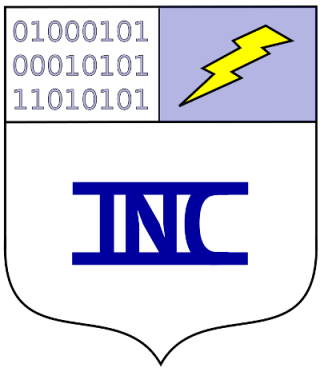
Properties of Network Time

B. Kirkpatrick *

July 20, 2020

Intrepid Net Computing



www.intrepidnetcomputing.com

*bbkirk@intrepidnetcomputing.com

Document Revision History

April 17, 2020 Drafted notes on network time.

July 13, 2020 Drafted manuscript.

July 20, 2020 Published on www.intrepidnetcomputing.com.

Abstract

Network time is the phenomenon of setting the time on various computers in a connected network. Applications of network time range from collecting time-series data from disparate sensors to use as an indicator of compromise for cybersecurity. Due to the network being a graph, the network time might vary on a single computer. We propose a solution to the divergence of network time which guarantees that the network time will converge to stability on each computer.

Network time is potentially useful as an indicator of compromise for cybersecurity. However, if the network time oscillates due to network features, then it will not be useful as an indicator of compromise or for time-series data collection. Our current recommendation is to *not* use network time as an indicator of compromise. We also recommend that all time-series data, such as log files, be adjusted for both network time and time drift.

1 Introduction

When considering network time, we set the time on each computer conditionally independent of the other computers in the network without regard for latency. This means that the network time is passed in a data packet across the network. As the data packet reaches a new computer, the time on that computer is updated to reflect network time.

Network time can be reduced to a machine learning algorithm called belief propagation. For the case of an arbitrary network, we look at loopy belief propagation. Indeed, we propose a new algorithm that is both fault-tolerant and guarantees that network time will converge to an accurate time.

2 Background

Graphical models are a subject of intense study in machine learning [1]. In this case, Bayesian statistical models are written as discrete graphs where the nodes represent random variables and the directed edges represent conditional probability distributions relating the adjacent nodes. There is a commonly used algorithm, known as the sum-product algorithm [2], that computes posterior probabilities for the entire graphical model. A useful sub-case of the sum-product algorithm is that on a tree, which is known as the belief propagation algorithm [3]. In some cases, the belief propagation algorithm is run on an arbitrary graph, in this case the algorithm is called loopy belief propagation [4].

The computational complexity of the sum-product algorithm is well-known to be NP-Complete [5, 6, 7]. Indeed, belief propagation on a tree is efficient [8]; however, belief propagation on an arbitrary network may not even converge [4].

The architecture of the Internet is a packet switching network [9] which is of arbitrary topology, just like a graphical model. This means that the Internet is a graph, where the nodes are computers and the edges are communication links. Any computer with multiple network interface cards (NICs) can route data packets across the network. A special computer is called a *switch*, which is typically implemented entirely in hardware and which routes data packets between all the adjacent computers. A *router* is a computer that does network address translation (NAT) between two sub-networks. A typical *server* can provide either a service, a gateway, or a sub-network for its client computers.

For a computer on the Internet, date and time can be updated automatically. This is most notable when connecting a computational end-point to the network for the first time in a new timezone. For example, when you travel, your tablet will be on the old timezone, until you connect to the Internet in the new timezone. Time, on the Internet, is propagated using two network protocols:

1. dynamic host configuration protocol (DHCP) [10] and

2. network time protocol (NTP) [11].

The network time protocol has been updated to be somewhat appropriate for atomic time [12]. The National Institute of Standards and Technology (NIST) runs several time servers that use the network time protocol [13].

The DHCP protocol is well-known to be loopy, in the sense that it sends data packets across a network that is *not* a tree. On the other hand, the NTP protocol is new enough that many of the queries may be tree-like by coincidence. This coincidence is not a happy one, in that with progress and infrastructure modifications, the tree-like structure will easily become a graph. At which point, the engineers may not notice that there was a fundamental change in the topology of the network queried by the NTP protocol, rendering the queries useless due to non-convergence.

The influence of latency on the accuracy of network time is a difficult discussion. *Latency* is defined as the amount of time it takes a data packet to travel from the source to its destination. Unfortunately, both the source computer and the destination computer must have accurate time, before the latency can be correctly computed. Since the destination computer might even set its time based on that from the source computer, it is not at all clear that we can accurately compute latency. Therefore, the discussion of latency and its impact on network time necessarily involves circular reasoning.

Time, itself, is relative, but can be measured by the international atomic time (TAI). The notion of coordinated universal time (UTC) is a standard for time that can be derived from any time-piece. However, TAI is specifically the standard for atomic time, which are the most precise and accurate clocks. Some colloquially discussed properties of the accuracy of time-pieces suggests that six to ten minutes is the typical error for setting an accurate time. If the error is greater than ± 3 minutes, or if you prefer ± 5 minutes, then the setting of the time-piece is unreliable. An unreliable clock should be synchronized to a time standard such as UTC or to TAI.

Synchronization of time across devices is a difficult topic [14]. One use of synchronization is to register time-series data taken from multiple sensor sources. This means sharing time across the network to disparate sensors.

Each CPU is a clock. The CPU frequency, meaning the cycle time, is designed for a specific frequency of communication on a digital channel. Clock frequency errors can result in dropped bits on the communication channel, and the frequency of the CPU is designed to minimized dropped bits [15]. This means that a CPU clock is allowed to have a drift, meaning the clock can over-all be slow or fast, but the frequency of oscillation is designed to be stable enough for reliable digital communication. Some CPUs are more accurate clocks than other CPUs.

Clock drift can be estimated. The drift is the change, over many minutes, of the clock, whether it is fast or slow. Clock drift can be estimated as a linear first-order property. Any time-series data taken with that clock can be adjusted for the drift [16].

Two clocks can be compared for accuracy over some specified time interval using the Allan variance [17]. The Allan variance compares the frequency differences between the two clocks over a specified time interval. Time-series data can be corrected for frequency problems and for time drift.

The topic of time and computer security is a deep topic. Since each CPU is a clock and the digital communications relies on the frequencies of the CPUs involved, there are many exploits that involve time. The side channel attack [18] and the speculative execution attacks known as Spectre [19] and Meltdown [20] are just three examples.

3 Results

We introduce a generalized method for reducing the network data sharing problem to belief propagation. The network data sharing problem is the problem of sharing data, in the form of variables, across a network. For example, we might discuss sharing a DNS domain name across a subgraph of the Internet. This method of reduction is general and can be applied to arbitrary data shared on a network. These data can be considered global variables for the Internet, or even a large sub-network, such as time, date, year, country, state, province, domain names, etc.

In this manuscript, we apply this reduction to network time. We discuss the conditions under which network time converges on a single Internet-connected computer. We consider the caveats, and we recommend a new algorithm for sharing network time that is guaranteed to converge efficiently with fault-tolerance.

3.1 Reduction

Let $I = (C, R)$ be a graph of the Internet where the nodes C are the computers and the edges R are to destinations that can return query requests. The connectivity graph of the Internet is an undirected graph. However, when a query is sent to a server, s, the edge

$e = (s, r)$ where $r, s \in R$ is directed, and the response is sent to the node, r, that made the request.

Let $G = (V, E)$ be a directed graphical model with variables X_v for node $v \in V$ and edge potentials $Pr[X_v|X_u]$ for directed edge $(u, v) \in E$ and nodes $u, v \in V$. Let $\mathcal{X} = \{X_v|v \in V\}$ be the set of all variables in the graphical model. For an arbitrary graphical model, the sum-product algorithm gives the potential for each clique in the graph. For a poly-tree, which is a graph with tree-width one, the message passing algorithm updates the potentials for each node in the graph. For a poly-tree, the sum-product algorithm, the message passing algorithm, and the belief propagation are all identical algorithms. The update is given by:

$$Pr[X_v] = Pr[X_v] \prod_{u \in V} Pr[X_v|X_u].$$

For a poly-tree, the posterior is computed as a product of the probabilities at each node in the poly-tree:

$$Pr[\mathcal{X}] = \prod_{X_v \in \mathcal{X}} Pr[X_v].$$

With tree-width one, the potentials converge in one pass through the updates.

On the other hand, when the graph is of arbitrary topology, we have a different scenario. The sum-product algorithm is known to be exact on arbitrary graphs, but its running time is exponential. Our alternative is to use message passing, which is only one iteration of loopy belief propagation. For each message passing iteration, the potentials are updated. With multiple passes through the graph, we get the loopy belief propagation algorithm where message-passing updates are used to update the potentials until they converge. We give these updates as:

$$\beta_v = \psi_v \prod_u \delta_{u \to v}$$

where β_v are the approximate potential for node v, ψ_v is the approximate potential for v from the previous iteration, and $\delta_{u \to v}$ are the approximate potentials assigned to the directed edges $(u, v) \in E$.

Generalized Reduction. For a multinomial random variable which is global to some portion of the Internet, we will give a reduction from the network data sharing problem to loopy belief propagation. In this case we have the network graph, $I = (C, R)$, which we use as the graph that describes the topology of the graphical model.

When a query is sent from a source node, r, to a server, s, that responds to the request, this is a directed edge $(s, r) \in R$. Our graphical model for this reduction has variables $\mathcal{X} = \{X_r|r \in R\}$. For the reduction, add the multinomial variable X_r to our graphical model. The potential $\delta_{s \to r} \simeq Pr[X_r|X_s]$ is the conditional probability of the source node r receiving a particular response from the server, s, that it queried. The message passing update to the potential for r is

$$\beta_r = \psi_r \prod_s \delta_{s \to r}.$$

At each iteration of (loopy) belief propagation this update gives a new approximation to β_r based on all previous iterations. The information propagates across the graph according to the number of updates and the tree-width of the graph.

This algorithm is only guaranteed to converge on trees. When we use this algorithm on graphs and run it until convergence, we call the resulting algorithm loopy belief propagation. Unfortunately, loopy belief propagation provably does not always converge [4].

This generalized reduction applies to any "global" variable on the network or on a sub-network. These types of variables include data such as: date, time, DNS queries, and reverse DNS, among others.

Network Time. We will now apply this reduction to the special case of network time. Network time will be stored on each computer as a multinomial random variable with 86400 values, one for each second of the 24 hour day. Then, as the day proceeds, we can periodically clear the time variables and do message passing iterations to obtain the accurate time on each computer. The period with which we update the time can be once every six minutes, as suggested in Section 3.3, or could once a day, once a week, once a month, or on demand.

For our network graph, we again have $I = (C, R)$ with directed edges $(s, r) \in R$. For graphical model multinomial variable X_r having value k, the updates are a product over all the adjacent variables and all their values ℓ having positive potential. The updates for belief propagation for multinomial random variables are as follows:

$$\beta_r^k = \psi_r^k \prod_s \prod_\ell \delta_{s \to r}^{\ell k}$$

where r is the node being updated, s are nodes adjacent to r, k is the value being updated for r, and ℓ are the values in the adjacent node, s, that contribute positive probability. These updates are multiplicative.

Provided that each node has probability one for exactly one event, then this model for network time is equivalent to the existing implementations. Network time is currently propagated around the network as a single DHCP data packet with the current time at the sending node.

Unfortunately, as we know from loopy belief propagation, our multinomial time variables do not always converge in probability [4]. Similarly, the DHCP implementation of network time will not always converge to a consistent time, even at a single node. This is due to the structure of the network and the probabilities for updates by message passing.

3.2 Guaranteed Convergence

If we use the depth first search (DFS) algorithm on the Internet, marking the visited nodes, we get a tree. Network time on a tree converges, which is a corollary of belief propagation converging on a poly-tree. This guarantees that the time will be accurate on all nodes in the network. However, this approach has a fault-tolerance problem. Since there is only one tree-edge that delivers the time to each computer, any link failure (i.e. failure of communication)

will result in some computer not receiving the network time. In order to fail gracefully, we need to do depth first search from multiple roots in the network. Such an approach will provide the redundancy necessary to fail-gracefully in presence of communication link failures.

3.3 New Algorithm for DHCP Network Time

In the network layer, each computer has a single time set, for the whole computer. Along with the time we need to set a color for the time. Now, we can imagine a server with several NICs that lead to several different networks. This computer could receive a time from any of the NICs via DHCP. Likewise, NTP can deliver a time over any of the NICs. Under both protocols, we need to use the same algorithm, which we will elucidate, next.

For each time server, we assign ten colors to each contiguous six minutes of time. Each DHCP packet with a time has the color that corresponds to the six minute window in which it falls. The colors are re-used each hour. The DHCP client remembers the color for six minutes, and then clears the color. When a DHCP time packet arrives at a DHCP client, the color of the packet is compared to the client's color. Only if the packet color is different than the stored color, then the packet's time is recorded locally on the client.

Now each time server will have its own colors, which need to be some random permutation of the ten colors. From a given DHCP client, there may be routes to multiple time servers. However, since the time on the client is only updated when the color of the packet is different than the current color on the client, this means that each subgraph of the Internet will have the same color for each time window in which that color is relevant. On each of these subgraphs the time will be updated according to a DFS tree, because of the colors of the nodes mark visited nodes.

Due to the multiple time servers, and the multiple colors, we will have multiple DFS trees on each relevant subgraph of the Internet. This provides the redundancy. If one communication link fails, there can be another time window with another time source for that subgraph.

The relevant time windows are actually a composition of the time windows on all the time servers. This is because the time servers may be out-of-synch with each other. So the relevant time windows now become the intervals in which the colors on all the time servers are static. The change in color on any time server will begin a new relevant time window for the entire Internet. This results in the update of time for the connected subgraph of the Internet having different colors from the color on the server that changed color.

This algorithm is DFS on sub-graphs of the Internet with redundancy every relevant time window, so that a different DFS sub-tree can be used in at least each six-minute time window. The proofs are left as an exercises to the reader.

Likewise, the convergence properties are straight-forward. Since DFS produces a tree, the network time is guaranteed to converge within each relevant time interval.

4 Conclusions

Network time is a "global" variable for the Internet. The accuracy of network time can be modeled using a graphical model. The network protocols that update network time on each computer in the Internet are approximated by the belief propagation algorithm on the graphical model of the Internet. This means that network time may not converge.

We offer a new algorithm for updating network time that is guaranteed to be fault-tolerant and to converge, meaning that the time set on each computer will be consistent with some time server. The danger of using the existing protocols is that the time set on one computer might oscillate due to network propagation problems. These problems are provable by reduction to loopy belief propagation on a graph.

We introduce a generalized reduction for data shared across a network. The reduction applies to variables such as time, date, year, country, DNS domain names, reverse domains, etc. Our current focus is on network time and its applications for collecting time-series data from disparate sensors, for comparing security log files, and for use as an indicator of compromise for cybersecurity.

Using time as an indicator of compromise for cybersecurity is an interesting prospect. If the time on each node of the network is updated by regularly scheduled data packets, it is possible that the time may not converge. On the other hand, if the data packets are sent by a tree-like portion of the network, then the time is guaranteed to converge. In this latter case, the time may be a useful indicator of compromise. Otherwise, if the time does not converge, then it might oscillate in a strange fashion that has nothing to do with compromise.

The use of time as a tool for exploit is an equally fraught topic. Exploiting bit errors that occur with CPU frequency errors has already been demonstrated in the side channel attack. Likewise the latency evident in the speculative execution attacks also exploit CPU frequency properties.

Biography

Dr. Kirkpatrick has a bachelor's in computer science from Montana State University-Bozeman, a master's and a Ph.D. in computer science from the University of California, Berkeley. Dr. Kirkpatrick is a Professor Emeritus of Computer Science from the University of Miami and is an expert in deterministic and statistical computer algorithms. His main application area is the field of computational biology. Recently, he has specialized in algorithms for computer systems and computer security.

References

[1] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[2] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*, pages 348–363. MIT Press, 2009.

[3] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*, pages 364–371. MIT Press, 2009.

[4] Y. Weiss. Correctness of local probability propagation in graphical models with loops. *Neural Computation*, 12:1–41, 2000.

[5] F. Cooper Gregory. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42(2-3):393–405, 1990.

[6] A. Piccolboni and D. Gusfield. On the complexity of fundamental computational problems in pedigree analysis. *Journal of Computational Biology*, 10(5):763–773, 2003.

[7] J. Li and T. Jiang. An exact solution for finding minimum recombinant haplotype configurations on pedigrees with missing data by integer linear programming. In *Proceedings of the 7th Annual International Conference on Research in Computational Molecular Biology*, pages 101–110, 2003.

[8] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[9] Jonathan S. Turner. Design of a broadcast packet switching network. *IEEE transactions on communications*, 36(6):734–743, 1988.

[10] R. Droms. Automated configuration of TCP/IP with DHCP. *IEEE Internet Computing*, 3(4):45–53, 1999.

[11] David L. Mills. Network time protocol (version 2) specification and implementation. *DARPA Network Working Group Report RFC-1119, University of Delaware*, 1989.

[12] Judah Levine and David Mills. Using the network time protocol (NTP) to transmit international atomic time (TAI). Technical report, National Inst of Standards and Technology, Boulder, CO, Time and Frequency Div, 2001.

[13] NIST-TIME. http://tf.nist.gov. Accessed July 2020.

[14] Qing Ye, Yuecheng Zhang, and Liang Cheng. A study on the optimal time synchronization accuracy in wireless sensor networks. *Computer Networks*, 48(4):549–566, 2005.

[15] Stefano Bregni. Measurement of maximum time interval error for telecommunications clock stability characterization. *IEEE transactions on instrumentation and measurement*, 45(5):900–906, 1996.

[16] Stefano Bregni. *Synchronization of Digital Telecommunications Networks*. John Wiley & Sons, 2002.

[17] William J Riley. Handbook of frequency stability analysis (no. nist sp 1065). 2008.

[18] Steven J Murdoch. Hot or not: Revealing hidden services by their clock skew. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 27–36, 2006.

[19] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. In *40th IEEE Symposium on Security and Privacy (S&P'19)*, 2019.

[20] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown: Reading kernel memory from user space. In *27th USENIX Security Symposium (USENIX Security 18)*, 2018.