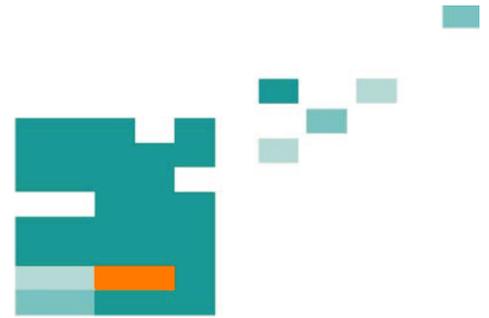


## 55. IWK

Internationales Wissenschaftliches Kolloquium  
International Scientific Colloquium



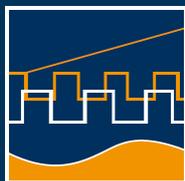
13 - 17 September 2010

# Crossing Borders within the **ABC**

**A**utomation,

**B**iomedical Engineering and

**C**omputer Science



Faculty of  
Computer Science and Automation

[www.tu-ilmenau.de](http://www.tu-ilmenau.de)

*th*  
TECHNISCHE UNIVERSITÄT  
ILMENAU

Home / Index:

<http://www.db-thueringen.de/servlets/DocumentServlet?id=16739>

## **Impressum Published by**

Publisher: Rector of the Ilmenau University of Technology  
Univ.-Prof. Dr. rer. nat. habil. Dr. h. c. Prof. h. c. Peter Scharff

Editor: Marketing Department (Phone: +49 3677 69-2520)  
Andrea Schneider (conferences@tu-ilmenau.de)

Faculty of Computer Science and Automation  
(Phone: +49 3677 69-2860)  
Univ.-Prof. Dr.-Ing. habil. Jens Haueisen

Editorial Deadline: 20. August 2010

Implementation: Ilmenau University of Technology  
Felix Böckelmann  
Philipp Schmidt

## **USB-Flash-Version.**

Publishing House: Verlag ISLE, Betriebsstätte des ISLE e.V.  
Werner-von-Siemens-Str. 16  
98693 Ilmenau

Production: CDA Datenträger Albrechts GmbH, 98529 Suhl/Albrechts

Order trough: Marketing Department (+49 3677 69-2520)  
Andrea Schneider (conferences@tu-ilmenau.de)

ISBN: 978-3-938843-53-6 (USB-Flash Version)

## **Online-Version:**

Publisher: Universitätsbibliothek Ilmenau  
[ilmedia](#)  
Postfach 10 05 65  
98684 Ilmenau

© Ilmenau University of Technology (Thür.) 2010

The content of the USB-Flash and online-documents are copyright protected by law.  
Der Inhalt des USB-Flash und die Online-Dokumente sind urheberrechtlich geschützt.

## **Home / Index:**

<http://www.db-thueringen.de/servlets/DocumentServlet?id=16739>

# DOS-RESISTANT DISTRIBUTED TIME SYNCHRONIZATION FOR VIRTUAL PRIVATE NETWORKS

*Rene Golembewski and Michael Rossberg and Guenter Schaefer*

Ilmenau University of Technology, Germany  
Telematics and Computer Networks Research Group

## ABSTRACT

In order to securely exchange data over the Internet, more and more organizations utilize virtual private networks (VPNs) which form potentially large overlay networks. Recently, approaches for VPN auto-configuration have been presented, and while VPNs usually do not address availability, novel systems based on peer-to-peer communication can do so. Nevertheless, there are no satisfying solutions for time synchronization within VPNs that are designed for availability against denial-of-service (DoS) attacks, node failure, and network partitioning. The Network Time Protocol (NTP) is widely used, but relies on hierarchical structures, and thus is not suitable for scenarios with high availability requirements.

Thus, in this article we present a novel, fully distributed, and fault tolerant time synchronization approach, that is designed to be transparently integrated in VPN gateways. Combining diffusion-based round-trip-synchronization with an external and internal attacker detection, the proposed mechanism is making a contribution to resilient VPN design.

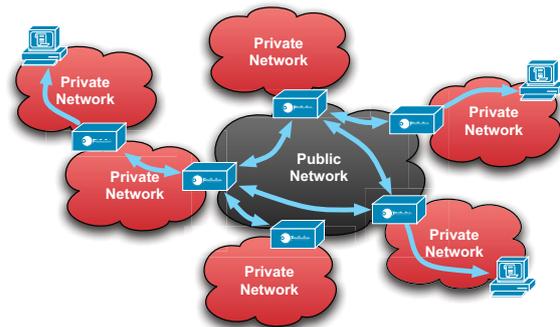
**Index Terms**— time synchronization, attacker detection, availability, security, VPN

## 1. INTRODUCTION

In order to exchange private information over untrusted networks, such as the Internet, worldwide operating organizations depend on VPNs to connect branch offices. Fig. 1 shows a typical scenario with six private networks, building a nested VPN. Inside of VPNs some important security goals are realized inherently by the deployed VPN protocol, e.g., IPsec. Communicating nodes are authenticated against each other and the transfer of data is confidential and integrity protected.

To successfully achieve these security goals within VPNs, just like for many other processes in modern communication infrastructures, time synchronicity is an important basis. The validation of certificates and distributed logging of network events are two examples, where synchronized clocks are required.

A disadvantage especially of large VPNs is the re-



**Fig. 1.** VPN topology with nested subnets and distributed logging hosts

quired configuration and maintenance effort, resulting in a limited scalability and difficulties in supporting mobile nodes. Hence, several auto-configuration approaches have been developed over the last decade, and notably the more recent ones are organized in a fully distributed way to avoid scalability issues. Another advantage is the absence of single points of failure, thus VPNs can now be optimized with respect to the previously unaddressed availability.

A fully automated approach for a distributed configuration mechanism is described in [1]. This approach proposes that all VPN gateways are communicating via a ring-based peer-to-peer overlay structure with proactive established shortcut connections to scale well with the number of nodes.

Yet, from the perspective of availability, it does make little sense to design only the VPN management algorithms in a distributed fashion, while other important higher layer functions like time synchronization, name resolution, or logging are still realized by central servers within the VPN, e.g., the current de facto standard for time synchronization – NTP [2] – depends on a hierarchical server infrastructure. The goal is to achieve DoS resilient behavior for all provided services, and therefore also a distributed algorithm for time synchronization. Moreover, it is possible that nodes get compromised by an attacker and can be used afterwards to disturb the synchronization process.

This article analyzes the current state of the art and

presents a novel distributed approach for secure time synchronization with DoS resistance properties. Furthermore, we contribute an evaluation of convergence and attacker resistance via a simulation study.

The rest of the article is organized as follows: section 2 describes the objectives for a distributed time synchronization in VPNs. Related work is discussed in section 3, and in section 4 we describe the main system design and functionality. Section 5 evaluates the mechanism with respect to the objectives by analysis and simulation study. Finally, section 6 concludes and gives an overview of planned further research.

## 2. OBJECTIVES

In order to ensure high availability in VPN scenarios, time synchronization algorithms should meet the following requirements:

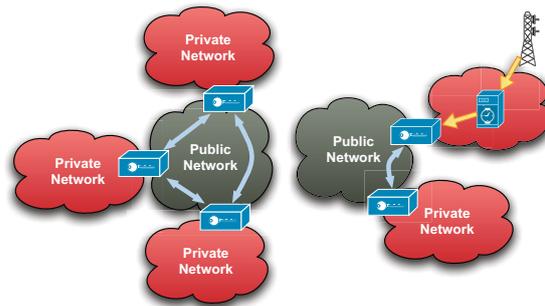
**1. Operation in global VPN environments:** As time needs to be adjusted within Internet Protocol (IP) based VPNs, there is no possibility to perform any-, broad-, or multicast. Furthermore, there is no access to link layer information and nested security associations must be regarded.

**2. Internal clock synchronization:** As long as all VPN gateways are synchronized internally, the whole VPN is in a consistent state. The local clocks should be synchronized to the second, which is believed to be sufficiently accurate to provide security services, e.g., to verify the lifetime of a certificate correctly and analyze log files from different nodes based on their local timestamps.

**3. Accordance of internal clocks and external time:** One consequence of a strictly internal synchronization is the inevitable divergence of internal and external time values. Normally, all core functions work properly without having a strict external synchronicity, but the algorithm must support means to explicitly synchronize to external clock sources, whose timestamps shall be securely injected at arbitrary VPN nodes.

**4. Integrity:** Even though, external attackers are not able to determine the meaning, or undetectably change any transmitted data, because of the encrypted VPN tunnels, but it might be possible to drop or delay packets to gain influence on time synchronization. Especially the latter, which will henceforth be referred to as *delay attack*, is difficult to detect and must not be able to successfully manipulate the synchronization mechanism.

Internal attackers are potentially able to take part in the synchronization process and try to manipulate other nodes. The synchronization algorithm shall be robust to such attacks as long as not more than half of its local neighborhood is compromised.



**Fig. 2.** Partitioned network scenario with lost external synchronicity in left partition

**5. Availability:** Even if one or more nodes are unable to communicate due to a DoS attack, the synchronization of the remaining ones should not be affected. Furthermore, it shall not be possible to distinguish nodes with an additional external synchronization from other participants, in order to prevent directed attacks.

**6. Robustness:** Closely related to availability is the resistance against random failures or harsh network conditions. If no communication between two or more groups of gateways is possible, e.g., because of broken links in the transport network (see Fig. 2), the whole VPN can be partitioned. The most important requirement is the permanent inner synchronicity of the subnets after partitioning. Furthermore, if two former partitioned subnets get united again with a certain time offset, they should not see each other as attackers but should instead perform an unexceptional synchronization.

Additionally, the algorithm should also be robust against comparably high packet loss in order to deploy it in scenarios with mobile nodes or bad connectivity.

**7. Efficiency:** Time synchronization needs additional information exchanges between the neighboring nodes, and the induced overhead shall be as small as possible. Moreover, the whole processing of the implemented features should not generate much load on VPN nodes, because the processing power may be limited, e.g., on embedded or mobile devices.

**8. Scalability:** In order to be deployed in large VPNs and transport networks like the Internet, synchronicity must be achieved by communicating only with a constant or logarithmic number of neighboring nodes. Furthermore, the use of broad- and multicast is not possible.

## 3. RELATED WORK

The following section gives an overview concerning the state of the art in synchronizing clocks over communication networks. The most important approaches

are described in more detail and rated according to the previously introduced objectives.

### 3.1. Approaches for wired networks

NTP and the Simple Network Time Protocol (SNTP) [2, 3] are perhaps the most popular protocols for time synchronization in IP networks. Being a hierarchical client-server approach, synchronization information is distributed from an external clock source at the top over intermediate servers to clients at the lowest level. Although actually forming a forest, DoS attacks and compromised nodes may cause serious problems. Thus, NTP is inappropriate in VPNs with high availability requirements.

Further approaches for fixed networks are the well known Berkeley synchronization [4] and the similar algorithm of Cristian [5]. In both mechanisms time servers send out the time values to the clients, which adjust their local clock without any further checks. These centralized servers are an exposed target for attacks against the availability, and it is possible to control all individual times by compromising the master.

### 3.2. Time synchronization in wireless sensor networks (WSNs)

The emergence of WSNs goes along with fundamental changes in nearly every distributed algorithm [6], and when comparing VPNs and WSNs similarities in the failure model emerge. Both require fully distributed approaches to manage random failures and internal as well as external attackers.

[7] and [8] summarize the most important algorithms for time synchronization in WSNs. Current research focusses on refining the approaches to realize attack resistance and to cope with node breakdowns [9, 10, 11, 12], as well as compensating local clock-drifts [13].

The article [14] proposes a method called *Asynchronous Diffusion* (AD) for use in WSNs, which is most applicable to the stated objectives. This algorithm depends on the asynchronous exchange of timestamps with all neighbors, which are collected and averaged. The obtained mean time is then periodically sent out to the neighbors, which in turn adjust their local clocks afterwards.

As long as all participants exchange synchronization information with nonzero probability, the algorithm was proven to converge. The use of broadcasts and link layer access makes the algorithm not directly deployable in VPNs, but the fundamental approach, i.e., the asynchronous averaging of timestamps, is a good starting point.

## 4. SYSTEM DESIGN

This section introduces a hybrid approach for keeping a common internal time on all gateways of a VPN, as well as the synchronicity between VPN and environment. In a first step, the internal system clock of a VPN node will be adapted concerning the drift, before the actual distributed algorithm performs a synchronization within the VPN. Both steps are covered in detail in the following.

### 4.1. Autonomous Drift Adaption

The adjustment of the drift of the local software clock is most important during the first participation in the synchronization process, and can be repeated later on if necessary. This can be realized utilizing one of two sources: Either the joining node trusts the existing network and starts compensating the drift of its software clock against the other nodes, or against the local hardware clock, which is mostly triggered by a Complementary Metal Oxide Semiconductor (CMOS) oscillator. Concerning security and reliability, a network with many already synchronized nodes is more reliable than a hardware clock, which may be rather heavily influenced by the local environment of the node, but in order to retain heavily drifting software clocks it is still profitable to synchronize against the local CMOS clock first.



Fig. 3. Drift Adaption Measurement

Drift adaptation is technically realized by a *training time* (see Fig. 3), whereas the precision of this process increases with longer measurement periods. Afterwards, a frequency correction can be computed and applied to the local software clock.

### 4.2. Measurement of Round-Trip-Times (RTTs)

After the initial local drift correction, the periodic neighborhood synchronization starts. The main principle for exchanging time information is the asynchronous measurement of RTTs (see Fig. 4). All communicating nodes are periodically asking their neighbors for their local time. The queries are immediately answered including a nonce  $N$  from the request, which ensures the freshness of the answer and helps to avoid delay attacks of arbitrary length as nonces will be marked invalid after a short time. The assumption of a symmetric RTT is made, because a neighbor is never seen as trustful, concerning its reported local time. Thus, it is also not possible taking

into account a processing time at the answering node, which could also be forged. In fact, only the delay of the answering packet is important, because it is used to estimate the actual time of the respective neighbor. Since the algorithm is executed by all nodes of a VPN, time changes.

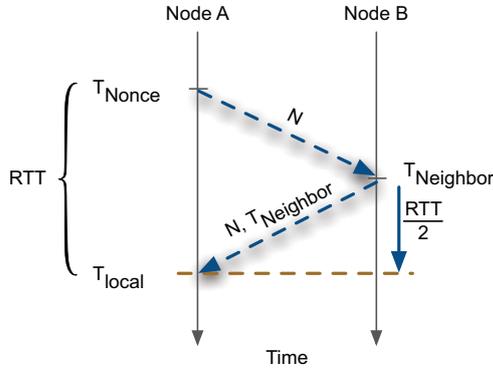


Fig. 4. RTT Measuring

To protect nodes against each other, synchronization is only allowed between direct neighbors or via shortcut connections, which are realized by the ring-based VPN overlay (see [1] for details). This is needed to avoid the excessive creation of new connections and gaining more influence on a specific node, that could enable an internal attack.

### 4.3. Averaging Algorithm

As described in section 3, a network can be synchronized by averaging all collected timestamps in each node and afterwards adjust all nodes' clocks. Our algorithm differs from the mentioned implementations, because the calculated mean time is not sent out to the neighbors to adjust their times. The computed average neighborhood time is only used to set the own local clock a step towards this value. A node has therefore not to trust any neighbor about the new time value it should accept. Given a number of  $N$  neighbors with different time offsets  $O_n$ , the correction value  $C$  can be calculated as

$$C = \frac{1}{N} \sum_{n=1}^N O_n. \quad (1)$$

Additionally, the information of some special gateway neighbors can be weighted stronger in order to speed up the synchronization, which is called *topology adaptation*. This situation evolves mainly in nested scenarios, where for example a certain amount of nodes is building an enclave with only one gateway to reach other nodes indirectly (see Fig. 5). In this case, the network would take a much longer time to converge globally without a special topology adaptation. Other distributed synchronization algorithms do not cover this

special case, therefore it is explained in more detail in the following.

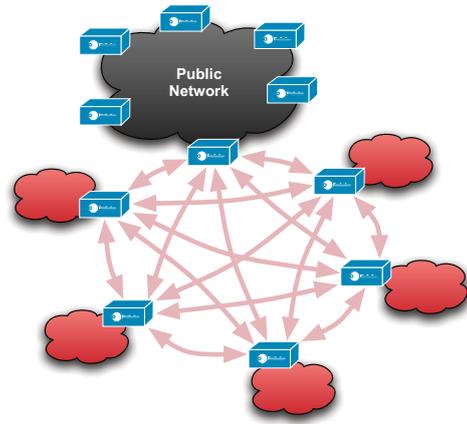


Fig. 5. Five enclaved nodes behind a single VPN gateway in a nested scenario

A gateway with many nested nodes behind it can mostly contribute better synchronization information from inside of the core network. But because of the averaging algorithm, it would not have a big influence on a strongly connected enclave. The goal is on the one hand to prefer a gateway's information, but on the other hand to empower it not too much, in order to avoid that it becomes a potential weak point for attackers. Therefore, all neighbored gateways that are serving the enclave must not exceed a certain weight  $w$ , which is calculated in every node based on the number of all neighbors  $N$  and the amount of gateways  $G$ :

$$w = \left\lfloor \frac{N - G}{G} \right\rfloor. \quad (2)$$

The resulting correction value  $C$  can now be given as

$$C = \frac{1}{N - G + wG} \left( \sum_{n=1}^{N-G} O_n + \sum_{g=1}^G w \cdot O_g \right). \quad (3)$$

All gateways can now have a little more influence, limited by the number of other neighbors.

Furthermore, there can also be inaccurate values caused by different malicious nodes or bad connection parameters, that should not be used to calculate the new local time. The classification of nodes in capable or incapable for synchronization is implemented in the following algorithm for attacker detection. The basic principle is to introduce a stability flag to classify each node, and use only the stable ones for synchronization.

### 4.4. Attacker Detection

Attacker detection is equivalent to the ability to identify nodes with corrupt synchronization information. There

can be various reasons for incorrect information: Processing times, queuing of packets, jitter, asymmetric RTTs or deliberate delaying of packets. Of course, an internal attacker could also forge the timestamps before sending them to the other nodes.

The only possibility to separate “good” and “bad” nodes is the examination of received timestamps. To have a more detailed view on the behavior of synchronization neighbors, a certain amount of previously received information is kept for statistical analyses and can be accessed by the detection algorithms.

The detection of malicious behavior is divided in different phases. To be marked as *stable*, a neighbor’s offset must not exceed a hard limit, which is determined in the configuration file, or automatically calculated inside each node, based on the average offsets of all stable neighbors. This value shall prevent the network from being influenced by new joining nodes with a strong time difference, and it is also a protection against internal attacks with huge offsets.

Afterwards, there is a threshold detection and adaptation mechanism, which automatically adapts to the received timestamps of each single synchronization neighbor, in order to get a first outlier detection by excluding nodes from the averaging algorithm, if they are exceeding the threshold temporarily (see Alg. 1). The value  $T_{max}$  is inherently given by the maximum allowed offset for all neighbors, and the minimum threshold  $T_{min}$  should be in the dimension of the timestamp’s standard deviation, or can be set to zero.

---

**Algorithm 1:** Threshold adaptation

---

```

1 if  $abs(offset) < \frac{threshold}{2}$  then
2   | threshold  $\leftarrow \max(threshold \times 0.9, T_{min})$ ;
3 if  $abs(offset) > threshold$  then
4   | excludeNodeTemporarily  $\leftarrow true$ ;
5   | threshold  $\leftarrow \min(threshold \times 1.1, T_{max})$ ;
6 else
7   | excludeNodeTemporarily  $\leftarrow false$ ;
```

---

Like mentioned before, we are using a stability criterion for each connected synchronization neighbor, and if more than half of a node’s neighbors are classified as *stable*, it only uses their timestamps for the averaging algorithm. In the beginning, a joining node will have a fully unstable neighborhood and accepts all synchronization messages to adapt to the network as quickly as possible. Also the next steps are integrated in the detection process and enable the possibility to set the stability flag of a neighbor.

Subsequent to the offset thresholds, an outlier correction and drift analysis is done by using linear regression. This is enabled by saving the last received timestamps of all neighbors and the respective local timestamps of their arrival in different queues, that build the underlying dataset for the following operations. Nor-

mally, the compared clock vectors should produce a regression function with a slope of 1.00, if the frequencies of both clocks are about the same (see Alg. 2). Therefore, we can make a comparison between local and neighbored clockspeeds, and either mistrust drifting nodes, or make the assumption that the own local clock needs to be tuned, if all neighbors seem to be drifting in the same direction.

---

**Algorithm 2:** Detection of clockdrift to a single neighbor with simple linear regression analysis

---

```

1 sumx, sumy, SSxx, SSxy  $\leftarrow 0$ ;
2 size  $\leftarrow neighborQueue.size()$ ;
3 for  $i \leftarrow 0$  to size do
4   | sumx += referenceQueue[i];
5   | sumy += neighborQueue[i];
6  $\bar{y} \leftarrow sumy / size$ ;
7  $\bar{x} \leftarrow sumx / size$ ;
8 for  $i \leftarrow 0$  to size do
9   | SSxy += (referenceQueue[i] -  $\bar{x}$ )  $\times$ 
10  | (neighborQueue[i] -  $\bar{y}$ );
11  | SSxx += (referenceQueue[i] -  $\bar{x}$ )2;
12 slope  $\leftarrow SSxy / SSxx$ ;
```

---

The third level of stability detection is also based on a statistical evaluation of the received offsets. Mobile devices are most likely connected via GSM, UMTS, WLAN, or Bluetooth. Large and asynchronous RTTs, more jitter and other special characteristics of the underlying transport networks can have a negative influence on the synchronization algorithm. On the other hand, a low standard deviation indicates a stable connection and an excellent counterpart for time synchronization.

After these three steps are processed, a stability decision can be made in every node, to evaluate and classify the neighborhood. Participants with bad connection parameters, who are always scored as *unstable* by others, thus have the ability to synchronize their own local clocks against the network, even if no other node is taking their information into account.

## 5. EVALUATION

In this section, we present the results of our simulations and evaluate the introduced mechanisms qualitatively. The synchronization algorithm has also been tested on a Linux-based scenario with 17 different x86 machines, whereas the results reflected the outcomes of the simulation study. The chosen network topology was generated by using the Skitter dataset, published by the Cooperative Association for Internet Data Analysis (CAIDA) [15]. Based on this Internet wide (BGP-)router topology, a reduced scenario with 217 nodes was computed to have a statistically similar ana-

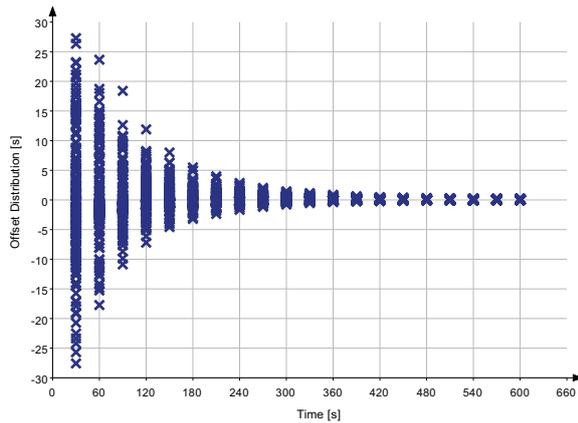


Fig. 6. Convergence with 204 nodes

lytic model of the Internet backbone. The used simulation framework was OMNeT++ [16] with INET extension and additionally implemented IPsec support.

### 5.1. Operation in global VPN environments

Since the algorithm does not make a use of any cross-layer communication, it is deployable in any kind of IP network. Features like any-, broad-, or multicast are also not needed to exchange synchronization messages. As long as the network is connected, a global mean time will be reached inside.

### 5.2. Internal clock synchronization

To measure the synchronization speed, we initially added an offset to all clocks. Different statistical distributions of that failure had no influence on the synchronization speed. Thus, we have chosen a uniform distribution, which allows us to make a statement about the stability of the global mean after synchronizing. Our metric was the elapsed time until the difference between lowest and highest clock reading fell below a defined value, which we set to 500ms for the network to be synchronized.

Figure 6 shows an exemplary process for about 10 minutes after the synchronization algorithm was started. This scatterplot makes it easy to see the exponential convergence of the local clock readings against the global mean.

We also analyzed, if there are any influences on the synchronization behavior with special topological characteristics like nested scenarios or clustering of gateways behind one another. With our computed simulation topology we were able to cover all of these special cases and can state a high flexibility concerning the underlying topology.

### 5.3. Accordance of internal clocks and external time

Reaching external synchronicity can be realized by using one or more trustful external timing sources. To enable this behavior, the time adjustment must be turned off for the externally synchronized node manually, and it is necessary, that the network is synchronized to the respective node first, because otherwise it could be seen as an attacker. An administrator can therefore set the time of all nodes at once, to reach this synchronicity initially. After that procedure, an externally synchronized node can prevent the whole network from a hardly detectable long-term drift. Of course, this solution can only work with a compensable clockdrift in all other nodes, but that is realized by the introduced drift adaptation method.

### 5.4. Integrity

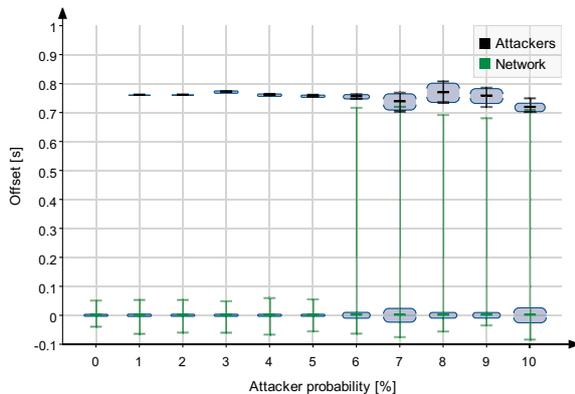
As mentioned before, an external attacker is only able to delay packets in the transport network, without knowledge if this might be synchronization information. That may result in an asymmetric RTT and can influence the algorithm to a minor degree. The solution is to exclude timestamps with a very high RTT value in general. For example, if we allow a maximum RTT of one second, and the attacker is able to delay all incoming packets of a certain node (ISP position), he can reach a maximum time failure of one second (minus the real one-way delay). Thus, an external attacker can only generate a limited bias at a certain node and is not able to harm the synchronicity. Even if a node is connected via WLAN or GPRS, the RTT is always below 2 seconds in the worst case [17].

Internal attackers have the same rights and possibilities as normal nodes, so they can actively distribute forged timestamps. Nevertheless, they are not able to eavesdrop any synchronization messages of other nodes. The main advantage for internal attackers is the ability to analyze a neighbor's reaction on forged timestamps. That is the reason why we also consider an adaptive attacker model called *frog boiling* [18]. The name for this attack bases on the rumor, that a frog would not recognize its own threat if he is put into cold water, that is slowly heated until it boils. Of course this is only a story, but it describes the attacker model very well. All other attacks (e.g., false timestamps with big offsets) are easy to recognize for the introduced attacker detection. At least about half of a nodes neighbors must be compromised to enable those trivial attacks. A frog boiling attack is very hard to detect, because the attacker can directly examine his own profit, and thus adapt the applied strategy. This can only be successful, if it is not possible for the victim to distinguish good and bad neighbors.

Obviously, a frog boiling attack needs a lot of time to be performed, and it is questionable how long it takes

to notice that a node has been compromised. Even though, it is possible to enforce a small drift to the whole network, if no external time sources are available. The forged timestamps would be diffused through the entire network, and a clock difference between internal and external time would occur. This is not possible, if the network is connected to one or more “hidden” clock sources. In this case, a frog boiling attacker is trying to pull a chosen victim away from the common mean time.

We simulated this behavior in the former described scenario with 217 nodes, whereas 204 are running the synchronization code, and the remaining 13 nodes are chosen as backbone routers (see Fig. 7). The attackers are positioned in the network randomly with a certain pre-defined probability (x-axis). Therefore, a node determines a random number and then makes a decision if it becomes an attacker or not. The resulting distribution of local times after an attack of 2 hours with variable attacker probability is depicted in the given Figure. By slowly increasing the transmitted timestamps, the attackers are about 750ms ahead, compared to all uncompromised nodes. With more than 5% attacker probability, some nodes start trusting their attackers and drift away from the average network time by adopting the forged time.



**Fig. 7.** Frog boiling attack with different attacker probabilities

Unfortunately, a distributed synchronization algorithm cannot be fully protected against the frog boiling attack, as long as small offsets are allowed, because the attacker can always adapt his behavior to the imposed restrictions. This raises the question, what we call a successful attack against the synchronization algorithm. Knowing that frog boiling is not recognizable in a system of potentially wrong clocks, we can confirm that the more attackers we have inside a network, the higher the influence on the synchronization algorithm. In fact, an amount of less than 5% had no measureable influence on our algorithm, because of the averaging of all timestamps. Furthermore, an attacker must act as a trustworthy node all the time and is not able to dictate

other node’s times, as long as he does not control more than 50% of their neighbors. In our simulation runs, we positioned the attackers in the network randomly. Hence, we were not able to exclude this special case of an overwhelmingly malicious neighborhood in some cases. This is the reason for some positive frog boiling attacks starting from 5% attacker probability, which are completely traceable back to the stability criteria, we introduced before. In a strongly connected network, with a high number of neighboring nodes, attacks would be much more complicated than in sparsely connected scenarios.

### 5.5. Availability

Due to the fact, that our algorithm is fully distributed and localized, there is no vulnerable single point of failure whose compromise can have a big influence on the synchronization. A potential attacker would have to prevent the whole communication of nodes for a long time, but then no VPN communication would be possible at all.

### 5.6. Robustness

In case of a network partitioning, the divided subnets proceed with the synchronization just like before, and the synchronicity is always ensured. If the separation continues for a long time, the subnets might drift away from each other because of small differences in the resulting clockdrifts. Due to the stability metric, a later reunion may lead to a situation with temporarily instable nodes until the whole network is fully synchronized again. Mostly, this will be the nodes of the smaller subnet, but that depends mainly on special topology properties and neighborhood relationships.

### 5.7. Efficiency

Synchronization messages are always piggybacked at communication processes of the VPN overlay. To realize the freshness of timestamps, a nonce is sent in the request message and has to be appended to the reply message, as well as a timestamp of the local clock. The timestamp is an 8 byte float value, and the nonces are 4 byte integer values.

The computing overhead for time synchronization depends linearly on the number of synchronization neighbors, and the steps for attacker detection are constant due to the fixed size of all data queues.

### 5.8. Scalability

To ensure a fast and precise synchronization, a node must not have a huge amount of neighbors. It is more important to reach the whole network with shortcut connections and a maximum number of  $O(\log n)$

hops. This is realized by establishing those connections proactively and thus limit the number of needed neighbors (see [1] for details). Because of that, the algorithm scales well with a growing number of nodes.

## 6. CONCLUSION AND FUTURE WORK

Within this article, a novel approach for distributed time synchronization in IP networks and large VPNs was presented. In combination with attacker detection and stability metrics, a topology adaptation was implemented to be used also in nested and complex scenarios. Compared to existing distributed synchronization approaches, e.g., the very similar *asynchronous diffusion*, this algorithm exclusively uses unicast IP communication for the exchange of messages. Furthermore, nodes do not trust each other, and do not accept any information without additional checks. The only reliable source of information is the measured RTT, whereas the neighbored time information is always assumed to be potentially corrupt.

Additionally, the proposed algorithm does not rely on any initiator and has no central instances, which could be a primary target for attackers. The use of an asynchronous and distributed mechanism eliminates this weak point. A temporary network partitioning is tolerable and occurs without a loss of inner synchronicity in the several subnets.

To analyze the robustness against external and internal attackers, a simulation study was conducted and discussed. External attackers are not able to harm the synchronization process by delaying packets because of the maximum RTT values. An adaptive internal attacker is able to manipulate timestamps and can have a bigger influence on the algorithm. Nevertheless, he is not able to really harm the network with only a small amount (< 5%) of compromised nodes. If an adaptive attack is implemented in several nodes, the attacker can stay undetectable and try to influence the network with seemingly good values, but several detection mechanisms and thresholds are reducing the achievable damage.

However, some issues remain to be improved in further studies. A problem that has not been addressed in the required depth is the fully automatic tuning of all synchronization parameters. Options like maximum RTT, standard deviation, or stability ratio are set via the configuration files or with a default value. But with fully automatic adaptations there should not arise new attack possibilities with compromised nodes. In all cases it should be considered how an adaptive attack can manipulate the implemented procedures.

## 7. REFERENCES

- [1] Michael Rossberg, Guenter Schaefer, and Thorsten Strufe, "Distributed Automatic Configuration of Complex IPsec-Infrastructures," *Journal of Network and Systems Management*, May 2010.
- [2] David Mills, "RFC1305: Network Time Protocol (Version 3) specification, implementation and analysis," 1992.
- [3] David Mills, "RFC4330: Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI," 2006.
- [4] Riccardo Gusella and Stefano Zatti, "The Accuracy of the Clock Synchronization Achieved by TEMPO in Berkeley UNIX 4.3BSD," *IEEE Transactions on Software Engineering*, vol. 15, no. 7, 1989.
- [5] Flaviu Cristian, "Probabilistic clock synchronization," *Distributed Computing*, vol. 3, no. 3, pp. 146–158, Sept. 1989.
- [6] Kay Römer, "Drahtlose Sensornetze," *VISIO-NEN*, pp. 34–38, June 2006.
- [7] Kay Römer, Philipp Blum, and Lennart Meier, "Time Synchronization and Calibration in Wireless Sensor Networks," *Handbook of Sensor Networks: Algorithms and Architectures*, pp. 199–237, 2005.
- [8] Fikret Sivrikaya and Bülent Yener, "Time synchronization in sensor networks: a survey," *IEEE Network*, vol. 18, no. 4, pp. 45–50, July 2004.
- [9] Saurabh Ganeriwal, Srdjan Čapkun, Chih-Chieh Han, and Mani B. Srivastava, "Secure time synchronization service for sensor networks," in *Proceedings of the 4th ACM workshop on Wireless security*. 2005, p. 106, ACM.
- [10] Kun Sun, Peng Ning, and Cliff Wang, "Secure and resilient clock synchronization in wireless sensor networks," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 2, pp. 395–408, Feb. 2006.
- [11] Hui Song, Sencun Zhu, and Guohong Cao, "Attack-resilient time synchronization for wireless sensor networks," *Ad Hoc Networks*, vol. 5, no. 1, pp. 112–125, Jan. 2007.
- [12] Yafei Yang and Yan Sun, "Securing Time-synchronization Protocols in Sensor Networks: Attack Detection and Self-healing," *Analysis*, 2008.

- [13] Tao Bian, Ramachandran Venkatesan, and Cheng Li, “Adaptive Time Synchronization for Wireless Sensor Networks with Self-Calibration,” *2009 IEEE International Conference on Communications*, pp. 1–5, June 2009.
- [14] Qun Li and Daniela Rus, “Global clock synchronization in sensor networks,” *IEEE Transactions on Computers*, vol. 55, no. 2, pp. 214–226, Feb. 2006.
- [15] CAIDA, “The Cooperative Association for Internet Data Analysis,” 2009.
- [16] András Varga, “The OMNeT++ Discrete Event Simulation System,” in *Proceedings of the European Simulation Multiconference (ESM’2001)*, 2001, pp. 319–324.
- [17] Peter Benko, Gabor Malicsko, and Andras Veres, “A large-scale, passive analysis of end-to-end TCP performance over GPRS,” in *IEEE INFOCOM*. 2004, pp. 1882–1892, IEEE.
- [18] Eric Chan-tin, Daniel Feldman, Nicholas Hopper, and Yongdae Kim, *The Frog-Boiling Attack: Limitations of Anomaly Detection for Secure Network Coordinate Systems*, vol. 19 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, chapter 1, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.