TimeRemap: Stable and Accurate Time in Vehicular Networks

Ryad Ben-El-Kezadri, University of California, Los Angeles; University Pierre et Marie Curie; University of Manouba
Giovanni Pau, University of California, Los Angeles

ABSTRACT

One fundamental limit for synchronizing a packet trace is that the two measurement systems that can be used to timestamp the packets, the operating system and the network interface, have mutually exclusive properties. The OS clock can be stabilized over the long run but the OS timestamps are not accurate; the PHY timestamps are accurate but they show a drift. To leverage the best features of both systems, we propose to stabilize the OS clock with a GPS and to remap the PHY timestamps over the OS/GPS time. This method is well suited for challenged scenarios like VANETs as no signaling is needed between nodes. We apply our approach to VANET monitoring and evaluate it on packet traces collected from our testbed. The results show that our solution reduces the mean synchronization error to 3 μs.

INTRODUCTION

A key challenge of vehicular networks is to deploy a secure communication system on a wide scale. Faulty communication, session hijacking, and jamming will be major concerns for car manufacturers, forensic experts, and transportation authorities. Synchronization is a key element for building a traceable environment because it allows reconstruction of packet sequences on the channel. We believe that vehicular networks, especially vehicular ad hoc networks (VANETs), will integrate in each vehicle packet capturing systems for distributed sensing and tracking purposes. In the case of a road accident or cyber-attack, the packet traces collected by the witness vehicles will be retrieved by authorities and processed offline or near real time in a central location as evidence of the facts. Synchronization plays an important role in this process. Indeed, it allows merging the local information collected by the witness vehicles into a coherent view of the channel that preserves the message order.

We envision two scenarios: single hop and multihop. Figure 1a shows a single-hop configuration. In this example a communication problem between the driving assistance modules of two cars leads to an accident and destruction of the vehicles. The multihop scenario relates to an accident that involves multiple cars or a network attack that spans over multiple links (Fig. 1b). In both scenarios we assume the presence of observers (i.e., third party tracking nodes) that record the activity of the wireless channel. As the two figures show, the observers do not necessarily share the same local broadcast domain at the time of the incident.

Packet-based time synchronization relies on inter-contacts. As a result, packet-based techniques are only useful if nodes have been in contact shortly before/after or during the incident. This assumption does not hold for highly dynamic networks such as VANETs. In vehicular networks the tracking nodes that observe the same incident may never be in range; and even if they would be, the inter-contacts may not be long enough for the synchronization process to converge. If synchronization cannot be always achieved, not all the information collected by the cars can be merged. As a result, synchronization solutions that do not need explicit signaling stand out as the best candidates.

The accuracy of the reconstruction of the channel activity depends on the synchronization quality. If the synchronization process guarantees that two 802.11 local traces can be synchronized with an error less than 106 μs then the duplicate frames in the traces can be removed [1]. Fine-grained analyses (e.g., inferring interference between concurrent transmissions) require even more stringent guarantees [2]; frames must be synchronized to at least the precision of a physical layer slot time (20 μs for 802.11b/g and 8 μs for 802.11a/DSRC).

Synchronizing a VANET with off-the-shelf hardware is particularly challenging because no system can provide accurate and stable timestamps: the operating system (OS) clock can be stabilized over the long run but the OS timestamps are not accurate; the radio (physical [PHY]) timestamps are accurate but show a drift.

This dilemma can be illustrated using an Allan deviation plot. The Allan deviation characterizes the frequency instability of the clocks in the short, mid, and long term. Figure 2 shows...
the deviation for the OS (dashed black line) and the PHY (solid red line). The Allan variance theory assumes that instability arises from noise sources of specific type. The magnitude of each noise source is estimated from the data. The x- and y-axes represent the time over which the frequency is averaged and the magnitude of noise, respectively. The noise terms that contaminate the clock are identified thanks to the curve slope. More specifically, the –1 slope refers to white noise while the positive slope identifies more systematic error such as frequency drift. The longer the averaging period, the more similar and accurate the estimates of frequency over time since white noise is removed by averaging. That is why the instability of both clocks decreases with the averaging period. However, at some point averaging no longer improves the stability of the PHY clock. This is because frequency drift appears in the medium and long run (between 100 and 200 s) and cannot be overcome by simple averaging. On the contrary, the OS clock does not drift because it is synchronized by a Global Positioning System (GPS). The y-intercept also provides precious information about the time accuracy of the PHY and OS logging systems. The OS shows higher noise, which means that the instantaneous frequency of the underlying clock cannot be computed as precisely from the OS timestamps as from the PHY timestamps. The reason is that the delay for timestamping a packet is higher and more variable for the OS than it is for the PHY.

To overcome these limitations, we propose to use the two systems (OS and PHY) jointly in order to leverage both the stability of the OS and the accuracy of the PHY: in each car, we enable packet timestamping at the PHY and OS levels. We synchronize the OS clock with a very accurate GPS. We then perform a time remapping operation that maps the PHY timestamps on the OS time on each node. The remapping process involves a regression between the packet timestamps collected at the PHY and OS levels. Our approach, TimeRemap, offers four advantages:

- It does not need intervehicle signaling.
- It reduces the mean synchronization error for each packet.
- It reduces the number of outlying timestamps.
- It works with off-the-shelf hardware.

The rest of the article is organized as follows. The next section provides an overview of the state of the art. We then present the details of our solution and our testbed. Finally, we compare the performance of TimeRemap with a synchronization algorithm that does not remap timestamps.

**RELATED WORK**

Synchronization in 802.11-based VANETs may rely on the 802.11 ad hoc in-built functions [3]. The problem of this approach is that the beacons which carry the relative time in the cell are generated every 100 ms. This means that in the worst case a node could wait 100 ms before being synchronized. Simply speaking, this implies that when a node connects to a new cell, the first packets it receives are not synchronized. As a result, if one looks traces collected from a 802.11 ad hoc network, one will observe that some parts of the traces are not synchronized with respect to each other. This approach also requires the nodes to be in direct contact.

The 802.11 synchronization scheme does not operate well at the network scope, so specific methods have been proposed to replace it. A lot of work comes from wireless sensor networks (WSNs) because time keeping is essential for monitoring the physical world and intra-network coordination [4]. Most 802.11-based WSNs are static with predefined deployments. Energy is generally the primary concern, so GPS is disabled at the wide scale (GPS applies continuous synchronization, so it needs the nodes to be constantly turned on). The time information is generally transmitted in-band on the channel. Time servers may need several time measurements to compute the clock compensation parameters of clients. This is not a concern in WSNs because packets arrive at the antenna and when its timestamp is actually written by the PHY or OS.
Figure 3. Illustration of the two-step regression procedure.

The links are static. However, such techniques are not well suited to VANETs because the synchronization procedure takes too much time and must be executed every time a node joins a new partition in the network. Besides, if each partition keeps a different local time, the clocks will be reset at each join. The first time remapping scheme has been proposed in the context of WSNs [5]. Sixteen sensors are deployed to monitor a volcano. The system uses three time bases: each node maintains the local and network times. The root node also maintains precise coordinated universal time (UTC) via GPS. When an event is detected, the nodes record the local and network times. Every second, the root node records the network and UTC times. Two successive regressions are applied between the local and network timestamps and then between the network and UTC timestamps to find the relation between the local and UTC time bases. Our approach relies on the same principle (regression) to convert the clocks of the nodes, but it uses the timestamps of the incoming packets as events. The other differences are that we make the GPS time available in each car in order to get rid of the second regression, and we do not need signaling between nodes to maintain the network time.

Specific synchronization methods have also been developed for measuring delays over the Internet with distributed monitoring infrastructure. These platforms generally use specific network interface cards (NICs) that can be interfaced to precise time sources. Reference [6] proposes a passive monitoring platform that uses special NICs known as DAG cards to collect packet traces. The DAG cards on each site are synchronized with a very precise pulse per second (PPS) signal from a GPS. The end-to-end synchronization error of the system is evaluated to 5 μs. Reference [7] studies the influence of NTP on the OS clock. The NIC is a DAG synchronized by GPS/PPS. The time difference between the PHY and OS timestamps is used to quantify the error of the OS clock. This work is closer to ours because it compares the OS and PHY timestamps from the same PC. However, the roles of the time sources are inverted because we synchronize the OS, not the PHY. Furthermore, this configuration is only for testing purposes and does not involve a conversion.

Synchronization has also been studied in vehicular networks, especially for time-division multiple access (TDMA)-based VANETs. The Fleetnet Project [8] investigates the extension of the Universal Mobile Telecommunications System (UMTS) terrestrial radio access (UTRA)-time-division duplex (TDD) standard for VANETs. Two synchronization schemes, GPS and decentralized synchronization, are proposed to guarantee that the timing constraints of the TDMA scheme of Fleetnet are respected. The advantage of the decentralized scheme over the 802.11 ad hoc mode is that it initiates the synchronization procedure as soon as the vehicles enter interference range of each other (i.e., before communication starts). The algorithm guarantees that the synchronization error between every pair of vehicles within a radius of 1800 m is less than 12.5 μs. The drawback of this approach is that it relies on specifics of UTRA-TDD radios and only works with Fleetnet NICs. The GPS scheme directly synchronizes the NICs. Dedicated short-range communications (DSRC) is another candidate technology for communication in vehicular networks. The dedicated synchronization schemes fall into two categories, depending on whether they synchronize the OS or the PHY clock. Reference [9] proposes to synchronize the OS clock through a precise GPS/PPS signal. The method achieves sub-200 μs accuracy. Reference [10] proposes to directly synchronize the DSRC NICs with a GPS/PPS signal. This approach requires the NIC to adjust the phase and/or the frequency of its clock to the PPS signal. Moreover, this solution exclusively works with radios that have a PPS input. By contrast, our solution does not add additional complexity to the NIC and works with different radio types VANETs will support.

A SYNCHRONIZATION SCHEME FOR VANET

The particularity of TimeRemap is to synchronize (directly or indirectly) both OS and PHY clocks:

• The OS clock is synchronized physically by GPS/PPS. This method belongs to the continuous synchronization category because the OS clock is permanently adjusted to UTC [4].

• The PHY clock is not synchronized directly. As a result, the PHY clocks of the vehicles may have different offset and skew, and can possibly drift over time. However, we provide on each vehicle a conversion service3 that can translate a timestamp in the PHY clock base into the UTC base on demand. Such services are commonplace in WSNs [4]. In [11] a daemon runs on each node to convert timestamps from one node clock to another.

TimeRemap can be applied online or offline. It may be used by the radio to convert the occurrence time of an internal event to the UTC base so that this time can be shared and exploited by other NICs. It can also be used by an application

3 The service can be implemented in the kernel or at the application level.
or the OS to convert the PHY timestamps of the received packets into the UTC base. This is the scenario on which we are focusing in this article. The offline version does not need to install the conversion service on each node. The packet traces are simply transferred and processed on a single machine. In both versions the time conversion involves a regression between the packet timestamps collected by the PHY and the OS. The NIC is set in monitor mode to simultaneously log both timestamps.

We synchronize the OS clock to the UTC time with a very accurate GPS/PPS signal. This does not need extra costly hardware because GPS is expected to become a built-in standard for most cars. The GPS receiver just needs to include a PPS output and an internal oscillator stable enough to keep the rate of the PPS signal constant in case the GPS satellite fix is lost [8].

The regression is performed on the received packets only (transmissions are filtered out). This is because the packet path between the OS and the antenna is different upward and downward (so the latency of the system is not the same in both directions). The PHY timestamps correspond, for transmission, to the time when the driver forwards a packet to the NIC and, for reception, to the time when a packet is received by the antenna. The OS timestamps are written by the kernel when a downward (upward) packet enters (leaves) the driver.

We use a two-step regression procedure to remap the PHY timestamps on the GPS time. The first step involves a locally weighted scatter-plot smoothing (LOWESS) regression between the PHY and OS timestamps (denoted \( t_{PHYbase} \) and \( t_{OSbase} \), respectively) collected on a node. LOWESS is similar to simple linear regression, but the procedure is performed over localized subsets of the data instead of the entire logs [12]. The subsets of data used for each fit are determined by a nearest neighbors algorithm. The bandwidth parameter controls how much neighboring timestamp values to use for fitting the regression model for a given timestamp. Our weight function is very simple. It assigns a weight of 1 to all observations. The role of the LOWESS step is to differentiate the good observations from the bad ones and keep only the points with the lower residual for step two. Figure 3 shows how step one is performed at the abscissa \( t_{PHYbase} \). The residuals’ PHYs are computed as the difference between the regression line (dashed line) and the OS timestamps. They are represented by solid vertical lines. The good observations (black dots) have negative residual and lie below the line. They correspond to observations with low OS timestamping delay. The bad observations (white dots) have positive residuals and lie above the regression line. They represent observations with large delay.

The second step provides the estimate, \( t_{OS_{base}} \), of \( t_{PHYbase} \) in the OS base. It performs a linear regression (dashed dotted line) between the observations selected in step 1. The advantage of our two-step technique is to use only the best OS timestamps for the conversion. Our estimate \( t_{OS_{base}} \) are ultimately much more accurate than the original \( t_{OS} \).

The two-step model is run for every PHY timestamp. The mapping function \( f \) is obtained by linear interpolation between the model estimates. The model can be fed with the packets sent by any node. As clock drift appears after a period of 2 to 5 min (Fig. 2), and each local regression only needs 20 observations, our scheme can accommodate very low packet rates.

Every PHY time reference \( t_{PHYbase} \), and obviously the PHY timestamps themselves, can be remapped into the OS timestamps with outlier removal.

\[
f(t_{PHYbase}) = t_{OS_{base}}. \tag{1}
\]

Figure 4 shows the performance of TimeRemap with example traces from our testbed. Each node sends and collects data for 30 min. The timestamps of the frames captured by two nodes are compared. The x-axis represents the frame number and the y-axis the time difference between the timestamps collected by the two nodes. The first PC has low CPU and memory workload while the second one is stressed with the stress Linux command. This configuration is a good representative of a real-world scenario with asymmetric workloads on the cars. As Fig. 4a shows, the error between the OS timestamps varies between –50 and 100 us. Errors are not equally distributed around 0 because of the asymmetry of the workload. The error for the remapped timestamps is plotted in Fig. 4b. One can see how TimeRemap improves synchronization. However, important discrepancies still remain, especially for frames 800 to 4000.
It is worth noting that the empirical distribution of the residuals of TimeRemap has a short tail in the three tests, and that the maximum observed error is obtained in test 2 and equals 12 μs.

Figure 5. GPS/PPS multiplexer. The PPS signal is split into 16.

The arguments of the man example, `–cpu 8 –io 4 –vm 2 –vm-bytes 128M`, are given to the command.

Note that the total jitter of the signal at the PPS driver is less than 1.5 μs (as displayed by the time daemon).

The arguments of the example, `–cpu 8 –io 4 –vm 2 –vm-bytes 128M`, are given to the command.

The remapped timestamps, \( t_{\text{PHY}}^{\text{OS-base}} \), are much more accurate than the original \( t_{\text{PHY}}^{\text{OS}} \) because the timestamping error is lower for the PHY than for the OS. Specifically, we show that \( t_{\text{PHY}}^{\text{OS-base}} \) does not deviate on average more than 3 μs from UTC time.

The PHY does not generate outliers as opposed to the OS. Outliers (at the OS level) arise when the GPS is lost or the interrupts from the NIC are delayed because the system is overloaded. The remapped timestamps are reliable provided that the mapping function \( f \) stays protected from OS outliers. Single outliers are removed during the first step of the algorithm. The recovery procedure is used to protect against multiple outlying observations.

The method does not need intervehicle signaling because only local information is used for regression.

The method works with off-the-shelf hardware.

Testbed Setup

The testbed comprises six Linux desktop PCs (kernel version 2.6.25) with identical configuration. The PCs are synchronized with a PPS signal from a Garmin 18 LVC GPS receiver. The PPS signal is an electric pulse sent once per second to indicate the exact start of each second. The GPS receiver is configured to continue to generate the PPS when the GPS fix is lost. A homemade multiplexer splits the PPS signal from the GPS in 16 (Fig. 5). The circuit board is designed such that the variability of the delay of the PPS signal between the 16 outputs is less than 1 μs. The specifications of our board are available online at [13]. Each PC is connected to the PPS multiplexer through its serial port.

The shmpps driver is installed on all PCs. It simply senses the PPS on the serial port, determines the offset (in microseconds) between the OS clock and the PPS, and passes the value to the time daemon. The role of the time daemon is to correct the wall clock time expressed in seconds with the PPS offset. In most configurations, the wall clock time does not come from the GPS (because it needs to be sent on a separate serial or parallel port) but from the Internet. Several time daemons can perform this task (ntpd, chrony, etc.). We opt for chrony because it supports intermittent Internet connection. The Network Time Protocol (NTP) daemon needs continuous connection, which is not feasible in VANETs. Furthermore, chrony can stabilize the OS clock to the UTC time in less than 10 min.

In comparison, NTP needs several hours from startup to converge to UTC.

The PCs are equipped with 802.11g wireless interfaces and the Madwifi driver. We enable the ahdemo and monitor modes simultaneously on each card to allow the nodes to transmit and collect data at the same time. The packet traces are recorded in the PCAP format. Each node broadcasts a packet every 2 s. The PCs are located within communication range so that the reception time of the packets at the six PCs can be compared.

Performance Evaluation

We compare TimeRemap with a scheme that operates on the OS timestamps and synchronizes the OS clock via GPS/PPS. The algorithms are tested under three scenarios. In the first one the testbed is not stressed. In the third test all the PCs are stressed with the stress Linux command. In both tests we compare the timestamps of the received frames for each possible pair of nodes. In the second test the nodes are divided into two halves, one stressed and the other not. The first node of each pair is picked in the first set, and the other is the second half. The results are averaged over all possible combinations of nodes. Each experience lasts an hour and is repeated five times.

The size of the neighborhood of the LOWESS regression of TimeRemap is set to 21. No loss of GPS signal is observed, so the performance of TimeRemap is the same with and without recovery mode.
For each pair of nodes we compute the offset between the timestamps of the packets common to the two nodes. The performance metrics are the mean error, standard deviation, and outlier ratio. An outlier is considered as an outlier if the offset between the timestamps exceeds 30 μs. These metrics turn out to be good performance indicators because they are simple, and assess the strengths and weaknesses of the two synchronization schemes.

The average results for the five runs are represented in Fig. 6. As we can see, TimeRemap reduces the mean error by 20 percent in test 1 (no stress), by 50 percent in test 2 (half stressed), and 75 percent in test 3 (all stressed). The mean error and standard deviation are below 3 μs in all three scenarios. The performance is slightly lower in test 2 because of the asymmetry of the workload between the nodes. This asymmetry creates a small offset between the mapping function of the nodes, which impacts the results. TimeRemap does not produce any outlier as opposed to the classical scheme that generates 1.5 to 8 outliers per 10,000 observations. It is worth noting that the empirical distribution of the residuals of TimeRemap has a short tail in all three scenarios. The performance is slightly lower in test 2 because of the asymmetry of the PHY and OS timestamps. TimeRemap is especially well suited to dynamic scenarios like VANETs and OS timestamps are substituted on each node by using PHY timestamps instead of OS ones. The OS and PHY packet capture systems. The OS synchronization schemes.

CONCLUSION

This article has presented TimeRemap, a technique that provides stable and accurate packet timestamps by combining the best features of the OS and PHY packet capture systems. The OS clock is stabilized by GPS. Accuracy is obtained by using PHY timestamps instead of OS ones. OS timestamps are substituted on each node by performing a local regression between the PHY and OS timestamps. TimeRemap is especially well suited to dynamic scenarios like VANETs as no signaling is needed between nodes. The trace analysis shows that TimeRemap reduces the mean synchronization error to 3 μs and removes all outliers. In future work we plan to deploy TimeRemap on the road and implement monitoring applications.

REFERENCES


BIographies

RYAD BEN-EL-KEZADRI is a research scientist at the University of California at Los Angeles (UCLA) Computer Science Department. He received his Master’s and Ph.D. degrees in computer science from Université Pierre et Marie Curie, Paris, France, in 2001 and 2007. His research interests include wireless networks, network simulation, and system design.

GIOVANNI PAVU is a research scientist at the UCLA Computer Science Department. He received a Laurea doctorate in computer science and a Ph.D. in computer engineering from the University of Bologna, Italy, in 1998 and 2003, respectively. His research interests include peer-to-peer networks, wireless multimedia, distributed systems, and ad hoc networks.