not hold, variations in delay can cause problems for the initial algorithm. Variations in delay imply that we can not be assured that $\max\{B_{gh}\} \leq \min\{T_{gh}\}$ holds for all hosts h, g. More formally, if $\max\{B_{gh}\} > \min\{T_{gh}\}$, then there exist four events $(i, \overline{p}_i, h)$, $(j, \overline{p}_j, g)$, $(k, \overline{p}_k, g)$, and $(l, \overline{p}_l, h)$ such that $i < j$ and $k < l$ and $\overline{p}_i - \overline{p}_j > \overline{p}_l - \overline{p}_k$. Then, by inspecting Equations 3.2 and 3.3 we can see that

$$a_i - a_j + d_i - d_j \quad > \quad a_k - a_l + d_k - d_l\,.$$

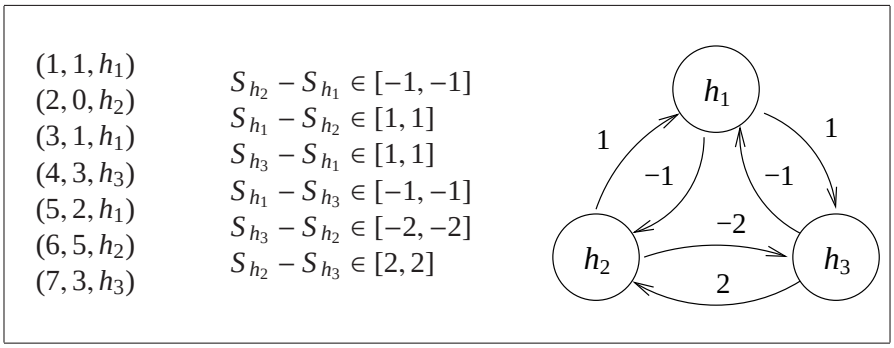

Figure 3.1: Example type-one log file that has an no intersection failures but a negative cycle in the associated difference graph.

Since $a_i - a_j \leq 0 \leq a_k - a_l$, the delays involved must have changed. A graph G constructed with any such interval will necessarily have a negative weight cycle. We have that

$$\begin{aligned} -w(h,g) &> w(g,h) \\ w(h,g) &< -w(g,h) \\ w(h,g) + w(g,h) &< 0 \end{aligned}$$

Thus, the cycle S_g, S_h, S_g must have negative weight.

Of course, there can still be negative-weight cycles even if all intervals are well-defined. Figure 3.1 illustrates an example. The first column gives an example type-one log file. The second column specifies the bounds $[\max\{B_{gh}\}, \min\{T_{gh}\}]$ for the relative skews of each pair of hosts $h, g \in \{h_1, h_2, h_3\}$. For example, the relative skew $S_{h_1} - S_{h_2}$ must be within $[1, 1]$. The lower bound is due to entries one and two while the lower bound is due to entries two and three. The last column gives the associated constraint graph G for the log file. It clearly has a negative-weight cycle although all intervals are well-defined. Thus, a more sensitive mechanism for detecting variations in delay is the existence of negative weight cycles in G.

Whether there are even more sensitive methods for detecting variations in delay is an open question. However there is an intrinsic limitation based on the inter-arrival times - small variations in delay will be masked by larger gaps in arrivals.

3.1.4 Heuristics

Now that variations in delay can potentially be detected, the question becomes what to do about it. An insufficient solution is to simply give up whenever delay variations are detected; variations that trip our detection mechanisms will most likely occur frequently in actual log files. Another approach would be to attempt to insulate the effects of variable delay from our correction

offset calculations as much as possible.

We propose a simple heuristic for accomplishing this, which we call the good prior state heuristic. First we modify our algorithm so that it is on-line: for each new entry in the log file we calculate all bounds that it implies (with previous entries), update the STP graph, and run the APSP algorithm. If the algorithm succeeds, we continue. If the algorithm fails, then we simply ignore the entry's temporal data entirely and remove its effects from the current state. As the name implies, this heuristic basically assumes that temporal data earlier in a log file is less likely to be poor than data later in a log file. One can easily imagine other potential heuristics; we leave analysis of them to future work.

3.1.5 Algorithm INTERVAL-SYNCHRONIZE

Figure 3.2 gives pseudocode for the proposed algorithm for type-one log files. It begins by initializing a complete, weighted graph $G' = (V, E, \delta)$ where V contains a node for each remote host represented in the log file and δ is a weighting function that labels each edge with the shortest path between any two nodes. All edges initially have infinitely large weight. The algorithm iterates over all entries within the log file. The set R includes the most recent entry (relative to the current entry) for all hosts. In each iteration, the graph weights are updated with the shortest paths between each pair of nodes. The subroutine APSP is any all-pairs shortest-path algorithm that returns true when the graph has no negative-weight cycles. Floyd's APSP [5] algorithm is simple and suitable and is utilized in our implementation. It suffices to simply maintain the shortest-path graph G' (as opposed to maintaining G and running APSP from scratch each iteration), since only weights that are smaller than the current shortest path value between any two nodes will have an effect on G'. If an entry causes the graph to contain a negative-weight cycle, we drop the entry as per the good-prior state heuristic. Of course, if the entries are handled in some order other than the one presented in the log file then another heuristic would be utilized. The subroutine P -R -H chooses a reference host. In our implementation this routine simply chooses the host which has the tightest constraints relative to all the other hosts. The subroutine F -S does a backtrack-free search of the constraints in order to generate an array of correction offsets, one per host.

The correctness of the algorithm follows from our previous development of the relationships between time stamps and shortest paths. As per our impossibility results of Chapter 2, the algorithm is best-effort and there are no guarantee that the synchronization solution is correct. In the next chapter we evaluate it experimentally through simulation. The running time is $O(n|V|^3)$ where n is the number of entries in the log file. Note that $|V|$ is the number of hosts found in the log file.

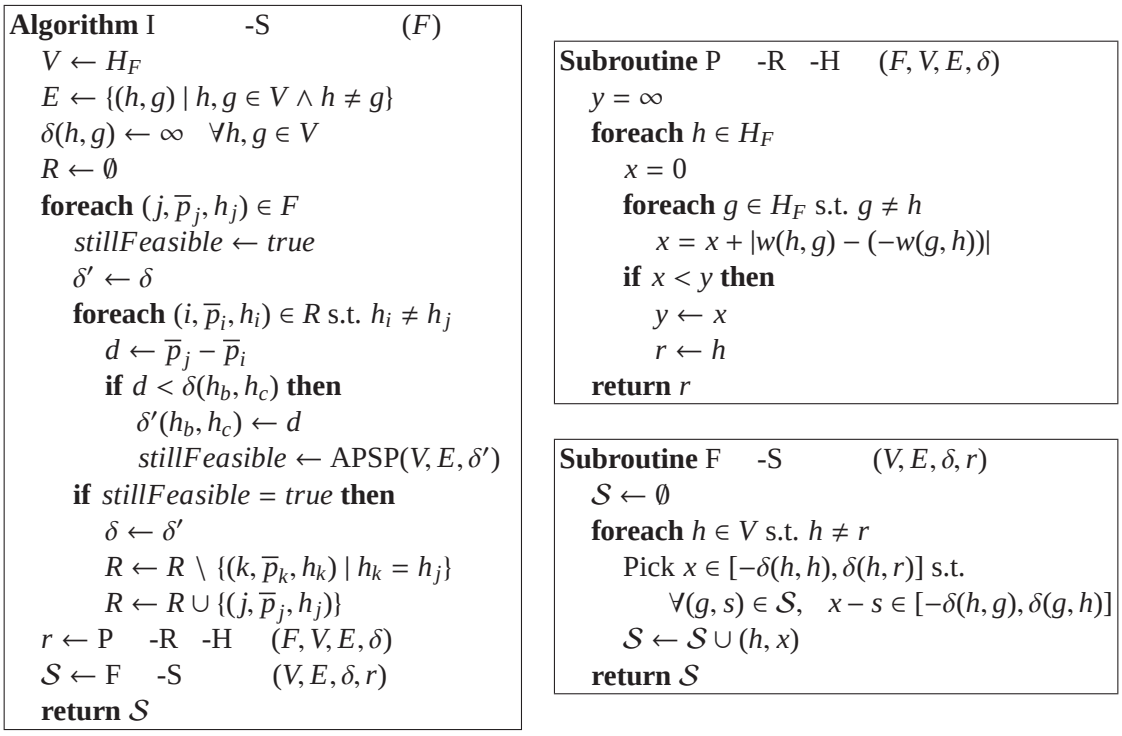

Figure 3.2: Algorithm for determining a synchronization solution for a type-one log file. The function APSP is any all points-shortest path algorithm that outputs false whenever the graph has negative-weight cycles (it also updates δ' to reflect the new shortest paths).

3.2 Type-two Log Files

Type-two log files include the arrival time stamps for each entry. This extra temporal information specifies, according to the log file host's clock, when the alert was received. Even with this extra temporal information our impossibility results from Chapter 2 still apply. In attempting to approximate solutions, we could utilize the approach given for type-one log files, but this would ignore the arrival time stamps. Instead, we propose two algorithms that utilize this extra temporal information: one using a simple statistical approach and the other based on finding entries with minimal delay.

3.2.1 Utilizing Arrival Time Stamps

Let $(i, \overline{p}_i, \overline{a}_i, h)$ and $(j, \overline{p}_j, \overline{a}_j, g)$ be two time stamps from a type-two log file. First we look at what is given by the difference of the two arrival time stamps:

$$\begin{aligned} \overline{a}_j - \overline{a}_i &= (1 + R_L)a_j + S_L - (1 + R_L)a_i - S_L \\ &= a_j - a_i + (a_j - a_i)R_L \end{aligned}$$

where L is the log file host. If we make the negligible drift assumption then $\overline{a}_j - \overline{a}_i = a_j - a_i$. We can thus utilize this information in combination with the difference in processing time stamps:

$$\begin{aligned} \overline{p}_j - \overline{p}_i &= p_j - p_j + S_g - S_h \\ \overline{p}_j - \overline{p}_i &= a_j - a_i + d_i - d_j + S_g - S_h \\ \overline{p}_j - \overline{p}_i - (\overline{a}_j - \overline{a}_i) &= d_i - d_j + S_g - S_h \,. \end{aligned} \tag{3.4}$$

For every pair of hosts we can now easily calculate the relative skew plus relative delay. If in the above $h = g$, then the right hand side of Equation 3.4 simplifes to $d_i - d_j$.

3.2.2 A Stastical Approach: Algorithm Stat-Synchronize

If we view each host's delay as a random variable, then we can utilize statistics to guide our search for relative skews. A very simple approach would be to assume that the difference in delays is uniformly distributed. Consider two hosts h, g and let $c_{ij} = \overline{p}_j - \overline{p}_i - (\overline{a}_j - \overline{a}_i)$ for all entries $(i, \overline{p}_i, \overline{a}_i, h)$ and $(j, \overline{p}_j, \overline{a}_j, g)$. A uniform distribution on delay differences implies that the mean of all the c_{ij} values is an approximation of $S_h - S_g$. Of course, if these differences are not uniformly distributed this estimation will be wrong. Perhaps other assumptions regarding the distribution of the delay (and also their differences) could be utilized; we leave this to future work.

Figure 3.3 gives pseudocode for a simple stastical algorithm called S -S that approximates relative skews. If r is a reference host then the offset calculated for every other host h is equal to

$$\sum_{i,j} [\overline{p}_i - \overline{a}_i - (\overline{p}_j - \overline{a}_j)] \,/\, \sum_{i,j} 1$$

where the summations are over all event entries of the form $(i, \overline{p}_i, \overline{a}_i, h)$ and $(j, \overline{p}_j, \overline{a}_j, r)$. In our implementation the reference host is chosen to be the host with the minimal sum of variances relative to the other hosts. The asymptotic complexity is $O(n^2)$.

3.2.3 A Minimal Delay Approach: Algorithm Min-Delay-Synchronize

Another approach revolves around finding entries from each host which have minimal delay. When both entries are from the same host, Equation 3.4 gives us values that are equal to the difference in two delays. This hints that between arrival time stamps and processing time stamps we have enough information to analyze the delays that were involved when entries were created. Particularly, we would like to find the entry from each host which has the smallest delay associated with it. Although we can not know the exact delays, we can determine which entry has the least delay.

```
Algorithm S  S  (F)
  D_hg ← 0 for all g,h ∈ H_F
  V_hg ← ∅ for all g,h ∈ H_F
  n_hg ← 0 for all g,h ∈ H_F
  foreach (i, p̄_i, ā_i, h) ∈ F
    foreach (j, p̄_j, ā_j, g) ∈ F s.t. i < j ∧ g ≠ h
      D_hg ← D_hg + p̄_i − ā_i − (p̄_j − ā_j)
      V_hg ← V_hg ∪ {p̄_i − ā_i − (p̄_j − ā_j)}
      n_hg ← n_hg + 1
  foreach h ∈ H_F
    foreach g ∈ H_F s.t. g ≠ h
      A_hg ← D_hg/n_hg
  r ← P  R  H  (F, A, V, n)
  S ← F  S  (H_F, A, r)
  return S
```

```
Subroutine P  R  H  (F, A, V, n)
  m ← ∞
  foreach h ∈ H_F
    s ← 0
    foreach g ∈ H_F s.t. g ≠ h
      foreach x ∈ V_hg
        σ² ← (x − A_hg)²/n_hg
      s ← s + σ²
    if s < m then
      m ← s
      r ← h
  return r
```

```
Subroutine F  S  (H_F, A, r)
  S ← ∅
  foreach h ∈ H_F s.t. h ≠ r
    x ← A_hr
    S ← S ∪ (h, x)
  return S
```

Figure 3.3: A simple stastical algorithm for finding a synchronization solution for type-two log files

Let $(i, \overline{p}_i, \overline{a}_i, h)$ be an entry for a host h. Under the negligible drift assumption, we have that

$$\begin{aligned} \overline{p}_i &= p_i + S_h \Leftrightarrow \\ \overline{p}_i &= a_i - d_i + S_h \Leftrightarrow \\ \overline{p}_i &= \overline{a}_i + S_L - d_i + S_h \Leftrightarrow \\ \overline{p}_i - \overline{a}_i &= S_L + S_h - d_i \,. \end{aligned}$$

Recall that both S_L and S_h are constants and $d_i \geq 0$. By comparing the values $\overline{p}_i - \overline{a}_i$ for all entries from the same host, we can see that the largest such value will be from the entry with the smallest delay. We'll call any such entry a *minimal-delay entry*.

To find relative skews, we first identify a minimal-delay entry from each host. Then, for each pair of minimal-delay entries $(i, \overline{p}_i, \overline{a}_i, h)$ and $(j, \overline{p}_j, \overline{a}_j, g)$ we calculate the relative skew

$$\overline{p}_j - \overline{a}_j - (\overline{p}_i - \overline{a}_i) = d_i - d_j + S_g - S_h \,.$$

Since the two values d_i and d_j are minimal, we should get a good approximation of $S_g - S_h$ (at the limit, when both delays are zero, then we know the exact relative skew).

Figure 3.4 gives the pseudocode implementing the M -D -S algorithm. The main loop iterates over all the entries in a log file, maintaining a set M of minimal-delay events

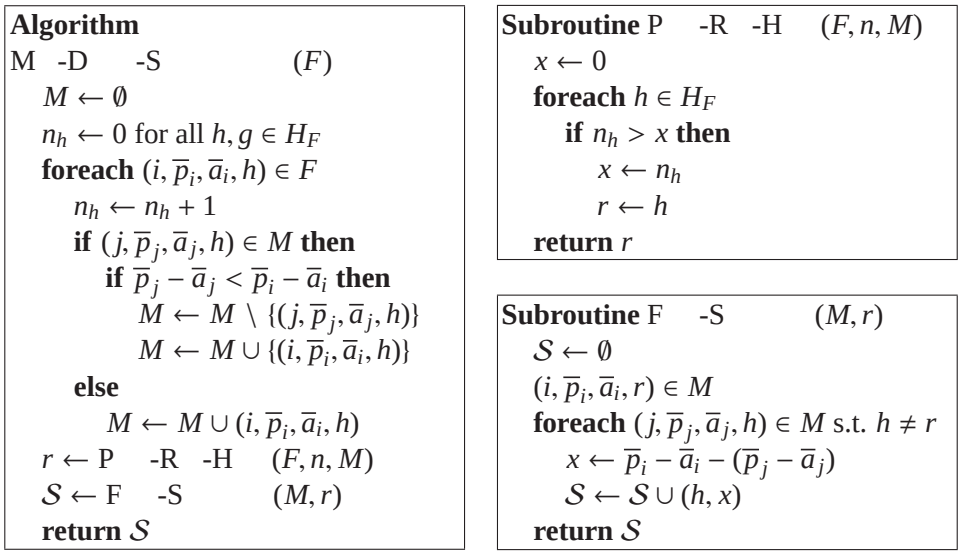

Figure 3.4: Algorithm for finding a synchronization solution for a type-two log file. By the end of the main loop, the set M contains a minimal-delay event for each host.

(one per host). After this a reference host is chosen. In our implementation we simply chose the host with the most entries in the log file. Finally, a synchronization solution is created utilizing the minimal-delay events. The algorithm is $O(n + |M|^2)$ where n is the number of entries in the log file. Note that the size of M is the number of hosts present in the log file. It is much simpler and asymptotically faster than the I -S algorithm. In the next section we will show that it is also far more resistant to variable delay.

The algorithm is $O(n + |M|^2)$ where n is the number of entries in the log file. Note that the size of M is the number of hosts present in the log file. It is much simpler and faster than the other algorithms so far considered. In the next chapter we will see that it is far more accurate, in addition.

3.3 Reference Host Selection

All of our algorithms select a reference host from among the remote sensors. The methods for selection described are designed to choose a sensor that will hopefully allow for a good synchronization solution. The choice of reference host matters. For example, sensors that have a small number of entries in the log file will intuitively not make good candidates. A more thorough investigation of reference host selection is left to future work.

3.4 Non-negligible Drift

Our algorithms assume negligible drift. This simplifies the development of algorithms greatly. Our experience indicates that negligible drift will be a safe assumption in many settings. Most common system clocks utilize quartz oscillators with drift rates of about 1 ± 10^{-6}, which implies that they drift by no more than one second in a day [14, 30]. The effect of this drift on approximations is relative to the scale considered. However, we expect that delay will overshadow drift as a source of error. Our experiments in Chapter 4 confirm this expectation.

Still, there are techniques for handling non-negligible drift. Hofmann and Hilgers present a simple approach that involves partitioning a log file into portions of equal duration [22]. Each portion is ran through a synchronization algorithm separately. Regression analysis can be applied to the resulting solutions to derive approximations for drift.

Chapter 4

Experiments

In this chapter, we test our algorithms experimentally using simulated log files. We confirm that our algorithms perform well in settings with small delays. Additionally, we show that our minimal-delay algorithm for type-two log files is resistant to significant variability and spikes in delay.

4.1 Simulating Log Files

Ideally we would like to test our algorithms using data collected from actual distributed sensors. To do this we would require extra timing information about the conditions under which the log files were collected. Particularly, we would need to know the configuration of the system clocks involved. Without it we would have no way to measure the accuracy of our approximations. Unfortunately no such data is currently available. Instead we use simulated log files to help gauge each algorithm's effectiveness. We wrote a log file simulator to generate log files with various temporal characteristics.

4.1.1 Temporal Variables

Each host in our simulator has a set of variables associated with it:

- Arrival rate - The rate at which alerts are received at the log file host from a sensor (equivalently, the rate at which the sensor generates alerts). An exponential random variable controls arrivals; the rate of this random variable is determined by a sinusoidal function. The sinusoidal behavior of the rate gives the alerts a burstier distribution. We control arrival rates in the experiments via the amplitude of this sinusoid. We call the amplitude the arrival rate scaler. The default value is 1.

- Clock skew - The skew associated with a sensor's system clock. By default, a value between -300 and 300 will be chosen at random.
- Clock drift - The drift associated with a sensor's system clock. For all the experiments, each host had a drift chosen randomly between -10^6 and 10^6.
- Constant delay - The minimum delay required to send an alert from the sensor to the log file host. The default is a value between 0 and 1 chosen at random.
- Variable delay - The variable portion of delay added to the constant delay for each host. Variable delay is controlled by a sinusoid (to mimic the periodic fluctuations of Internet traffic density). The amplitude of this sinusoid is controlled to increase, or decrease, the variability of the delay and is called the variable delay scaler. The default value is 0 (i.e., no variable delay).

Constant delay, variable delay, and arrival frequency affect the accuracy of our time stamp synchronization algorithms. Our simulations do not include clock drift. Times can be thought of in terms of seconds, though the simulations are independent of scale.

4.2 Experiments

We conducted several experiments with simulated log files to help understand the limitations of our algorithms. For each experiment, we specify a free variable that iterates over a set of values. We generate one log file for each value in the set. The other parameters for each host are fixed. Table 4.1 summarizes the parameters for each experiment. The experiments are split into categories based on the free variable. The first has varying arrival rates with constant delays (VAR), the second varying arrival rates with variable delay (VARD), the third varying constant delays (CDC), and the last varying delays (VD). The number in each label corresponds to the number of hosts for which the free variable is varied; the remaining hosts have the default value for that parameter (as specified previously). The free variable is in bold and contains a range plus an increment value (in paranthesis). The rest of the parameters are either specified or a range is given from which a value is chosen randomly. The experiments are designed to identify how changes in one parameter impact on the performance of the three algorithms considered.

The metric utilized to measure the correctness of an algorithm's synchronization solution for a given log file is *average time stamp error*. This metric is simply a weighted average of the differences between the actual relative skew of each host and the correct offsets as specified by the synchronization solution. For each host h, let S_h represent the clock skew as given by the

Experiment	VAR1	VAR5	VAR10	VARD5
Number of hosts	10	10	10	10
Hosts varied	1	5	10	5
Arrival scaler	**[.01,1] (.01)**	**[.01,2] (.02)**	**[.01,2] (.02)**	**[.01,1] (.01)**
Skew	[-300,300]	[-300,300]	[-300,300]	[-300,300]
Constant delay	[0,1]	[0,1]	[0,1]	[0,1]
Variable delay scaler	0	0	0	[0,1000]

Experiment	VARD10	CDC1	CDC5	VD1
Number of hosts	10	10	10	10
Hosts varied	10	1	5	1
Arrival scaler	**[.01,1] (.01)**	1	1	1
Skew	[-300,300]	[-300,300]	[-300,300]	[-300,300]
Constant delay	[0,1]	**[0,1000] (10)**	**[0,1000] (10)**	[0,1]
Variable delay scaler	0	0	0	**[0,1000] (10)**

Experiment	VD5	VD10
Number of hosts	10	10
Hosts varied	5	10
Arrival scaler	1	1
Skew	[-300,300]	[-300,300]
Constant delay	[0,1]	[0,1]
Variable delay scaler	**[0,1000] (10)**	**[0,1000] (10)**

Table 4.1: Experiments conducted to analyse affects of various variables. Bold parameters represent the free variable in the experiment with the increment given in parenthesis.

configuration in use when the log file was created. Let O_{hr} be the correction offset as specified by the algorithm's synchronization solution relative to a reference host r. Let n_h be the number of entries in the log file that are from host h and let n be the total number of entries in the log file. Then the average error of the synchronization solution is

$$e = \sum_h \frac{n_h}{n} |S_r - S_h - O_{hr}|$$

where the summation is over all hosts h in the log file.

4.3 Results

Figures 4.1 through 4.10 graphically display the results of running the experiments. The x-axis of each graph represents the free variable. The y-axis represents the average error of the log file. Several of the results yield interesting conclusions. We discuss how the results reflect on the effects of each free variable in turn.

A R ( ). Experiments VAR1, VAR5, and VAR10 measured the effects of differing arrival rates on the algorithms' performance. Since we fix the number of entries in each simulated log file, changing the arrival rate has two affects. First, it changes the time spanned by the log file. For example, log files generated with an arrival rate scaler of 0.01 spanned over two years while an arrival rate scaler of 2.0 generated log files spanning a little over two months. Since drift accumulates over time, it adds significantly more error when lower arrival rates are utilized. It's apparent from the results of VAR1 and VAR5 that if only one, or a few hosts, have lower arrival rates then synchronization is not significantly affected. In VAR5, the outliar for M -D -S resulted from poor choice of reference host: the reference host's minimal event happened to be one heavily affected by drift (i.e., it was at the end of the log file). In VAR10, where all hosts have lower arrival rates, we see a distinct trend that error decreases with more frequent arrivals. For the type-two log file algorithms, this decrease in error is due to a decrease in the effect of drift. For the type-one algorithm, this is due to an increase in the tightness of intervals calculated. Drift does not have as significant an effect on the I -S algorithm since the good prior state heuristic discards entries later entries that are heavily affected by drift.

A R ( ). Experiments VARD5 and VARD10 measured the effects of differing arrival rates when entries are affected by large variable delays. Both the type-one algorithm and the stastical type-two algorithm do poorly regardless of arrival rate. This implies that the errors due to variable delay greatly outweigh those from arrival rate differences. In this case, again, the minimal-delay type-two algorithm performs excellently.

C D . Experiments CDC1 and CDC5 measured the effects of increased constant delay. In general, constant delay is going to translate directly into error since all the algorithms take a 'minimal delay' approach (i.e., they assume minimal delay and thus lump the constant delay in with skew). The two graphs reflect this, showing error to grow linearly with constant delay. Figure 4.7 shows that the selection of reference host matters: it happened that the statistical algorithm (green) chose as reference host the one host with constant delay. Thus all relative skews calculated against it included that error (pushing these errors much higher than if another host had been selected). The other algorithms would have suffered the same fate if they choose the 'wrong' reference host.

V D . Experiments VD1, VD5, and VD10 measured the effects of increasing the magnitude of variable delay. When only one or five of the hosts had large variable delays, the type-one algorithm and the stastical type-two algorithm performed similarly. When all ten hosts had large variable delays, the stastical type-two algorithm performed much better. The minimal delay type-two algorithm performed exceptionally in all cases, clearly doing better than the other algorithms.

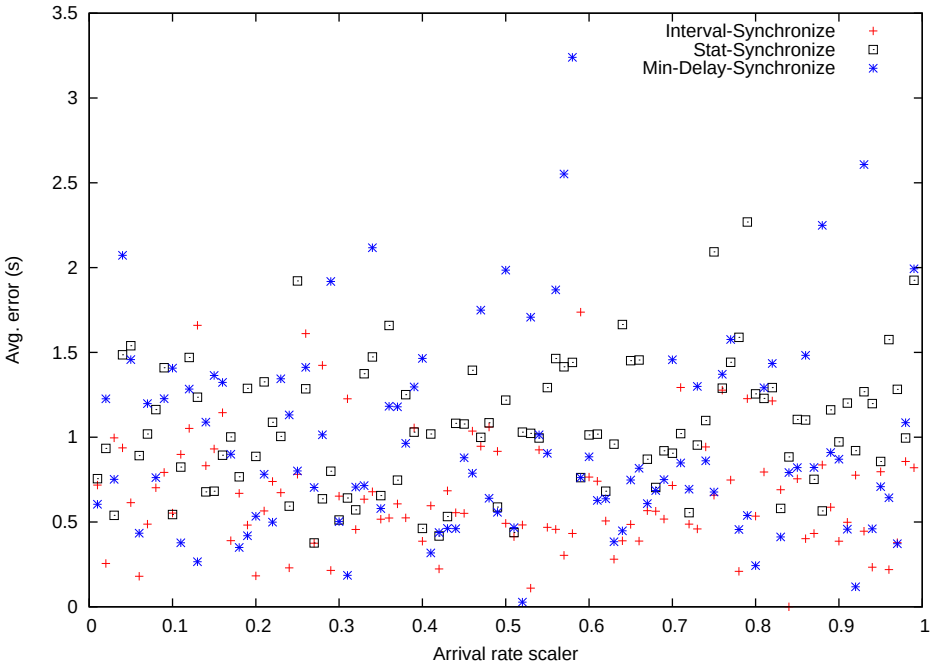

Figure 4.1: Results of experiment VAR1. In the experiment, arrival rates for one host is varied from low to high. All hosts had constant delays.

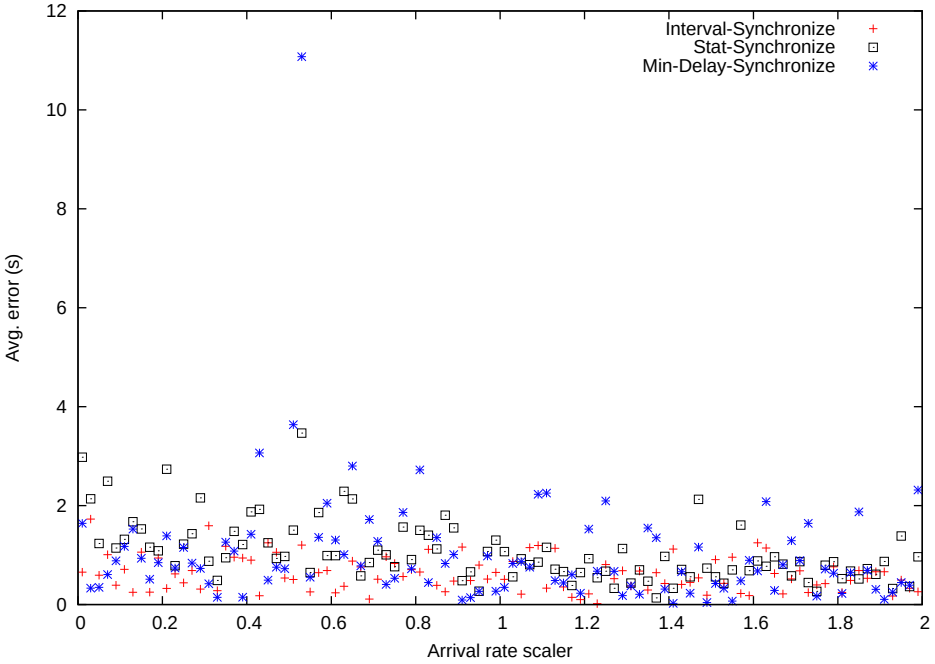

Figure 4.2: Results of experiment VAR5. In the experiment, arrival rates for five hosts are varied from low to high. All hosts had constant delays.

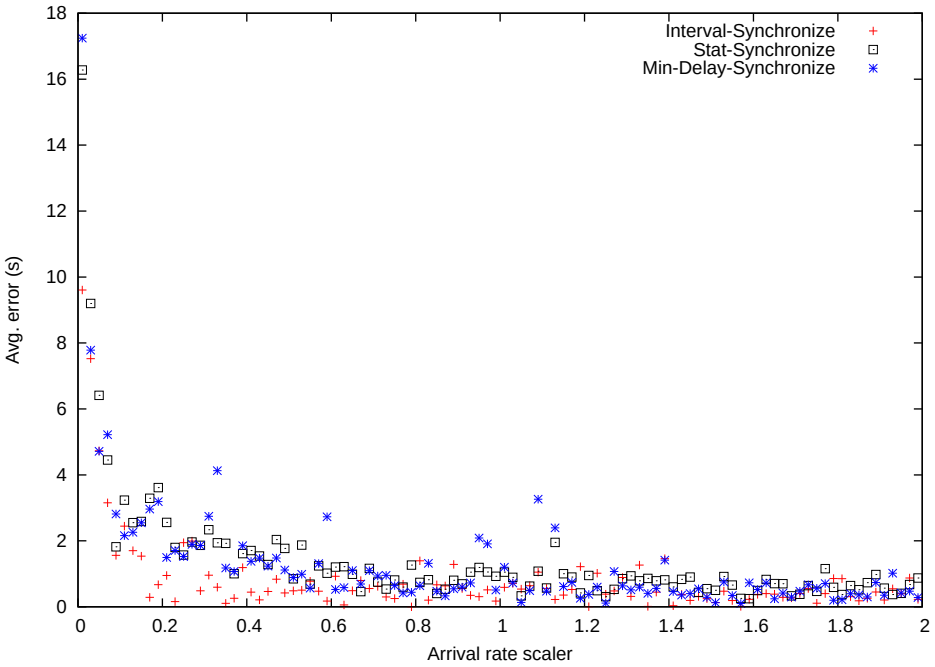

Figure 4.3: Results of experiment VAR10. In the experiment, arrival rates for ten hosts are varied from low to high. All hosts had constant delays.

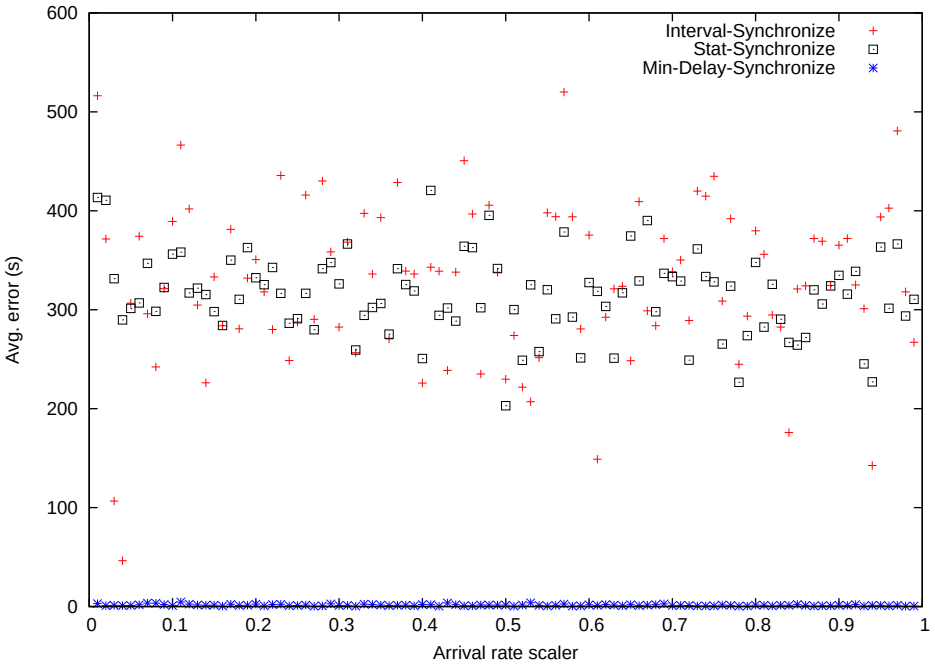

Figure 4.4: Results of experiment VARD5. In the experiment, arrival rates for five hosts are varied from low to high. All hosts had large, variable delays.

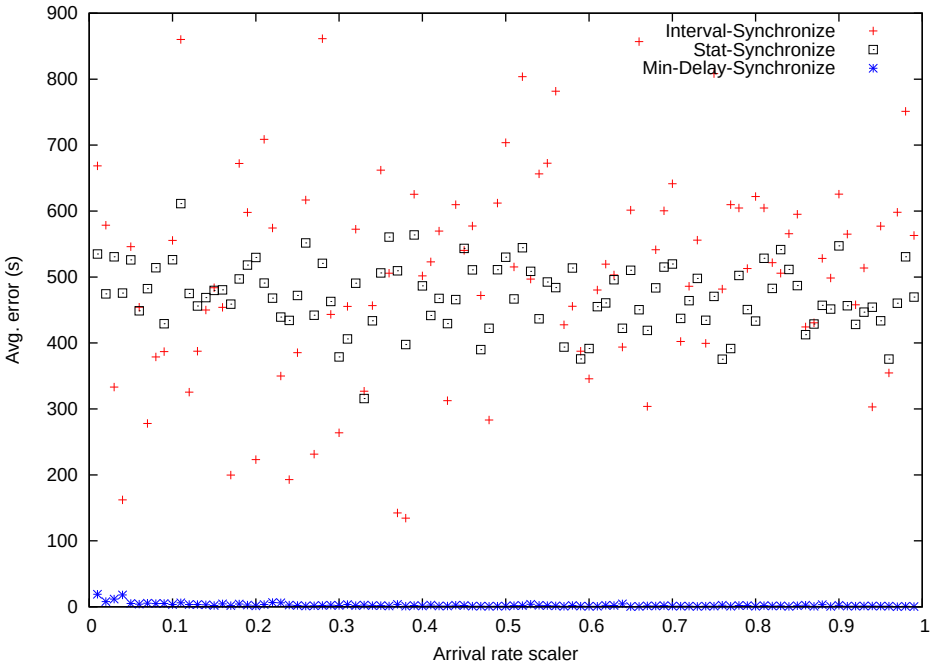

Figure 4.5: Results of experiment VARD10. In the experiment, arrival rates for ten hosts are varied from low to high. All hosts had large, variable delays.

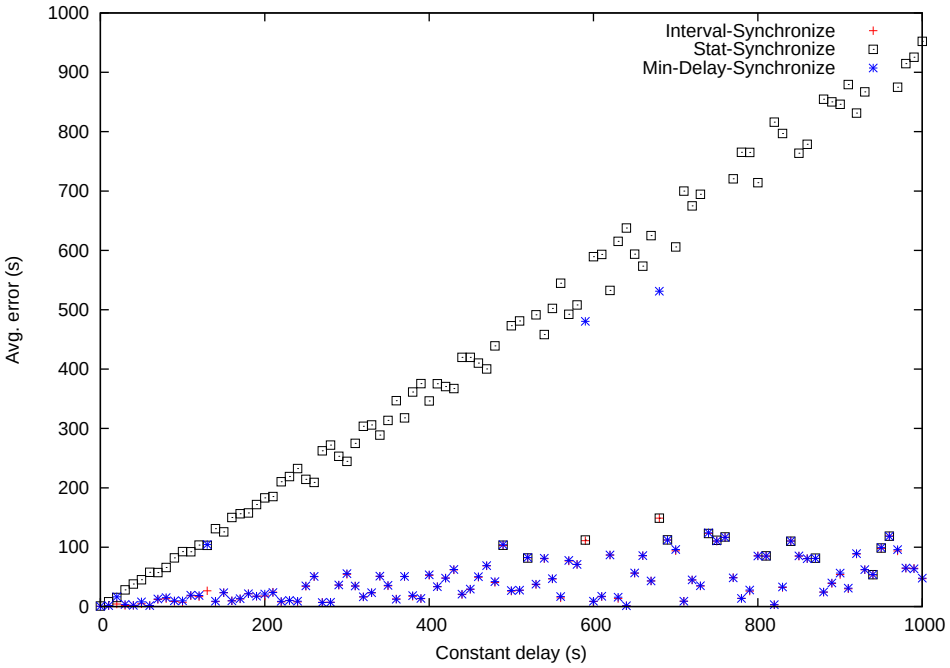

Figure 4.6: Results of experiment CDC1. In the experiment, the constant delay for one host is varied from low to high.

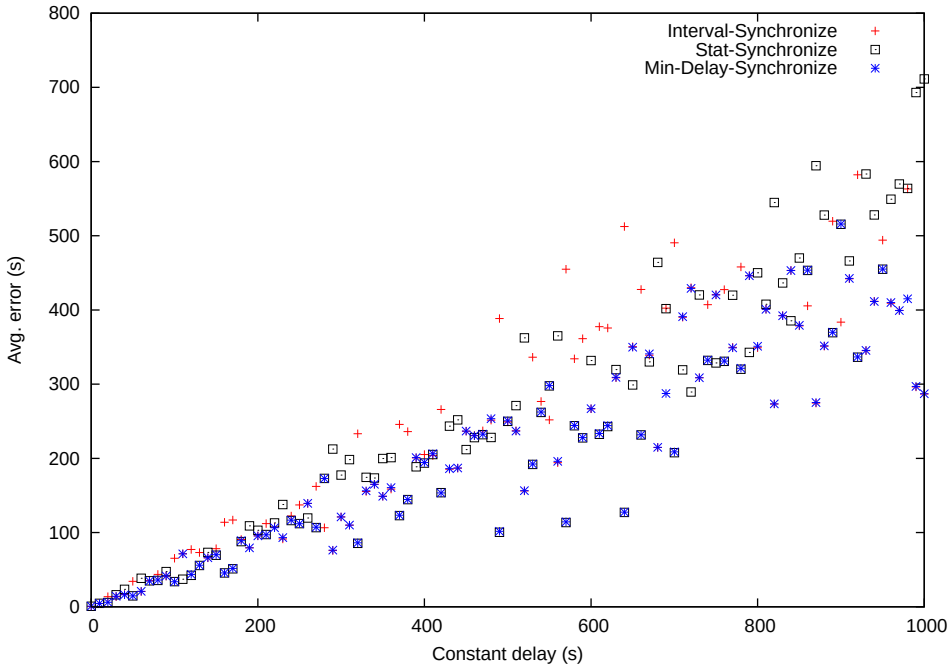

Figure 4.7: Results of experiment CDC5. In the experiment, the constant delay for five hosts is varied from low to high.

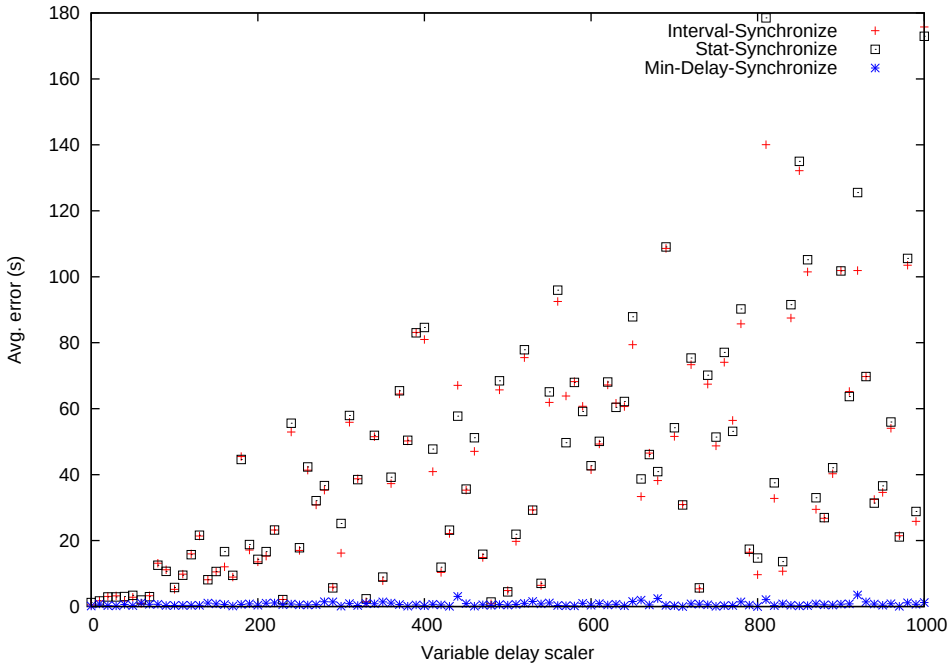

Figure 4.8: Results of experiment VD1. In the experiment, the magnitude of variable delay spikes for one host is varied from low to high.

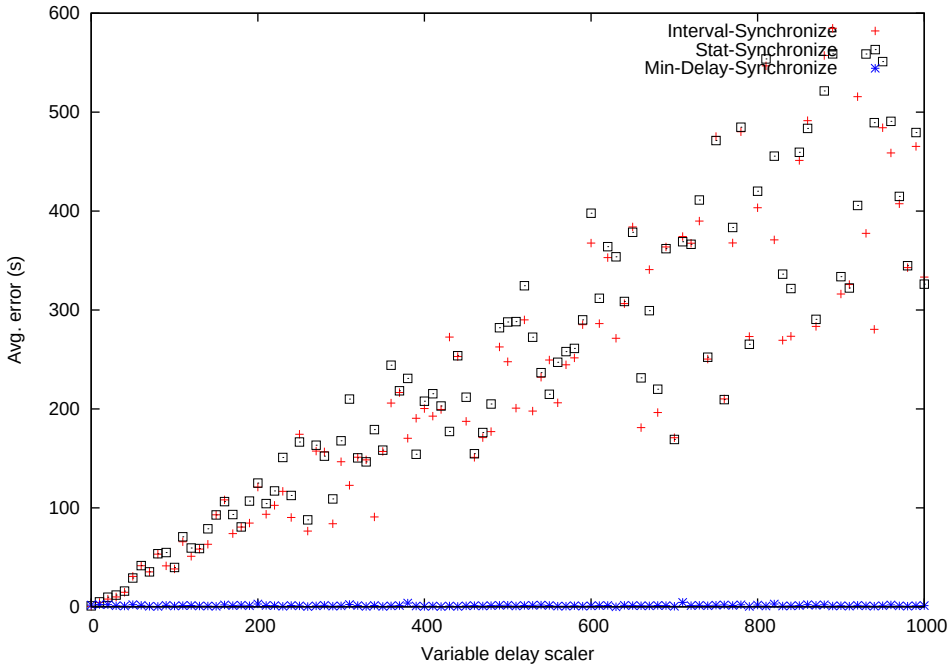

Figure 4.9: Results of experiment VD5. In the experiment, the magnitude of variable delay spikes for five hosts is varied from low to high.

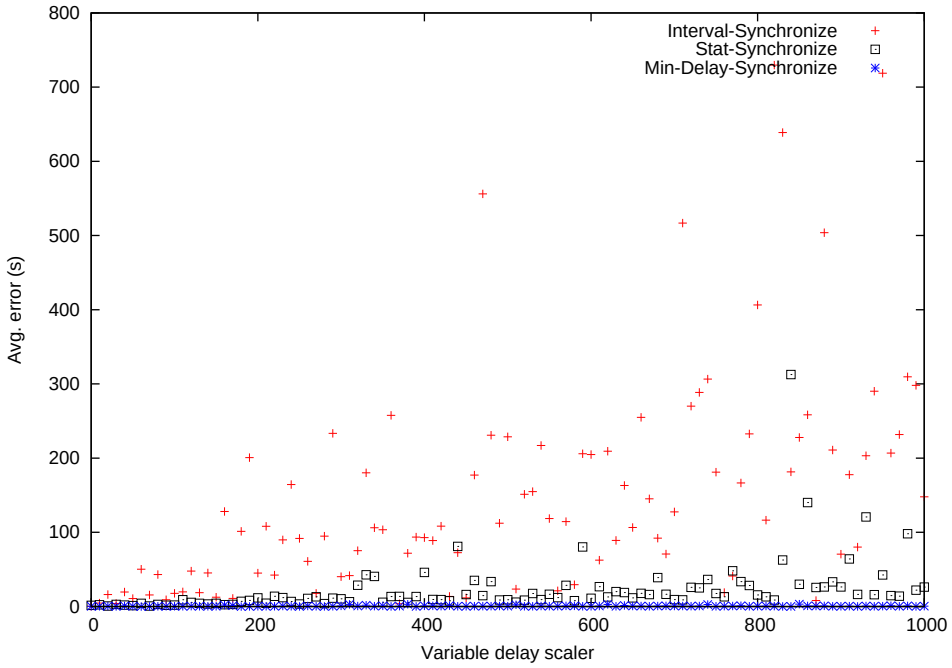

Figure 4.10: Results of experiment VD10. In the experiment, the magnitude of variable delay spikes for ten hosts is varied from low to high.

Chapter 5

Related Work

Prior work can be divided into several broad categories: sensor alert analysis, clock synchronization algorithms, and time stamp synchronization algorithms.

5.1 Sensor Alert Analysis

An important theme in IDS research has been in reducing the number of alerts security auditors must handle [28, 11, 6, 19, 7, 29, 18]. Particularly, state of the art IDS systems report alerts for "low-level" events (e.g., specific network activity), and in practice this results in alert explosion: security auditors are faced with an overwhelming number of alerts (many of which are most likely false positives). Researchers have developed methods for grouping alerts that are redundant in some way.[1] In addition, methods for creating synthetic alerts based on the inferred relationships of multiple low-level alerts (correlation) have been explored. Algorithms for grouping and correlating alerts require correct temporal information: in all prior work we're aware of at least order is essential for proper functioning. When only one IDS's alerts are considered, correct temporal information is trivially acquired.

Another vein of research has been in utilizing distributed, possibly heterogenous IDS's to acheive better intrusion incident detection [27, 26, 20]. In these settings, alert explosion is an even larger problem since multiple IDS's will generate distinct alerts for the same observed events. Thus, correlation and grouping of alerts is also very important. Again correct temporal information is required for grouping and correlation, but not trivially acheived as in the previous setting. Serrano identified this problem and briefly addressed it by loosening the temporal constraints within his correlation algorithms [26]. In other works, correct temporal information is implicitly assumed.

[1] In the literature, this is often referred to as aggregation of alerts. However, we already utilize the term aggregation to describe the collection of alerts from multiple sources. We therefore stick to using the term grouping.

5.2 Clock synchronization

Message timing in distributed systems was first considered in Lamport's seminal work [14]. Particularly important are his discussion of logical clocks and causality. Numerous other projects followed; most had proposed mechanisms (such as vector and matrix clocks) for determining causal relationships [12, 21].

We utilize intervals to estimate temporal information. Marzullo and Owicki and later Schmid and Schossmaier developed interval-based clock synchronization algorithms [16, 24, 25]. Instead of representing local time by a single integral value they represent it with an accuracy interval that contains the actual time at which the event occurred. This powerful representation of local time allows for more accurate and precise synchronization. They introduced a priori clock synchronization mechanisms for distributed systems utilizing these local interval clocks. The popular Network Time Protocol utilizes such accuracy intervals in its clock synchronization algorithm [17]. In [15], Liebig et al. describe methods for event composition in time-dependent distributed systems. They use accuracy intervals provided by NTP as time stamps within their system, and discuss the ramifications of this approach. Although our approaches are similar in that we also utilize intervals to estimate temporal information, their intervals are defined using more precise measurements of error only available when performing a priori synchronization. They do not consider the settings in which time stamp synchronization is needed.

5.3 Time stamp synchronization

Previous research in time stamp synchronization stems mostly from the distributed systems community. In their settings, one is given access to a number of local event traces created at distinct systems. These event traces are given time stamps from local system clocks that are not synchronized. The event traces are aggregated after they are generated, and the goal is then to determine a global time base for them. One application is distributed monitoring: a monitor utilizes such algorithms to help developers temporally relate debugging events in the system, but without the need for a priori clock synchronization.

Duda et al. first proposed algorithms for tackling this problem [9]. They assume that the event traces contain pairs of time stamps (t_i, θ_i) where t_i is the time a message was sent from a host and θ_i is the time the message was received on another host. In this case t_i would be in one event trace and θ_i in another. This sent/receive semantics yielded convenient causal dependencies between such pairs. With such pairs of points, regression analysis or convex hull estimates can be utilized to approximate the initial skews and the drift rate between the two hosts' clocks. Ashton later built on

this work by adding estimates for minimum delay between the two hosts and a two-point method for calculating the initial skew and drift rate [4].

Hofmann and Hilgers [22] proposed a more general algorithm for determining a global time base from such traces. Generalizing on Duda et al. 's assumptions, they assume knowledge of some set of causal dependencies between events in the various local traces. Using these dependencies, they create intervals which are guaranteed to include the correction offset between the two involved host clocks. This is very similar to our approach for type-one log files: our approach can be viewed as an adaptation of their algorithm to a type-one setting where we utilize ordering as approximate causal dependencies. They likewise utilize shortest-path analysis to refine the intervals. Because of their exact causal dependencies, however, they did not have to deal with infeasible shortest-paths and thus our exploration of the ties between feasibility and delay is novel. They also describe how to utilize simple linear regression analysis to approximate drift.

In all of this prior work, the algorithms require temporal information about messages passed both to and from a pair of hosts. In Duda et al. the messages must be passed directly between the pair of hosts; in Hofmann and Hilgers the messages can be indirectly passed. Either way, if the communication architecture is many hosts to a single one (many-to-one), with no communication in the other direction, then these algorithms will not be applicable.

Chapter 6

Conclusion

6.1 Future Work

Several aspects of time stamp synchronization would benefit from further study. To begin with, a more extensive analysis of how to best choose a reference host could yield even more accurate synchronization. Extending the model to include more general temporal phenomena could increase the robustness of our approach. For example, including non-monotonically increasing clocks (i.e., allowing clock resets) could be very important. Allowing knowledge of causal dependencies between alerts might benefit synchronization algorithms in some settings. Finally, investigating the usefulness of this approach in orthogonal areas, such as sensor networks, is left as open work.

6.2 Conclusion

State of the art analysis techniques and algorithms for distributed sensor alerts require synchronized temporal data. In practice synchronization can be difficult to achieve: even if a priori clock synchronization algorithms are supposed to be utilized, misconfiguration or poor oversight can result in time stamps that are not tightly synchronized. This problem has received little attention from the research community even though it can render a large body of analysis techniques and algorithms essentially useless. The only previous work addressing this problem requires modification of existing analysis algorithms.

We proposed using a posteriori time stamp synchronization to solve this problem and rigorously investigate the approach. We described a model of temporal data in distributed sensor logs, that can also be applied to any many-to-one message passing system. The model gives us insight into the inherent ambiguity between delay and skew: we showed that no general algorithm can precisely synchronize time stamps. In spite of this general result, we gave two best-effort

approximation algorithms that can synchronize time stamps accurately in realistic settings. Even though our algorithms were developed under the assumption of negligible drift, simulation shows that typical clock drifts do not significantly affect accuracy. Our simulations show that a posteriori synchronization can be a valuable tool for security auditors.

Bibliography

[1] CAIDA, the Cooperative Association for Internet Data Analysis, 2005. `http://www.caida.org/`.

[2] SANS Internet Storm Center, 2005. `http://isc.sans.org/`.

[3] Symantec DeepSight, 2005. `http://tms.symantec.com/`.

[4] P. Ashton. Algorithms for off-line clock synchronization. *Technical Report TR COSC 12/952 Department of Computer Sciences University of Canterbury*, December 1995.

[5] Thomas T. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to algorithms*. MIT Press, Cambridge, MA, USA, 1990.

[6] F. Cuppens. Managing alerts in a multi-intrusion detection environment. In *ACSAC '01: Proceedings of the 17th Annual Computer Security Applications Conference*, page 22, Washington, DC, USA, 2001. IEEE Computer Society.

[7] Herv Debar and Andreas Wespi. Aggregation and correlation of intrusion-detection alerts. In *RAID '00: Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection*, pages 85–103, London, UK, 2001. Springer-Verlag.

[8] Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artif. Intell.*, 49(1-3):61–95, 1991.

[9] A. Duda, G. Harrus, Y. Haddad, and G. Bernard. Estimating global time in distributed systems. *7th International Conference on Distributed Computing Systems (ICDCS'87)*, September 1987.

[10] Jeremy Elson, Lewis Girod, and Deborah Estrin. Fine-grained network time synchronization using reference broadcasts. *SIGOPS Oper. Syst. Rev.*, 36(SI):147–163, 2002.

[11] F. Cuppens and A. Miege. Alert Correlation in a Cooperative Intrusion Detection Framework. In *SP '02: Proceedings of the 2002 IEEE Symposium on Security and Privacy*, page 202, Washington, DC, USA, 2002. IEEE Computer Society.

[12] Colin Fidge. Logical time in distributed computing systems. *Computer*, 24(8):28–33, 1991.

[13] Saurabh Ganeriwal, Ram Kumar, and Mani B. Srivastava. Timing-sync protocol for sensor networks. In *Proceedings of the first international conference on Embedded networked sensor systems*, pages 138–149. ACM Press, 2003.

[14] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[15] C. Liebig, M. Cilia, and A. Buchmann. Event composition in time-dependent distributed systems. In *Proceedings of the Fourth IECIS International Conference on Cooperative Information Systems*, page 70. IEEE Computer Society, 1999.

[16] Keith Marzullo and Susan Owicki. Maintaining the time in a distributed system. In *Proceedings of the second annual ACM symposium on Principles of distributed computing*, pages 295–305. ACM Press, 1983.

[17] David L. Mills. Improved algorithms for synchronizing computer network clocks. *IEEE/ACM Trans. Netw.*, 3(3):245–254, 1995.

[18] Peng Ning, Yun Cui, Douglas S. Reeves, and Dingbang Xu. Techniques and tools for analyzing intrusion alerts. *ACM Trans. Inf. Syst. Secur.*, 7(2):274–318, 2004.

[19] Peng Ning and Dingbang Xu. Learning attack strategies from intrusion alerts. In *CCS '03: Proceedings of the 10th ACM conference on Computer and communications security*, pages 200–209, New York, NY, USA, 2003. ACM Press.

[20] O. Dain and R.K. Cunningham. Building scenarios from a heterogeneous alert stream. In *Proc. of the 2001 IEEE Workshop on Info. Assurance and Security*, 2001.

[21] Michel Raynal and Mukesh Singhal. Logical time: Capturing causality in distributed systems. *Computer*, 29(2):49–56, 1996.

[22] Richard Hofmann and Ursula Hilgers. Theory and tool for estimating global time in parallel and distributed systems. In *Proceedings of the Sixth Euromicro Workshop on Parrallel and Distributed Computing, PDP'98*, pages 173–179, January 1998.

[23] Kay Romer. Time synchronization in ad hoc networks. In *Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing*, pages 173–182. ACM Press, 2001.

[24] Ulrich Schmid and Klaus Schossmaier. Interval-based clock synchronization. *Real-Time Syst.*, 12(2):173–228, 1997.

[25] Ulrich Schmid and Klaus Schossmaier. Interval-based clock synchronization with optimal precision. *Inf. Comput.*, 186(1):36–77, 2003.

[26] Alfredo Serrano. Integrating Alerts from Multiple Homogeneous Intrusion Detection Systems. Master's thesis, North Carolina State University, May 2003.

[27] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, and D. GrIDS - a graph based intrusion detection system for large networks. In *In Proceedings of the 19th National Information Systems Security Conference*, volume 1, pages 361–370, 1996.

[28] Steven J. Templeton and Karl Levitt. A requires/provides model for computer attacks. In *NSPW '00: Proceedings of the 2000 workshop on New security paradigms*, pages 31–38, New York, NY, USA, 2000. ACM Press.

[29] Bo Sun, Kui Wu, and Udo W. Pooch. Alert aggregation in mobile ad hoc networks. In *WiSe '03: Proceedings of the 2003 ACM workshop on Wireless security*, pages 69–78, New York, NY, USA, 2003. ACM Press.

[30] John R. Vig. Introduction to quartz frequency standards. Technical Report Technical Report SLCET-TR-92-1, Army Research Laboratory, Electronics and Power Sources Directorate, October 1992. Available at http://www.ieee-uffc.org/freqcontrol/quartz/vig/vigtoc.htm.