Abstract—Running clock synchronization protocols over packet based networks introduces a considerable challenge, since clock accuracy is highly sensitive to the network latency behavior. As packet based networks are becoming the common transport for most applications requiring clock synchronization, accuracy requirements are becoming increasingly stringent. In this paper we introduce a novel approach that uses multiple communication paths between the master and slave clocks to improve the clock accuracy without increasing the total rate of protocol messages. We show that the multi-path approach can also be used to reduce the time error caused by asymmetric communication paths. We present simulation results that demonstrate the effectiveness of our approach.

Keywords: multiple paths, slave diversity, IEEE 1588, PTP, NTP, time synchronization, clock synchronization.

1 INTRODUCTION

Maintaining clock synchronization over packet based networks is quite challenging. Time synchronization protocols use protocol messages to convey time information between clocks. Depending on the nature of the underlying network, theses packets can be subjected to variable network latency. However, as time sensitive applications evolve, accuracy requirements are becoming increasingly stringent.

Antenna diversity is a well-known scheme in wireless communications, that uses two or more antennas to improve the quality of a received signal. In this paper we introduce an analogous approach in clock synchronization protocols. Since master and slave clocks are often connected through more than one path in the network, as shown in Figure 1, we suggest to run the time synchronization protocol over all available paths. We shall show that this approach can significantly increase the clock accuracy even without increasing the total rate of protocol messages in the network. Furthermore, we show that in the presence of asymmetric communication paths the multiple path approach can reduce the time bias caused by the asymmetry.

The usage of timing information from multiple clock sources to improve accuracy is applied in the Network Time Protocol [2]. A somewhat similar paradigm is to run a time synchronization protocol with all neighbors in the network, and to combine this information to estimate the time and frequency (e.g. [9]).

Redundancy in clock synchronization protocols has also been widely discussed in the literature. Redundant master clocks are used in [16] for protection. Redundancy is also a major issue in power substation automation; High Availability Seamless Redundancy (HSR) is a ring redundancy protocol, where all traffic, including time synchronization traffic, is duplicated and sent on both directions of the ring. In this context, the challenge of integrating path redundancy with time synchronization protocols was thoroughly analyzed (e.g., in [21], [19], [8]). Note that these analyses presented the redundancy as a challenge for the synchronization protocol, and not an advantage.

A different form of redundancy was proposed in [16], using a hardware oriented method utilizing two slave clock blocks running at two different clock frequencies to improve the clock accuracy. The analysis in [15] suggested to allocate redundant paths for time synchronization, and when the packet delay variation in the currently used path exceeds a certain degree, to switch-over to an alternative path.

From a security perspective, redundant paths [17] or redundant clock sources [2] have been suggested as a means to mitigate security attacks on synchronization protocols.

The problem of clock synchronization protocols in the presence of asymmetric paths has also been thoroughly analyzed. Various methods have been proposed to mitigate the asymmetry problem. The methods presented in [9] and [10] used delay measurements through cyclic paths in the network to estimate the delay asymmetry. Another method ([11], [12]) is to use probe packets at various packet lengths, allowing to detect asymmetry in the path bitrates. The method in [13] measures delay asymmetry at the physical layer of copper based Ethernet.

The contributions of this paper are as follows:

- We introduce a novel approach called slave diversity which utilizes multiple paths between a master and a slave clock to attain high clock accuracy. In this approach, a time synchronization protocol is run between the two nodes concurrently through multiple paths. Combining the information received from each of the paths enables a significant improvement to the clock accuracy compared to the conventional paradigm using a single path.

- We show that when the network paths are asymmetric and there is no correlation between the asymmetry values at different paths, the multi-path approach reduces the time error caused by the unknown delay asymmetry.

Figure 1. Slave Diversity
We propose three slave diversity algorithms, and analyze the effectiveness of these algorithms by simulation.

We discuss several network delay models, and perform our analysis under these models.

The remainder of this paper is organized as follows. Section 2 presents the network delay models we use in our analysis. The slave diversity approach is described in Section 3, as well as three slave diversity algorithms. Section 4 presents and discusses the simulation results. Section 5 discusses how the multi-path approach affects the time synchronization protocol. Concluding remarks are given in Section 6.

2 NETWORK DELAY MODEL

We assume the clock synchronization protocol takes place between two nodes, a master and a slave. The master is assumed to have an accurate time source, and uses protocol messages to distribute its time to the slave. The slave uses the information from these messages to estimate the master’s time. One of the key factors that affect the accuracy of a clock synchronization protocol is the latency of the underlying network. Therefore, we carefully consider several network delay models.

The impact of network latency on time synchronization protocols has been previously analyzed (e.g. [3]). There is also quite a bit of work in the literature about how to model the behavior of network latency. A common approach is to model the network latency using a Gamma distribution ([4], [6], [5]). Using other distributions such as an exponential or an Erlang [4] has also been proposed.

A common approach when modeling the network latency is to model it as a random white noise, i.e., to assume that two latency measurements at two different times are independently distributed. However, it has been shown that network latency is typically not white, i.e., that a correlation exists between network latency measurements taken at adjacent times [6]. This correlation is also analyzed in the frequency domain in [4], showing that network latency has a Gamma distribution, but also includes additional low frequency components. Other analyses show that the latency through network routers has a bursty nature ([6], [7]), and typically has a “sawtooth” curve.

The best practice of clock synchronization protocol deployment is to assign the highest possible priority to time protocol packets, which significantly reduces the network latency. Moreover, high priority queues are not likely to be subjected to high bandwidth traffic and the congestion effects discussed in [6] and [7]. It is reasonable to assume that some latency jitter can be caused by collisions, i.e., multiple high priority packets forwarded by the same switch or router at the same time. Thus, intuitively the random white distribution is suitable for protocol traffic that is forwarded with high priority. However, in many deployment scenarios protocol packets are transported over a commercial provider network, where the customer has no control over the priority of this traffic. In these cases, protocol packets have to compete with other high bandwidth traffic, and the congestion effects mentioned above significantly affect the latency behavior. Thus, when protocol packets cannot be assigned high priority, the white noise model does not necessarily capture the bursty nature of high bandwidth traffic.

Commercial network emulators typically support both white and non-white latency profiles. For example, Calnex Paragon-X [23] supports white distributions such as Gamma and Gaussian, as well as non-white modes such as a sawtooth curve. Netem [23] is an open source network emulator, that includes an explicit mode that creates a correlated latency distribution, creating a correlation between two adjacent latency values.

We analyze the network latency in a discrete time domain, where \( L_i[n] \) is the network latency from the master to the slave through path \( i \) during time period \( n \). The discrete timeline captures the fact that time synchronization protocol messages ([1], [2]) are typically invoked periodically, and thus we perform a discrete-time analysis according to the protocol’s transmission period.

For simplicity we assume in our analysis that the latency distribution is identical for the \( N \) paths, and that it is identical on the forward and reverse directions. Note that this assumption can easily be removed without affecting the algorithms we shall present.

Network latency is typically comprised of two components, deterministic and stochastic. Thus, we define \( L_i[n] = C + D_i[n] \), where \( C \) is the deterministic component, and \( D_i[n] \) is the stochastic component.

Our analysis is performed under 3 network delay models, as follows.

2.1 Random White Latency

The first model assumes independence between different measurements. We use an exponential distribution, which is a private case of the commonly used Gamma distribution, and thus \( D_i[n] \) has the following probability distribution function:

\[
P_{D_i}[x] = \begin{cases} \lambda e^{-\lambda x}, & \text{for } x \geq 0 \\ 0, & \text{for } x < 0 \end{cases}
\]

Intuitively, as mentioned above, the random white latency distribution captures the behavior of real-world latency when time protocol packets are assigned high priority.

2.2 Random Colored Latency

In the second model, we use an exponential distribution with correlation between measurements. We define time slots as follows: a time slot is a time interval of \( k \) periods. We define the network latency in this model to be fixed during each time slot. The latency at each time slot is selected according to the exponential distribution in Eq. (1).

Thus, when measured over a long period of time, the network latency measurements are exponentially distributed. However, the “stretching” effect of using time slots creates correlation between adjacent measurements. Intuitively, the network latency of a path is determined by the status of the queues in the intermediate routers and switches. Thus, when congestion builds up in the queues, it typically takes a while until the queues are emptied. This behavior is captured by the “stretching” effect.

2.3 Bursty Latency

The bursty model emulates the typical behavior of a network with occasional large bursts of data such as file downloads.

This model assumes that traffic in the network behaves according to the random white latency model (Section 2.1) most of the time, and occasionally there is a traffic burst that increases the network latency. Burst events in our model occur according to a Poisson distribution. Further details about the burst model we simulated are presented in Section 4.

2.4 Asymmetric Latency

In the previous subsections we assume that the forward and reverse paths are identically distributed. In the asymmetric latency model we try to capture the challenge of deterministic asymmetry between the forward and reverse paths. Delay asymmetry can be caused by various aspects of the network
topology, e.g., different cable lengths or different network equipment on the forward and reverse paths.

In order to analyze delay asymmetry we isolate it from other potential causes for slave time error, and define a very simple asymmetric latency model. The model described here is does not necessarily reflect real-life behavior of asymmetric paths, but allows us to demonstrate that in some cases slave diversity can be used to mitigate the effects of asymmetric paths. Each path $i$ is characterized by two constant latency values $L_{MS,i}$ and $L_{SM,i}$, corresponding to the master-to-slave path, and the slave-to-master path, respectively. In our simulation $L_{MS,i}$ and $L_{SM,i}$ are two identically distributed random variables that are selected once and remain constant throughout the run. Since the slave can only measure the round trip delay, once the two latency values are determined the difference $L_{MS,i}-L_{SM,i}$ inflicts a constant error in the slave time computation that is independent of the noisy behavior of the network latency. Note that the mean value of $L_{MS,i}-L_{SM,i}$ is 0, i.e., the mean value of $L_{MS,i}-L_{SM,i}$ is exactly the case where the path is symmetric.

2.5 The Intuition behind White Latency vs. Colored Latency

In this section we distinguish between random white latency and colored latency. In the single path paradigm, white latency is a series of independent latency values, $L[n]$ for $n=0,1,2,...$. Slave algorithms typically measure the delay and average it over time. According to the well-known Law of Large Numbers, the average of a sequence of independent and identically distributed random variables converges to the mean value of these random variables. Now if we consider two paths with identically distributed white latency, the slave algorithm measures $L_1[n]$ and $L_2[n]$ for $n=0,1,2,...$. Again, we have a sequence of independent and identically distributed measurements, and thus $M$ measurement values from the one path paradigm are equivalent to $M$ measurement values from two paths. Therefore, under random white latency we do not expect multiple paths to have an advantage over a single path protocol.

In the colored and in the bursty models latency changes are rather slow. Intuitively, if there unusual congestion occurs in the network causing high latency in one of the paths, it takes a while until the latency changes back to a normal value. It is easy to understand why in this case combining time information from different paths can help to compensate for the “noisy” data from the congested path.

3 Slave Diversity

The intuition behind the slave diversity approach is very similar to the one behind antenna diversity. Antenna diversity combines the signals from two or more antennas, to compensate for signals with low signal to noise ratio (SNR). Similarly, slave diversity uses timing information from two or more paths in the network to compensate for unstable latency behavior in one or more of the paths.

In this section we discuss three possible methods to implement slave diversity, i.e., to combine the information from the $N$ paths to a single time of day value. In our analysis we assume the existence of a servo algorithm, which uses timestamp information from received protocol messages to compute the time of day, and possibly also the clock frequency. The exact implementation of the servo algorithm is independent of the slave diversity algorithms, and is outside the scope of this paper. Typical servo algorithms take special care to avoid leaps in time, by updating the TOD in a graceful and continuous manner. We shall show that the slave diversity algorithms we present preserve this time continuity property.

We now present the three slave diversity algorithms, and then analyze their performance in Section 4.

3.1 Slave Diversity: Equal Gain Combining Algorithm

Equal-gain combining is one of the common approaches in antenna diversity, and simply combines the signals from all antennas, giving an equal weight to each one.

The equal-gain combining approach is applied to slave diversity using the algorithm described in Figure 2. The slave independently runs the time synchronization protocol with the master through each of the $N$ paths. Specifically, for each path $i$, the slave computes an independent Time Of Day, TOD, based on the protocol run over path $i$. At all times the combined TOD is simply the average of the $N$ time of day values.

| 1 | For each path $i$ |
| 2 | Maintain TOD, according to protocol messages in path $i$ |
| 3 | At any given time |
| 4 | CombinedTOD = average(TOD$_i$) |

Figure 2. Slave Diversity: Equal-Gain Combining Algorithm

As stated above, the algorithm in Figure 2 is independent of the servo algorithm used to compute TOD$_i$. In addition to the time of day, the servo algorithm may also compute and maintain the clock frequency corresponding to the protocol that is run through path $i$. The frequency of the combined TOD is then implicitly affected, and is derived from the frequencies of the $N$ paths. The TOD continuity property is preserved in the combining algorithm, since the averaging operation preserves the continuity of the individual TODs.

3.2 Slave Diversity: Switching Algorithm

The switching approach in antenna diversity chooses one of the antennas based on a signal to noise ratio (SNR) criterion.

For slave diversity we present a similar approach, that prefers the least “noisy” path, i.e., the path with the most stable latency behavior. The challenge is to define a metric for measuring the “noisiness” of a path. The metric we use is based on the difference between the last measured path delay from master to slave and the average path delay. Intuitively, if the last measured delay is significantly different than the average delay, it is fair to assume that the path is temporarily under heavy or irregular congestion.

The switching algorithm is described in Figure 3. The slave runs the synchronization protocol through each of the $N$ paths, and maintains for each path $i$:

- **LastDelay**: the value of the last measured one-way delay from the master to the slave. The one-way delay is typically computed as simply $\frac{1}{2}$ of the round-trip delay.
- **AvgDelay**: the average value of the measured one-way delays through path $i$. Time protocols use various filtering algorithm to average the delay measurements over time. The exact implementation of the averaging algorithm is independent of the diversity method, and is outside the scope of this paper.

At any given time, the slave chooses path $i$ with the minimal **LastDelay** as the primary path. The servo algorithm updates SwitchedTOD using only protocol messages received through the primary path, and ignores protocol messages received through other paths. Note that while **LastDelay** and **AvgDelay** are measured and computed for all paths, the SwitchedTOD is updated only based on information received from the primary path.
For each path i
2. Maintain AvgDelayi and LastDelayi, according to protocol messages in path i
3. At any given time
4. PrimaryPath = path i with min |AvgDelayi - LastDelayi|
5. Maintain SwitchingTOD according to protocol messages in PrimaryPath

Figure 3. Slave Diversity: Switching Algorithm

When the slave switches the primary path from path i to path j, the servo algorithm simply starts to use protocol messages from path j instead of message from path i. The switcher from i to j also implies that the servo algorithm uses the computed delay corresponding to path j instead of the delay for path i. Since the slave maintains a single time of day, SwitchingTOD, the switcher does not compromise the continuity of SwitchingTOD.

3.3 Slave Diversity: Dynamic Algorithm

The third algorithm we present takes a hybrid approach, where at any given time the slave maintains both the CombinedTOD and the SwitchingTOD, according to the algorithms described in Figure 2 and Figure 3, respectively. At any given time the slave’s state is either “combining” or “switching”, and the slave’s corresponding time of day is either CombinedTOD or SwitchingTOD, respectively.

0 Init: state = combined
1 At any given time
2 if state=combining
3 DynamicTOD = CombinedTOD
4 DelayDiff = |AvgDelayPrimaryPath - LastDelayPrimaryPath|
5 if |SwitchingTOD - CombinedTOD| > α · DelayDiff
6 SwitchingTOD = CombinedTOD
7 state = switching
8 else
9 DynamicTOD = SwitchingTOD
10 if state=switching for a period of SwitchingTimeout
11 foreach path i
12 TODi = SwitchingTOD
13 state = combining

Figure 4. Slave Diversity: Dynamic Algorithm

The state is initialized to “combined”, and is changed to “switching” when the condition on line 5 is satisfied. Intuitively, this condition detects bursty or noisy behavior in one or more of the paths; when one of the paths becomes significantly noisy it causes the CombinedTOD to be noisy as well, and thus significantly different than the SwitchingTOD, satisfying the inequality on line 5.

The assignment on lines 6 and 12 guarantee the continuity of the DynamicTOD when switching between the “combined” state and the “switching” state.

4 SIMULATION – RESULTS AND ANALYSIS

4.1 Simulation Setting

We simulated a system with two nodes running IEEE 1588, a master and a slave. The master and slave were connected through N paths, and we ran the simulation for various values of N: 1, 2, 4, 8, 16.

Master: The master ran the protocol independently through each of the paths independently.

Slave: We simulated each of the 3 slave algorithms presented in Section 3. The delay averaging algorithm and the servo algorithm were taken from the PTPv2d open-source package [22]. For the dynamic algorithm we heuristically chose α=5, as this seemed to produce the best results.

Sync interval: we used a Sync interval equal to N*10ms, i.e., the master transmitted 100 Sync messages per second regardless of the number of paths. This shows that the accuracy improves with N even without increasing the total number of Sync messages per second.

Delay_req interval: the slave transmitted Delay_req messages through each path independently. A Delay_req message was sent every 2-3 Sync messages received through the corresponding path. The slave randomly selected whether to wait 2 or 3 Sync intervals.

4.2 Slave Diversity Algorithm Comparison

Paths: All paths had an identical delay distribution, and this distribution was identical on the forward and reverse paths. We simulated each of the following delay distribution models described in Section 2:

- Random white latency: we used an exponential distribution with a mean \( \lambda = 10 \) usec, and with a constant \( C = 15 \) usec.
- Random colored latency: we used time slot of 320 ms, with the same parameters as the random white latency model: \( \lambda = 10 \) usec, and \( C = 15 \) usec.
- Bursty delay: in addition to the random white latency above, each path experienced occasional burst events. Each burst lasted for 10 seconds. The occurrence of the burst events followed a Poisson distribution, with a mean of 1 occurrence per 100 seconds. During each burst, the latency was comprised of two components:
  - Random white component with \( \lambda = 10 \) usec, and \( C = 15 \) usec.
  - Bursty component: a sawtooth curve where the latency fluctuates between 1ms and 0.5 ms.

Simulation run: in each run we simulated a 3 hour period, and analyzed the mean time error of the slave for the various cases. The mean time error is defined as the average of the absolute value of the offset between the master and slave clocks.

Simulation results: The results are illustrated below. The results for the three analyzed latency models are presented in Figure 5, Figure 6, and Figure 7. Each of the figures shows the mean time error as a function of the number of paths.

![Time Error under Random White Latency](Figure 5)
Figure 5 illustrates the results for the random white delay model. As expected, in this delay model increasing the number of paths does not reduce the time error.

Figure 6 shows the results in the random colored model, and demonstrates the significant decline of the time error as a function of the number of paths. The results show that the combining algorithm produces the lowest error rate compared to the other two slave diversity algorithms.

Figure 6. Time Error under Random Colored Latency

Finally, Figure 7 shows the results for the bursty delay model, with the switching algorithm producing the lowest time error.

Figure 7. Time Error under Bursty Latency

Comparing the slave diversity algorithms: the switching diversity algorithm produces the best results in the bursty model. Figure 9 provides a strong intuition to this result. The figure shows the TBD. As illustrated in the figure, when a burst occurs in one of the paths, the switching algorithm still provides a low error rate, as it uses the timing information from the more stable path. The combining algorithm, on the other hand, simply averages between the TOD obtained from the N paths. Thus, when a burst occurs in the bursty model, it significantly affects the average TOD used by the combining algorithm. The switching algorithm does not perform as well in the colored latency model, since our metric for selecting the best path, $|\text{AvgDelay} - \text{LastDelay}|$, is very effective for detecting bursts, but is less accurate in the presence of random fluctuations as in the colored latency model. The combining algorithm simply averages between the TOD values of the N paths, and thus performs best in the colored latency model.

The dynamic diversity algorithm tries to capture the best of both worlds. By default it uses the combining algorithm, and upon detecting bursty behavior it uses the switching algorithm. Thus, as Figure 6 and Figure 7 show, it is a fair compromise between the two algorithms.

4.3 Time Error in the Colored and Bursty Models

In this subsection we show the time error results during an interval of 100 seconds of two simulation runs using two paths. The two runs are illustrated in Figure 8 and Figure 9.

Figure 8 illustrates the time in the colored latency model. We show the results for the Combining algorithm, since as we saw in Figure 6, it provides the best accuracy in the colored model. Figure 8 also illustrates the time error when the slave uses only information from path 0, and the error when using only information from path 1. The figure clearly shows that by averaging the TOD values the combining algorithm also averages the error values, reducing the absolute error value.

Figure 8. Time Error in Combining Diversity under Colored Latency Model

Figure 9 illustrates the time error of the switching diversity algorithm under a bursty latency model. The figure illustrates the time errors of TOD0 and TOD1, corresponding to path 0 and path 1, respectively. The SwitchingTOD error is also shown, and illustrates that even when a burst occurs, the time error of SwitchingTOD remains low.

Figure 9. Time Error in Switching Diversity under Bursty Model

4.4 Asymmetric Latency

Path latency: the master-to-slave latency and the slave-to-master latency, $L_{\text{MS},i}$ and $L_{\text{SM},i}$, were both uniformly distributed between 10 usec and 50 usec.

Simulation runs: we ran 20 simulation runs, each for $N=1,2,4,8,16$. In each run $L_{\text{MS},i}$ and $L_{\text{SM},i}$ were independently selected at the beginning of the run, and remained constant throughout the run. The simulation was run with the combining algorithm$^1$.

Figure 10 illustrates the time error as a function of the number of paths. The results show that if the asymmetry values in the N paths are independently distributed with a mean value

$^1$ The switching algorithm selects the primary path based on the difference between the last delay and the average delay, and in the asymmetric latency test the delay is constant throughout the run, making the switching algorithm irrelevant.
of 0, using multiple paths reduces the uncertainty by cancelling out the asymmetry at the different paths.

![Figure 10. Time Error in Asymmetric Delay Test](image)

5 SLAVE DIVERSITY – PROTOCOL IMPLICATIONS

5.1 Impacts on the Synchronization Protocol

The slave diversity approach uses multiple paths between the master and slave nodes. Obviously, while path redundancy is very common in some network topologies, it is not present in others. Thus, slave diversity can selectively be applied in network topologies that provide redundancy, while the single path paradigm is applied in other cases.

Slave diversity also requires a method that allows the master and slave to agree on the number of paths used. Once the N paths are agreed upon, the network routing mechanisms must allow traffic between the master and slave to be forwarded consistently along the N paths.

While these issues are outside the scope of this paper, a clock synchronization protocol that deploys slave diversity must consider these issues.

5.2 Transparent Clocks

Throughout the paper we implicitly assumed that the master and slave are connected through a network “cloud” with variable latency. Specifically in the IEEE 1588 protocol [1], the network cloud may be comprised of Transparent Clocks (TC), allowing latency measurements to be performed more accurately. Using TCs significantly reduces the latency jitter, but does not eliminate it completely. Latency jitter can be caused for example by inaccurate timestamping at the TCs, or by inaccurate frequency of the TC clocks. Thus, the analysis in this paper applies to systems using TCs, although since the latency jitter in these systems is significantly lower than systems without TCs, the accuracy improvement in the multiple path approach would be appropriately lower. In future work, it would be interesting to further explore slave diversity in the presence of TCs.

6 CONCLUSION

We have shown that running a clock synchronization protocol concurrently through multiple communication paths can achieve higher accuracy than the conventional single path approach. The slave diversity approach is especially effective when the distribution of the network latency has low frequency components, or has a bursty nature. Slave diversity is also effective in reducing the time bias caused by asymmetric network paths. We presented three possible slave diversity algorithms, and discussed the tradeoffs.

In future work it would be interesting to further optimize the benefits of the slave diversity method by using other algorithms. Another interesting aspect for future work is defining the required features in the time synchronization protocol and in the underlying transport protocol in order to implement slave diversity over multiple paths.

REFERENCES

[8] A. Abdul, G. Ng, P. Lupas, “Integration of HSR and IEEE1588 over Ethernet networks”.
[21] K. Weber, C. Hoga, “High availability seamless automation ring (Draft IEC 62439-3) and IEEE 1588 time sync dependencies”.
[23] Netem, the Linux Foundation. http://www.linuxfoundation.org/collaborate/workgroups/networking/netem