Synchronizing Internet Clock frequency protocol (sic)
draft-alavarez-hamelin-tictoc-sic-05

Abstract

   Synchronizing Internet Clock Frequency specifies a new secure method
   to synchronize difference clocks on the Internet, assuring smoothness
   (i.e., frequency stability) and robustness to man-in-the-middle
   attacks. In 90% of all cases, Synchronized Internet Clock Frequency
   is highly accurate, with a Maximum Time Interval Error less than 25
   microseconds by a minute. Synchronized Internet Clock Frequency is
   based on a regular packet exchange and works with commodity terminal
   hardware.

Requirements Language

   The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
   "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
   document are to be interpreted as described in [RFC2119].

Status of This Memo

Table of Contents

1. Introduction

   There are different types of clock synchronization on the Internet.
   NTP [RFC5905] remains one of the most popular because a potential
   user does not need any extra hardware, and it is practically a
   standard in most of the operating systems distributions. Its working
   principle relies on time servers having some kind of precise clock
   source, like atomic clocks or GPS based. For most of the needs, NTP
   provides an accurate synchronization. Moreover, NTP recently
   incorporates some strategies oriented to avoid man-in-the-middle
   (MitM) attacks. NTPs potential accuracy is in the order of tens of
   milliseconds.

Synchronizing Internet Clock frequency (sic frequency) is a protocol providing synchronized difference clocks in two endpoints connected to the Internet. While synchronized absolute clocks aim on a measurement of exact time differences between them, synchronized difference clocks allow measurements during identical time intervals at two locations. This is useful if loads, packet loss or a variation in delay is to be measured.

The sic frequency design is close to TSClocks (see below) but it takes advantage of statistics to perform better. sic frequency synchronization relies on Internet based delay measurements. Route changes are frequent, so we include its detection. Finally, our implementation also contemplates the protection to MitM attacks, including the signature of measurements in each packet. sic frequency does neither put constrains on the quality of a server's clock, nor does it require a limitation of the distance of synchronized end systems.

Another proposal is the TSClocks [ToN2008], which take advantage of the internal computers' clock. This work has been shown a very interesting solution because it is not expensive and can be used in any computer connected to the Internet. This solution was proposed in the beginning at LAN (Local Area Network) level, and then it has been extended to other situations. In [ToN2008] authors report a difference clock error of about half of hundred of microseconds for a WAN connection with 40ms of RTT (Round Trip Time).

When accuracy and stability are needed, further options arise, e.g., the PTP clock [RFC8173] (this mechanism was also defined as the IEEE Std. 1588-2008). The PTP clock however incorporates specialized hardware to provide a highly accurate clock, which is required in each point to be synchronised. Also the GPS (Global Position System) requires specialized hardware in every point of measurement. While GPS may be less expensive than PTP, the GPS unit requires a sky clear view for working. The latter may be costly or impossible in some locations.

Finally, we mention the [ITU-G.8260] shows a methodology to measure delays in networks. It is based on filtering that selects some packets to perform the delay computation. The packet selection is based on the minimum and average RTT, and we show that both of them have some statistical problems to determine (see Section 2).

2. sic frequency protocol overview

Synchronizing Internet Clock frequency (sic frequency) is a protocol providing synchronized difference clocks in two endpoints connected to the Internet. Synchronized difference clocks allow measurements

during identical time intervals at two locations. This is useful if
loads, packet loss or a variation in delay is to be measured. The
model of typical Internet time-measurement is shown in Figure 1.

```
                               XXXXXXXX  XXXXXXXX
                           XXXXXX       XXX        X
                          XX                      XXX
        +----+---------+XX                         XXXX
             |           XX                          XX
             |           X          Internet         XX
             |           XX                         XXX
          +--+------+    XXXXXX                     XX+---------+------+
          |        |        X                  XX              |
          | Client |       XX                  XXX             |
          |        |        XX XXX        XXXXX    XX      +---+----+
          |        |         XXX  XXXXXXX    XXXXXX        |        |
          +---------+                                      | Server |
                                                           |        |
                                                           |        |
                                                           +--------+
```
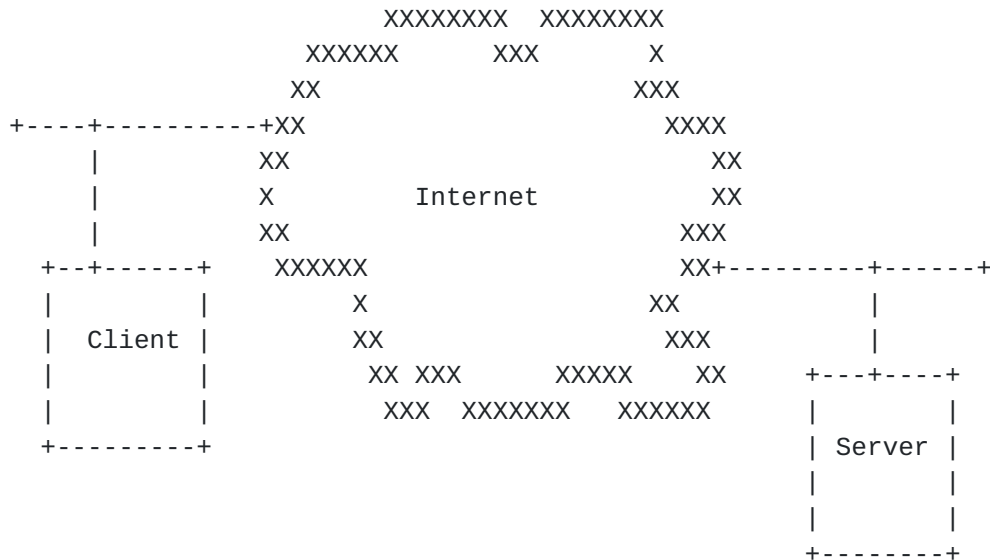
                Figure 1: The clock synchronization of sic.--

In this model, sic frequency performs measurements with packets in
the way shown in Figure 2.

```
                      t2        t3
        Server +--------------@-------*------------------------------->
                             /         \_                  C_s [s]
                            /            \_
                           /               \_
                          /                  \_
                         /                     \_
                        /                        \_
                       /                           \_
                      /                              \_
                     /                                 \_
                    /                                    \_
                   /                                       \_
                  /                                          \_ C_c [s]
        Client +---*-----------------------------------------@------>
                 t1                                          t4
```

                  Figure 2: Time line of packets.--

Here, C_s is the server clock, C_c is the client clock and t1...t4
are timestamps.

Figure 2 shows a horizontal time line for client and server. The
diagonal lines depict a packet traversing some physical space (wires,
routers, and switches). The packet travel times are not assumed to
be identical, because routes and background load may differ in each
direction.

The difference between the client clock C_c and the server clock C_s
can be modeled as:

$$C\_c = C\_s + phi \qquad\qquad\qquad\qquad ,$$

$$phi(t) = C\_c(t) - C\_s(t) \ , \qquad\qquad (1)$$

where phi is the absolute clock difference. If RTT is constant (i.e.
little or no background load) and routes are symmetric in both
directions, the difference between clocks can be computed as:

$$phi[c\text{-}>s] = t1 - (\ t2 - RTT/2\ ) \ , \qquad (2)$$

$$phi[c<\text{-}s] = t4 - (\ t3 + RTT/2\ ) \ , \qquad (3)$$
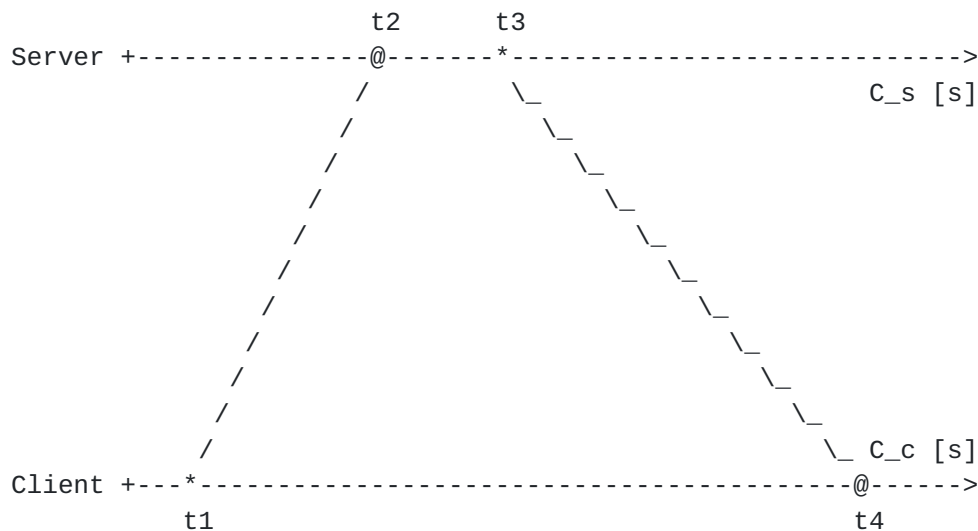
and phi[c->s] = phi[c<-s]. The general equation for the RTT is:

$$RTT = (\ t2 - t1\ ) + (\ t4 - t3\ ) \ . \qquad (4)$$

Computing Equations 2 and 3 for the this simplified case allows
calculation of phi as a function of RTT. Note that if routes are not
symmetrical it is impossible to determine the absolute clocks'
difference.

The sic frequency protocol is based on statistics, background
traffic- and network behavior observations. The RTT between two
endpoints follows a heavy-tailed distribution. An alpha-stable
distribution shows as one possible model [traffic-stable]. This
distribution can be characterized by four parameters: the
localization "delta," the stretching "gamma," the tail "alpha," and
the symmetry "beta," [alfa-estables]. The location parameter is
highly related to the mode of the distribution: delta > 0. The
stretching is related to the dispersion: gamma > 0. The symmetry, -1
<= beta <= 1, indicates if the distribution is skewed to the right
(the tail decays to the left) for positive values or the opposite
direction for negatives ones. Finally, the tail alpha, defined in
(0,2], indicates if the distribution is Gaussian one when alpha=2, a
power law without variance for alpha <2, and also without statistic
mean for alpha<1. The alpha-stable distribution is the
generalization of the Central Limit Theorem for any distribution
(i.e., it includes the cases without variance or mean).

Then, the phi(t) estimation involves the subtraction of two alpha-
stable random variables, which yields on another alfa-stable
distribution but symmetrical [alfa-estables]. Due to the
characteristic of this result, i.e., a fixed mode and symmetry, a
good estimator of the mode is the median.

Therefore, sic performs periodic measurements to infer the difference
of two clocks in the Internet taking advantage of the empiric
observations. The periodicity of RTT measurements is set to 1
second.

The parameters of the simple skew model [ToN2008] are estimated by
the following equation:

$$phi(t) = K + F * t ,   (5)$$

where phi(t) = C_c - C_s, K is a constant representing the absolute
difference of time of client clock C_c and server clock C_s, and F is
the rate parameter. As sic frequency is a difference clock, we only
estimate the frequency parameter "F."

Note that the "K" parameter cannot be estimated using just endpoints
measurements. Estimating the "K" parameter accurately is out of
scope, and we use K=min(RTT)/2, as it used in several synchronization
protocols under the assumption of symmetric paths. Considering the
following asymmetry definition,

```
              t[c->s]
      A = 1 - --------- ,                    (6)
              t[c<-s]
```

where t[c->s] is the minimum delay measured from the client to the
server. The maximum asymmetry A of equation 6 is A=1, which is
unlucky, and this establishes the hard bound for the error of K as
min(RTT): if t[c->s] approaches RTT, t[c->s] approaches zero. The
difference between the two is phi (t), and this difference hence is
close to min(RTT), if A=1. In our experiments the error in
estimation phi(t) was always less than min(RTT)/2.

Another problem with most of the synchronization protocols is the
estimation of the minimum RTT, which depends upon the time-window
within which the RTT is captured. A minimum RTT can only be measured
in the absence of any cross traffic. In a first step, the minimum
RTT measured during a window of 10 minutes (mRTT10m) is captured.
Based on these values, the minimum RTT over a week (mRTTw) is
determined. RTTee is defined as mRTT10m - mRTTw. Figure 3 shows the
the RTT estimation error captured during an experiment where the
minimum latency between probes was 9431 microseconds during one week,

i.e., mRTTw=9431 microseconds. Notice that mRTT10m varies a lot, and
the observed values can be more than 450 microseconds above the
minimum RTT over a week. This error is a consequence of the
statistical behavior of the RTT which can be modeled by the alfa-
stable distribution.

Finally, it is mostly believed there always exist NTP servers at less
than five hops with few milliseconds of RTT, because of the NTP
deployment. In Appendix A we show a typical case in Latin America
region where the RTT differ notably form host in the same city
(Buenos Aires). This example reveals that in some countries could be
not possible to have this desired situation and other synchronization
tools are needed.

```
 Error of the min(RTT)
 [micro-seconds]
   500 +------|-------|------|------|------|-------|------|------+
       |        +        +        +   O   +       +        +        +      |
       |                                *                                  |
   400 |-+                             **                       O        +-|
       |                              *  *             O    O    **  O  O  |
       |                          O    *  *                 **    *   ** ** ** |
   300 |-+                      *  O*O  *  O        O*   * O*O  *    *  O   O *-|
       |            O*     O     O        *  *       * O *       *  *       *|
       |     O        *  O    **     *           *  O      *    *  *    O**       O|
   200 |-*  *       *  *  *  O     *          O  *  O*O     *          O        +-|
       |** O   O *      **  *      *              *  *        O                 |
       |O     *  ***     O     *    *                  *                        |
   100 |-+    O  O        O  *                    O                          +-|
       |                  **                                                |
       |                  **                                                |
     0 |-+       +          +   O   +        +       +        +        +    +-|
       +------|-------|------|------|------|-------|------|------+
        0       50      100     150    200     250      300     350     400
                                                       time [minutes]
```
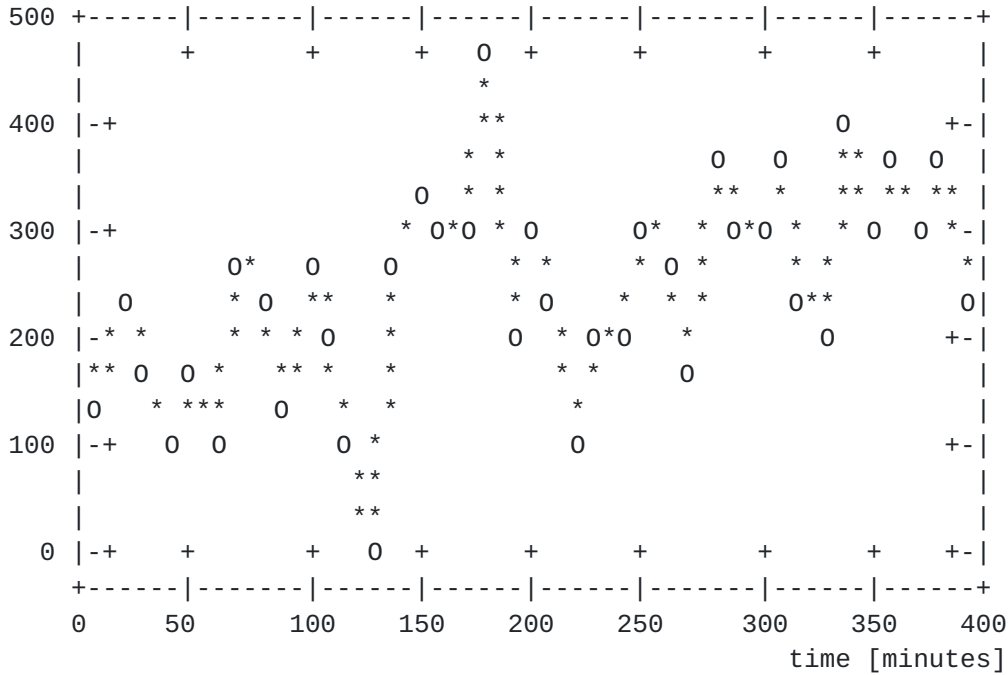
Figure 3: Min RTT error, estimated every 10 minutes along 7 hours.--

The sic frequency protocol estimates phi(t) of Equation 5 using
measurement statistics and taking advantage of the inherent RTT
properties, i.e., the heavy tail distribution and its alfa-stable
distribution model. The basic sic frequency operation is to
periodically send packets, estimate phi(t), and correct the local
clock with:

$$t\_c = t + phi(t) ,\qquad\qquad (7)$$

where t_c is the corrected time and t the local clock time (notice
that phi(t) is calculated according to Equation 1).

The sic protocol also detects route changes by seeking a non-
negligible difference between the minimum RTT of the actual and past
round trip measurement. The next section also discusses different
mechanisms to detect route changes by RTT evaluation.

3. The formal definition of sic frequency protocol

Section 3.1 presents the sic frequency algorithm. In addition,
parameters and their definitions are introduced. Finally, formal
packet formats are provided.

The sic frequency protocol MUST sign the packets with the
deterministic Elliptic Curve Digital Signature Algorithm (ECDSA)
specified by [RFC6979] to protect sic frequency from MitM attacks.
To avoid delays when a packet is signed, sic frequency signs them in
a deferred fashion. That is, in each packet carries the signature of
the previous packet (see algorithms in Figure 6 and Figure 5).

3.1. Algorithm description

sic frequency implementations MUST support the formal description
specified by this section. Once activated, the sic frequency
protocol MUST operate permanently while a client and a receiver
exchange measurement packets. sic frequency works with three states:
NOSYNC, PRESYNC, and SYNC. These states are triggered by the
variables errsync, presync, and synck.

Lines 1 to 4 of the pseudocode in Figure 4 initialize the required
data structures needed and set the sic frequency state to NOSYNC. In
NOSYNC state, a complete measurement window estimates phi's by
Equation 2 (see line 8). Notice that also Equation 3 can be used, or
an average of both Equations. During the experiments, using a single
equation only resulted in estimations with a smaller error. The
possible explanation is that measurements are affected by the same
type of traffic.

The median of the measurement window is also computed in line 9,
while lines 10-12 are used to verify if there is a path change in the
measurements. When an appreciable difference is detected (bounded by
errRTT) in line 13, the "else" clause is executed and the systems re-
initiates the cycle (see lines 17-22). Notice that line 13 verifies
if the absolute value of the minimum RTTs is lower than a percentage
of minimum over the complete RTT window.

The sic frequency algorithm specification is presented by three
tables of pseudocode. The parameters are explained after the third
table.

```
========================================================================
|                    sic frequency algorithm                          |
========================================================================
1   Wmedian <-0, Wm <-0, WRTT <-0, actual_m <-0, actual_c <-0
2   presync <- INT_MAX - P, epochsync <- INT_MAX - P, n_to <-0
3   synck <- false, errsync <- epoch, set(0, 0, NOSYNC), e_prev<-epoch
4   send_sic_packet(SERVER_IP, TIMEOUT)
5   for each timer(RUNNING_TIME) == 0
6   |    (epoch, t1, t2, t3, t4, to) <- send_sic_p(SERVER_IP,TIMEOUT)
7   |    if (to == false) then
8   |    |    Wm <- t1 - t2 + (t2 - t1 + t4 - t3)/2
9   |    |    Wmedian <- median(Wm)
10  |    |    WRTT <- t4 - t1 size(W)
11  |    |    RTTf <- min(WRTT[size(WRTT)/2,size(WRTT)])
12  |    |    RTTl <- min(WRTT[0,size(WRTT)/2])
13  |    |    if ((|RTTf - RTTl| <= errRTT * min(WRTT)) then
14  |    |    |    if (epoch >= presynck + P))  then
15  |    |    |    |    presynck <- true
16  |    |    |    end if
17  |    |     else
18  |    |    |    synck <- false, Wmedian <- 0
19  |    |    |    Wm <- 0, errsync <- epoch, n_to <- 0
20  |    |    |    epoch_sync <- INT_MAX - P, pre_sync <- INT_MAX - P
21  |    |    |    set(0, 0, NOSYNC)
22  |    |    end if
23  |    |    if ((synck == true) && (epoch >= epochsync + P)) then
24  |    |    |    (m, c) <- linear_fit(Wmedian)
25  |    |    |    actual_c <- c
26  |    |    |    actual_m <- (1-alpha) * m + alpha * actual_m
27  |    |    |    epochsync <- epoch,  n_to <- 0
28  |    |    |    set(actual_m, actual_c, SYNC)
29  |    |     else
30  |    |    |    if (epoch == errsync + MEDIAN_MAX_SIZE) then
31  |    |    |    |    presync <- epoch
32  |    |    |    end if
33  |    |    |    if (epoch >= presync + P) then
34  |    |    |    |    (actual_m, actual_c) <- linear_fit(Wmedian)
35  |    |    |    |    synck <- true , epoch_sync <- epoch
36  |    |    |    |    set(actual_m, actual_c, PRESYNC)
37  |    |    |    end if
38  |    |     end if
39  |     else
40  |    |    to <- false
41  |    end if
42 end for
========================================================================
```

                    Figure 4: Formal description of sic.--

Several conditions should be verified to pass from NOSYNC to PRESYNC.
First, the "else" condition of line 29 should occur, and also the
elapsed time between errsync and actual epoch should be
MEDIAN_MAX_SIZE (30-32). Therefore, when it also P time is passed
form presync, the condition on line 33 is true, and the system
arrives at PRESYNC, providing an initial estimation of phi.

Then, if there is no route change, the condition in line 14 will be
true when the time was increased in another P period. Then, the
system is in SYNC state, and it provides the estimation of phi(t) in
line 28. Notice that every P time the estimation of phi(t) is
computed unless a route change occurs (lines 13 and 17-22).

The function in line 6: (epoch, t1, t2, t3, t4, to) <-
send_sic_packet(SERVER_IP, TIMEOUT), has a special treatment. It
sends the packets specified in (Section 3.3), which have signatures.
To avoid the processing delay caused by the signature computation, we
implemented a policy to send the signature of the previous packet,
and if an error is detected, we can stop the synchronization just one
loop ahead.

Figure 5 illustrates how the client side MUST implement the function
send_sic_p (SERVER_IP, TIMEOUT). This function computes the
timestamp t1 in line 1, build and send the UDP packet in lines 2-3.
Then, if there is no timeout, it calculates the t4 timestamp (line
5), and if no packets were lost, verifies the signature of the
previous one in lines 8-18. If the signature is not valid with the
received certificate, then the system MUST change to NOSYNC state
immediately (see line 11). NOSYNC state MUST also be set, if the
limit of time without receiving packets MAX_to is reached. Finally,
it stores the received packet into prev_rcv_pck (a global variable)
to use in the next packet (line 19). Notice that n_to, the lost
packets, is a global variable, as well as the epoch of the previous
packet: e_prev.

```
==========================================================================
|                  function: send_sic_p(server, TIMEOUT)                 |
==========================================================================
1  t1 <- get_timestamp()
2  sic_P <- sic_pck(t1, 0, 0, prev_sig)
3  (to, rcv_sic_pck) <- send(sic_P,UDP_PORT, SERVER_IP, TIMEOUT)
4  if (to == false) then
5  |   t4 <- get_timestamp()
6  |   epoch <- trunc_to_seconds(t1)
7  |   prev_sig <- get_signature(sic_P)
8  |   if (epoch - e_prev <= RUNNING_TIME) then
9  |   |   if (n_to < MAX_to) then
10 |   |   |   if (verify(prev_rcv_pck,rcv_sic.CERT) == false) then
11 |   |   |   |   set(0, 0, NOSYNC)
12 |   |   |   else
13 |   |   |   |   n_to <- 0,  e_prev <- epoch
14 |   |   |   end if
15 |   |   else
16 |   |   |    set(0, 0, NOSYNC)
17 |   |   end if
18 |   end if
19 |   prev_rcv_pck <- rcv_sic_pck
20 |   t2 <- rcv_sic_pck.t2
21 |   t3 <- rcv_sic_pck.t3
22 else
23 |   n_to <- n_to + 1
24 end if
25 return (epoch, t1, t2, t3, t4, to)
==========================================================================
```

                    Figure 5: The send_sic_p function.--

   The server sic algorithm is presented in Figure 6. It uses
   prev_sic_P{}, which is a structure to store the received previous
   signatures, indexed by the IP client addresses (CLIENT_add contains
   its IP and UDP port); and the same for prev_sig{} with the previously
   sent signatures. Line 6 verifies either signature is null because it
   is the first packet, or it is a valid signature. In both cases, the
   algorithm process the packet computing t3, building up the sic
   frequency packet, sending it and computing its signature (stored to
   send in the next reply) in lines 7-11. Next, the actual packet is
   stored in the prev_sic_P{} structure, line 13.

```
=======================================================================
|                      sic Server algorithm                           |
=======================================================================
1   prev_sic_P{} <- null, prev_sig{} <-- null
2   while (RUNNING == true) then
3   |    if (receive() == true) then
4   |    |  t2 <- get_timestamp()
5   |    |  prev_sig <- get_signature(prev_sic_P{receive().CLIENT_add})
6   |    |  if (prev_sig == null)  ||
    |    |            (verify(prev_sig, CLIENT_add.CERT) == true)  then
7   [    |    |  t3 <- get_timestamp()
8   |    |    |  sic_P<-sic_pack(t1, t2, t3, prev_sig)
9   |    |    |  send(sic_P, CLIENT_add.UDP, CLIENT_add.IP, TIMEOUT)
10  |    |    |  prev_sig <- get_signature(sic_P)
11  |    |    |  prev_sig{receive().CLIENT_add} <- prev_sig
12  |    |  end if
13  |    |  prev_sic_P{receive().CLIENT_add} <- receive().sic_pack
14  |    end if
15  end while
=======================================================================
```

Figure 6: Algorithm sic for the Server.--

3.2. Protocol definitions

We provide a formal definition of each used constant and variables;
the RECOMMENDED values are displayed in parentheses at the end of the
description. These constant and variables MUST be represented in a
sic frequency implementation. All the types MUST be respected. They
are expressed in "C" programming language running on a 64-bit
processor.

a. Constants used for the sic frequency algorithm (Figure 4)

    1.   RUNNING_TIME: is the period between sic packets are sent (1
         second).

    2.   MEDIAN_MAX_SIZE: is the window size used to compute the
         median of the measurements (600).

    3.   P: is the period between phi's estimation (60).

    4.   alpha: is a float in the [0,1], the coefficient of the
         autoregressive estimation of the slope of phi(t) (0.05).

    5.   TIMEOUT: is the maximum time in seconds that a sic packet
         reply is expected (0.8 seconds).

6.   SERVER_IP: is the IP address of the server (@IP in version 4 or 6).

7.   errRTT: is a float that bounds the maximum difference to detect a route change (0.2).

8.   MAX_to: is an integer representing the maximum number of packet lost (P/10).

9.   CERT: is a public certificate of the other end, it is used to verify signs of the packets.

10.   UDP_PORT: is an integer with the port UDP where the service is running on the server. (4444)

11.   SERVER_IP: is the IP address of the server.

12.   CLIENT_IP: is the IP address of the client.

b. States used for the sic frequency algorithm (Figure 4)

1.   NOSYNC: a boolean indicates that it is not possible to correct the local time.

2.   PRESYNC: an integer indicates that sic is almost (P RUNNING_TIME) seconds from the synchronization.

3.   SYNC: a boolean indicates that sic is synchronized.

c. Variables used for the sic frequency algorithms (Figure 4, Figure 5 and Figure 6)

1.   errsync: is an integer with the UNIX timestamp epoch of the initial NOSYNC cycle. It is used to complete the window or measurements (Wm) to compute their medians.

2.   presync: is an integer with the UNIX timestamp epoch of the initial PRESYNC cycle. It is used to wait until (P RUNNING_TIME) seconds to the linear fit of phi(t).

3.   synck: is an integer with the UNIX timestamp epoch of the initial SYNC cycle. Every P RUNNING_TIME) seconds the phi(t) function is estimated.

4.   epochsync: is an integer with the last UNIX timestamp epoch of synchronization. It is used to compute a new estimation of phi(t), every (P RUNNING_TIME) seconds.

5.   epoch: is an integer with UNIX timestamp in seconds. It
     carries the initial epoch of each sic measurement packet.

6.   t1, t2, t3, t4: are long long integers to store the t UNIX
     timestamps in microseconds.

7.   actual_m : is a double with the slope for the phi(t)
     estimation.

8.   actual_c: is a double with the intercept for the phi(t)
     estimation.

9.   Wm: is an array of doubles of MEDIAN_MAX_SIZE. It stores
     the instantaneous estimates of phi(t).

10. Wmedian: is an array of doubles of P size. It saves the
     computed medians of Wm every RUNNING_TIME.

11. WRTT: is an array of doubles of (2 P) size. It stores the
     calculated RTT of last measurements.

12. RTTl: is a double with the minimum of last P RTTs. It is
     used to detect changes on the route from the client to the
     server.

13. RTTf: is a double with the minimum of previous P RTTs. It
     is used to detect changes on the route from the client to
     the server.

14. n_to: is an integer representing the number of lost packets
     in the actual synchronization window P.

15. e_prev: is an integer with the UNIX timestamp epoch of the
     last valid packet.

16. prev_rcv_pck: is a sic packet structure, the previous
     received one.

3.3. Protocol packet specification

The sic frequency uses UNIX microsecond format timestamps. Regarding
Figure 2, the client takes a timestamp t1 just before it sends the
packet. When the server receives the packet, it immediately computes
t2, and just before it is sent back to the client, it computes t3.
When the client receives the packet, it calculates t4.

   The server does not need the timestamp t1 because the proposed
   protocol synchronizes a client with the server clock. This
   information could however be useful for the server for future use.

   The packets respects the NTP4 fortmat as they are defined in the
   Section A.1.2 of [RFC5905] and the signature of the fields, shown in
   Figure 7. We use the time formats of 64 bits with seconds and
   fraction of seconds. They MUST be sent as UDP data, and it MUST have
   the mentioned fields.

   The client and server certificates SHOULD be valid and signed ones
   (only for experimentation user MAY use autogenerated ones).

```
     +-------------------------------------------------------------+
     |  NTP_packet (t1_c,t2_s,t3_s)  |  Sig_r n-1  |  Sig_s n-1  |
     +-------------------------------------------------------------+
                        Client --> Server


     +-------------------------------------------------------------+
     |  NTP_packet (t1_c,t2_s,t3_s)  |  Sig_r n-1  |  Sig_s n-1  |
     +-------------------------------------------------------------+
                        Server --> Client
```
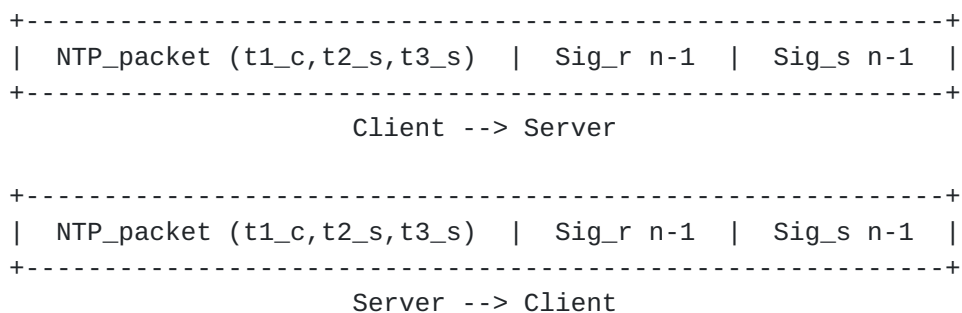
            Figure 7: Packet format for the sic protocol.--

3.4. Minimum sic deployment

   To deploy the sic frequency algorithm, as a minimum a Server and one
   Client are needed. The Server can support multiple clients. The
   maximum number of clients is for further study. The Server clock is
   considered the master one, and all clients synchronize with it. The
   Server side runs sic frequency as a server with a UDP_PORT number, as
   specified by the algorithm shown in Figure 6.

   Client sic runs the algorithm shown in Figure 4 and also SHOULD
   provide the corrected time as

   t = actual_c + actual_m * timestamp   (8)

                              Figure 8

   Different ways of doing this task are possible:

      Providing a client capable of reading the variables actual_m and
      actual_c in shared memory and producing the result of Equation 8.

      Providing a service in a UDP port answering the correct timestamp
      queries with Equation 8.

Other solution.

4. Implementation of sic frequency protocol

In this section we present the prove of the sic concept through some
test that we already performed, and the current implementation of sic
in C language. Our implementation is publicly available
[sic-implementation].

This protocol implements protection against MitM attacks. The
identity of endpoints is guarantee by signed certificates using the
deterministic Elliptic Curve Digital Signature Algorithm (ECDSA)
specified in the [RFC6979]. Server and Client should use signed and
valid ECDSA certificates to ensure their identity, and each side has
is responsible to verify the public certificate of the other side
before to run the algorithm in Figure 4.

4.1. Evaluation

To verify the sic proposal, we tested it using three hosts with GPS
units. The first two were located at Buenos Aires, and the third at
Los Angeles. We slightly modified the algorithm in Figure 4 to
trigger each measurement using the PPS (pulse per second) signal
provided by the GPS unit. Then, recording the client and server
clocks with the PPS signal, we can determine the real phi function of
Equation 1, within the GPS error (it is several orders of magnitude
smaller than the error of the sic frequency protocol).

We use MTIE defined as follows (Maximum Time Interval Error, see
[ToIM1996]):

$$MTIE = \max [phi(t')] - \min [phi (t)] , \quad (9)$$

for every t' and t in the interval [t,t+s]; and we chose s=60
seconds. We first used two host (RaspBerriesPI-2) connected back to
back to analyze the minimum achievable precision, yielding a MTIE of
15.8 microseconds for the 90 percentile. Then, we selected two real
cases of study, one national and other international. In Figure 9 we
show the result of the MTIE, evaluated in 60 seconds intervals, for
the experiment Buenos Aires-Buenos Aires (RTT of 10ms) and Buenos
Aires-Los Angeles (RTT of 198ms). The percentile 90 corresponds to
18.35 microseconds for the Buenos Aires case, and 25.4 microseconds
for the Los Angeles case. The percentile 97.5 corresponds to 30
microseconds for the Buenos Aires case, and 42 microseconds for the
Los Angeles case. We display the quartiles in Figure 10. These
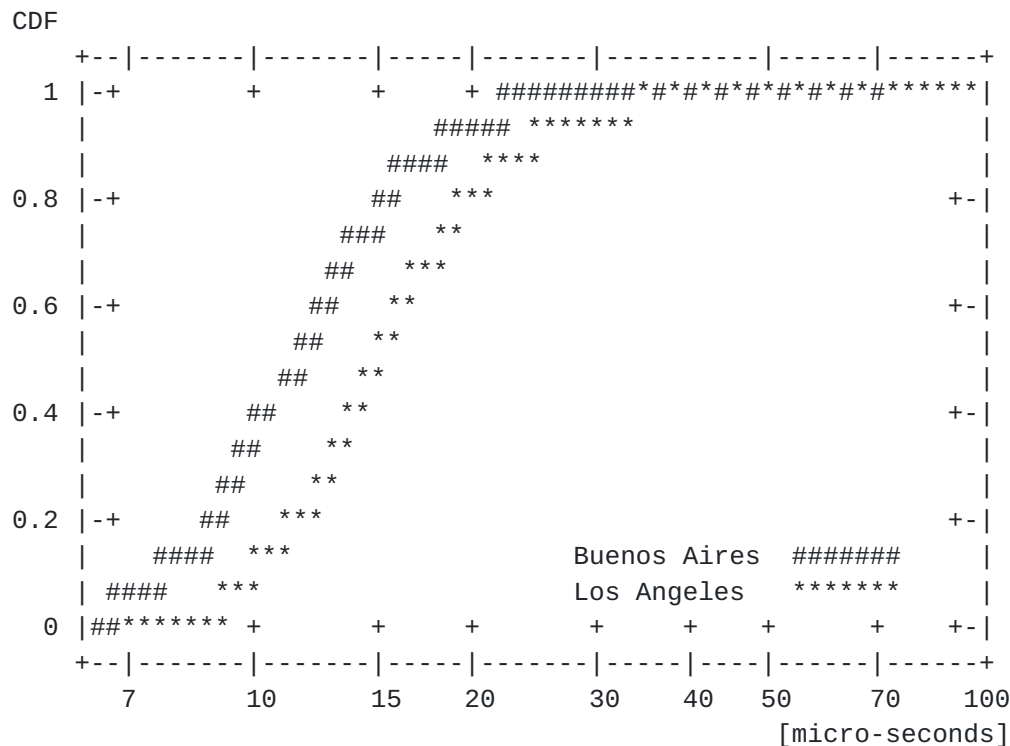measurements were performed during a week in each case.

```
     CDF
         +--|-------|-------|-----|-------|----------|------|------+
       1 |-+        +        +       + ##########*#*#*#*#*#*#*#*#*******|
         |                        ##### *******                        |
         |                      ####   ****                            |
     0.8 |-+                    ##    ***                         +-|
         |                    ###    **                              |
         |                  ##    ***                                |
     0.6 |-+                ##   **                               +-|
         |                ##    **                                  |
         |              ##    **                                    |
     0.4 |-+          ##     **                                  +-|
         |           ##     **                                      |
         |         ##      **                                       |
     0.2 |-+       ##    ***                                     +-|
         |      ####  ***              Buenos Aires  #######       |
         | ####    ***              Los Angeles    *******       |
       0 |##******* +        +      +       +     +     +      +  +-|
         +--|-------|-------|-----|-------|-----|----|------|------+
            7       10      15    20      30    40   50     70   100
                                                      [micro-seconds]
```

Figure 9: Cumulative distribution function of the MTIE (60s).--

| | Buenos Aires (10ms) | Los Angeles (198ms) | local NTP4 (12ms) |
|======|=====================|=====================|===================|
Q3	14.69	19.29	9059
Q2	11.60	14.93	5245
Q1	9.41	12.26	3338

Figure 10: Table with MTIE quartiles for two RTT sic cases, and for a
          local NTP4 system (the numbers indicate microseconds).--

We also conducted another test for NTP4 in Buenos Aires, Argentina.
We used a host with GPS, whose PPS signal triggered a process to log
actual timestamps. This host was also running NTP4 with the server
time.afip.gov.ar, also located at Buenos Aires city, with an average
RTT of 12ms. Applying the same process of the previous cases, we
obtained that the following quartiles Q3: 9.1ms, Q2: 5.2ms, and Q1:
3.3ms for the MTIE of the NTP4 measurements (also reported in
Figure 10). Finally, the percentile 90 of the NTP4's MTIE is 11.1ms.

The comparison of NTP4 with frequency sic shows that this new method
performs two orders of magnitude better.

5. Conclusions

This document presents the sic algorithm to synchronize host clock
frequency using the Internet and resistant to MitM attacks. It also
shows the complete specification, implementation, and experiments
results that support it working principle. In particular, sic
frequency provides a clock rate stability of less than 1ppm for most
of the time.

6. Security Considerations

Following [RFC7384] enumeration of Time Protocols in packet-switched
networks, the proposed encryption of timing packets, based on a
mechanism of secure key distribution, provides the following
characteristics:

3.2.1 Packet Manipulation: Prevented by packet signature.

3.2.2 Spoofing: Prevented by packet signature and secure key
distribution.

3.2.3 Replay Attack: Prevented by chain signing of packets.

3.2.4 Rogue Master Attack: Prevented by secure key distribution.

3.2.5 Packet Interception and Removal: If several packets are
removal, the protocol do not arrive to SYNC state.

3.2.6 Packet Delay Manipulation: Not prevented. Future versions
may prevent this using over-specification of timing (using
redundant masters)

3.2.7 L2/L3 DoS attacks: Not prevented. This can be prevented in
future versions using over-specification of timing and redundant
masters time servers.

3.2.8 Cryptographic performance attacks: Not an issue in ECDSA.

3.2.9 DoS attacks agains the time protocol: Prevented by secure
key distribution.

3.2.10 Grandmaster Time source attack (GPS attacks): Not
prevented. Future versions may prevent this using over-
specification of timing (using several time servers) .

3.2.11 Exploiting vulnerabilities in the time protocol: Not
prevented, future vulnerabilities are unknown.

3.2.12 Network Reconnaissance: Not prevented in this version. No
countermeasures were done in node anonymization.

The Packet Delay manipulation is one of the hardest problems to solve
because there exist some smart ways to attack any synchronization
protocol. Even thou, the sic frequency protocol can protect itself
because can identify several attacks of this type, i.e., it is
challenging to mimic traffic behavior. But, this emulation of
behavior can be easy overcomed regarding the distribution of the
RTTs, which have to be a heavy tailed one. This way the analysis
should be studied to assure the implementation of robust and light
test of the raw data.

7. IANA Considerations

This memo makes no requests of IANA.

8. Acknowledgements

The authors thank to Ethan Katz-Bassett, Zahaib Akhtar, the USC and
CAIDA for lodging the testbed of sic frequency.

9. References

9.1. Normative References

[RFC2119]   Bradner, S., "Key words for use in RFCs to Indicate
            Requirement Levels", BCP 14, RFC 2119,
            DOI 10.17487/RFC2119, March 1997,
            <https://www.rfc-editor.org/info/rfc2119>.

[RFC5905]   Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch,
            "Network Time Protocol Version 4: Protocol and Algorithms
            Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010,
            <https://www.rfc-editor.org/info/rfc5905>.

[RFC6979]   Pornin, T., "Deterministic Usage of the Digital Signature
            Algorithm (DSA) and Elliptic Curve Digital Signature
            Algorithm (ECDSA)", RFC 6979, DOI 10.17487/RFC6979, August
            2013, <https://www.rfc-editor.org/info/rfc6979>.

[RFC7384]   Mizrahi, T., "Security Requirements of Time Protocols in
            Packet Switched Networks", RFC 7384, DOI 10.17487/RFC7384,
            October 2014, <https://www.rfc-editor.org/info/rfc7384>.

   [RFC8173] Shankarkumar, V., Montini, L., Frost, T., and G. Dowd,
              "Precision Time Protocol Version 2 (PTPv2) Management
              Information Base", RFC 8173, DOI 10.17487/RFC8173, June
              2017, <https://www.rfc-editor.org/info/rfc8173>.

9.2. Informative References

   [alfa-estables]
              Uchaikin, V. and V. Zolotarev, "Chance and stability:
              stable distributions and their applications.", Walter de
              Gruyter. (Book), 1999.

   [ITU-G.8260]
              ITU, T. S. S. O., "Definitions and terminology for
              synchronization in packet networks (Recommendation ITU-T
              G.8260)", August 2015.

   [sic-implementation]
              Samariego, E., Ortega, A., and J. Alvarez-Hamelin,
              "Synchronizing Internet
              Clocks", https://github.com/CoNexDat/SIC, February 2018.

   [ToIM1996]
              Bregni, S., "Measurement of maximum time interval error
              for telecommunications clock stability
              characterization", Bregni S. Measurement of maximum time
              interval error for telecommunicIEEE transactions on
              instrumentation and measurement. 45(5):900-906, October
              1996.

   [ToN2008] Veitch, D., Ridoux, J., and S. Korada, "Robust
              synchronization of absolute and difference clocks over
              networks.", IEEE.ACM Transactions on Networking (TON)
              17.2 (2009): 417-430, 2009.

   [traffic-stable]
              Carisimo, E., Grynberg, S., and J. Alvarez-Hamelin,
              "Influence of traffic in stochastic behavior of
              latency.", 7th PhD School on Traffic Monitoring and
              Analysis (TMA), Ireland, Dublin,, June 2017.

Appendix A. Example of RTT to NTP servers

   This appendix shows an experiment to measure the RTT and the distance
   in hops from four different points to a time server in Buenos Aires
   city (the capital of Argentina). We did the measures two times from
   the four points, and we used one hundred packets to determine some
   statistical parameters. Next traceroute measurements show that the

number of hops and RTT are very different from each point also
changes a lot. For instance, taking a distinctive look at the STD,
average, and maximum is possible to detect huge variations. We
provide here a case in Argentina, trying to reach an NTP server from
4 different points at the Buenos Aires city.

```
------------------------------------------------------------------------

host1$ mtr -r -c 100 time.afip.gov.ar
Start: Tue Mar 27 19:03:51 2018
HOST: raspbian-server          Loss%  Snt  Last  Avg  Best  Wrst StDev
  1.|-- gw-vlan-srv.innova-red.ne  0.0%  100   2.2  2.8   2.1  37.7   4.9
  2.|-- rnoc5.BUENOS-AIRES.innova  0.0%  100   2.3  3.8   2.1  55.8   7.9
  3.|-- 10.5.10.2                  0.0%  100   2.5  2.6   2.2   3.1   0.0
  4.|-- 200.0.17.104               0.0%  100   3.1  3.1   2.4  13.7   1.1
  5.|-- 172.18.2.53                0.0%  100   4.8  5.7   3.8  12.4   1.7
  6.|-- time.afip.gob.ar           0.0%  100   5.2  5.2   3.8  12.0   1.3

host1$ mtr -r -c 100 time.afip.gov.ar
Start: Tue Mar 27 18:57:06 2018
HOST: raspbian-server          Loss%  Snt  Last  Avg  Best  Wrst StDev
  1.|-- gw-vlan-srv.innova-red.ne  0.0%   50   2.4  2.8   2.0  34.2   4.5
  2.|-- rnoc5.BUENOS-AIRES.innova  0.0%   50   2.1  3.8   2.1  52.8   7.7
  3.|-- 10.5.10.2                  0.0%   50   2.7  2.9   2.2  15.6   1.8
  4.|-- 200.0.17.104               0.0%   50   2.8  3.0   2.3   3.9   0.0
  5.|-- 172.18.2.53                0.0%   50   4.5  5.8   3.8  17.8   2.2
  6.|-- time.afip.gob.ar           0.0%   50   4.7  9.9   4.2 238.5  33.0


------------------------------------------------------------------------

host2$ mtr -r -c 100 time.afip.gov.ar
Start: Tue Mar 27 19:03:47 2018
HOST: ws-david                 Loss%  Snt  Last  Avg  Best  Wrst StDev
  1.|-- 10.10.96.1                 0.0%  100   0.3  0.2   0.2   0.3   0.0
  2.|-- 200.16.116.171             0.0%  100   1.0  5.9   0.6 158.4  22.9
  3.|-- static.33.229.104.190.cps  1.0%  100   1.6  2.5   1.5  80.6   8.0
  4.|-- static.129.192.104.190.cp  0.0%  100   2.1  1.9   1.8   3.0   0.1
  5.|-- 200.0.17.104               1.0%  100   2.0  2.2   1.8   9.4   0.7
  6.|-- 172.18.2.53                0.0%  100   3.2  4.2   3.1  12.0   1.5
  7.|-- auth.afip.gob.ar           0.0%  100   4.2  4.5   3.3   9.8   1.2

host2$ mtr -r -c 100 time.afip.gov.ar
Start: Tue Mar 27 18:57:00 2018
HOST: ws-david                 Loss%  Snt  Last  Avg  Best  Wrst StDev
  1.|-- 10.10.96.1                 0.0%   50   0.3  0.3   0.2   0.7   0.0
  2.|-- 200.16.116.171             0.0%   50   0.9  6.7   0.7 196.5  29.1
  3.|-- static.33.229.104.190.cps  2.0%   50   1.6  1.7   1.5   2.2   0.0
  4.|-- static.129.192.104.190.cp  0.0%   50   2.1  2.0   1.7   2.4   0.0
```

```
  5.|-- 200.0.17.104              0.0%   50    2.0   2.1   1.8    2.6   0.0
  6.|-- 172.18.2.53               0.0%   50    4.8   4.3   3.2    9.1   1.3
  7.|-- time.afip.gob.ar          0.0%   50    4.3   9.4   3.3  234.9  32.7


-------------------------------------------------------------------------

host3$ mtr -r -c 100 time.afip.gov.ar
Start: 2018-03-27T19:03:51-0300
HOST: aleph.local               Loss%  Snt  Last   Avg  Best   Wrst StDev
  1.|-- 10.17.71.254             0.0%  100    4.7  30.3   3.5  280.4  52.4
  2.|-- 10.255.254.250           0.0%  100    2.5  31.1   1.8  285.4  59.2
  3.|-- 209.13.133.10            0.0%  100   30.5  43.9   2.3  237.2  64.9
  4.|-- host169.advance.com.ar   3.0%  100   36.0  64.8   3.1  404.4  86.9
  5.|-- 200.32.33.33             2.0%  100  106.9  70.6   2.8  315.0  80.4
  6.|-- 200.32.34.66             5.0%  100   93.1  56.8   2.7  336.1  74.5
  7.|-- 200.32.33.38             7.0%  100   42.8  58.0   2.9  357.8  76.7
  8.|-- 209.13.139.211           4.0%  100   46.2  56.7   2.8  298.8  69.9
  9.|-- 209.13.139.209           1.0%  100   84.5  57.1   2.6  277.7  72.3
 10.|-- 209.13.166.211           1.0%  100   43.4  63.5   3.2  390.6  78.7
 11.|-- 200.32.34.137            2.0%  100   68.7  64.1   3.7  259.5  75.5
 12.|-- 200.32.33.37             0.0%  100    4.1  56.9   3.3  249.6  64.3
 13.|-- 200.32.34.121            3.0%  100   10.9  65.0   4.1  415.7  85.1
 14.|-- 200.32.33.241            2.0%  100  252.6  61.8   3.8  355.9  74.5
 15.|-- 200.16.206.198           3.0%  100  188.0  54.6   3.1  461.7  74.9
 16.|-- 172.18.2.53              2.0%  100  133.4  53.1   4.3  402.1  69.2
 17.|-- time.afip.gob.ar         4.0%  100   72.5  54.1   4.9  343.2  66.9

host3$ mtr -r -c 100 time.afip.gov.ar
Start: 2018-03-27T18:57:05-0300
HOST: aleph.local               Loss%  Snt  Last   Avg  Best   Wrst StDev
  1.|-- 10.17.71.254             0.0%   50  125.6  88.1   3.7  392.4  79.3
  2.|-- 10.255.254.250           0.0%   50   62.1  54.8   2.1  333.2  68.0
  3.|-- 209.13.133.10            0.0%   50    4.0  33.9   2.4  280.8  51.3
  4.|-- host169.advance.com.ar   2.0%   50    4.1  21.3   2.9  236.7  40.4
  5.|-- 200.32.33.33             2.0%   50    4.5  32.2   3.2  341.3  69.4
  6.|-- 200.32.34.66             4.0%   50    7.7  26.0   3.5  278.8  55.8
  7.|-- 200.32.33.38             2.0%   50    4.8  29.4   3.0  221.3  43.4
  8.|-- 209.13.139.211           0.0%   50   84.8  40.3   3.1  250.4  53.0
  9.|-- 209.13.139.209           0.0%   50   25.1  35.0   2.8  249.2  55.4
 10.|-- 209.13.166.211           0.0%   50    3.7  33.5   2.6  188.9  54.3
 11.|-- 200.32.34.137            0.0%   50    5.6  28.2   3.7  224.3  51.1
 12.|-- 200.32.33.37             0.0%   50    3.7  24.2   3.5  189.5  44.9
 13.|-- 200.32.34.121            0.0%   50    4.7  30.8   4.0  213.2  51.6
 14.|-- 200.32.33.241            0.0%   50   14.4  33.1   3.9  364.6  67.2
 15.|-- 200.16.206.198           0.0%   50    5.0  58.2   3.1  300.7  88.5
 16.|-- 172.18.2.53              0.0%   50    9.4 117.8   4.4  315.1 103.4
 17.|-- time.afip.gob.ar         0.0%   50  199.6 120.2   5.2  484.0  96.2
```

```
host4$ mtr -r -c 100 time.afip.gov.ar
Start: 2018-03-27T19:03:51-0300
HOST: cnet                    Loss%  Snt  Last  Avg  Best  Wrst StDev
 1.|-- 157.92.58.1             0.0%  100   6.6  2.8   0.3  12.8   2.5
 2.|-- ???                    100.0  100   0.0  0.0   0.0   0.0   0.0
 3.|-- ???                    100.0  100   0.0  0.0   0.0   0.0   0.0
 4.|-- host98.131-100-186.static  0.0%  100   5.7  5.6   1.5   9.4   2.2
 5.|-- host130.131-100-186.stati  0.0%  100   6.5  6.3   2.5  10.3   2.2
 6.|-- 200.0.17.104            0.0%  100   2.4  2.7   2.3  15.6   1.4
 7.|-- ???                    100.0  100   0.0  0.0   0.0   0.0   0.0
 8.|-- time.afip.gob.ar        0.0%  100   4.9  7.6   3.9 243.0  23.9

host4$ mtr -r -c 100 time.afip.gov.ar
Start: Tue Mar 27 18:41:40 2018
HOST: cnet                    Loss%  Snt  Last   Avg  Best Wrst StDev
 1.|-- 157.92.58.1             0.0%   50   4.0   1.6   0.3   9.1   1.6
 2.|-- ???                    100.0   50   0.0   0.0   0.0   0.0   0.0
 3.|-- ???                    100.0   50   0.0   0.0   0.0   0.0   0.0
 4.|-- host98.131-100-186.static  0.0%   50   4.7   5.5   1.5  10.9   2.4
 5.|-- host130.131-100-186.stati  0.0%   50   8.4   6.5   2.6  10.5   2.2
 6.|-- 200.0.17.104            0.0%   50   2.5   2.8   2.3  11.0   1.2
 7.|-- ???                    100.0   50   0.0   0.0   0.0   0.0   0.0
 8.|-- time.afip.gob.ar        0.0%   50   4.9   9.2   3.8 226.7  31.4
```

Authors' Addresses

   Jose Ignacio Alvarez-Hamelin (editor)
   Universidad de Buenos Aires - CONICET
   Av. Paseo Colon 850
   Buenos Aires C1063ACV
   Argentina

   Phone: +54 11 5285-0716
   Email: ihameli@cnet.fi.uba.ar
   URI: http://cnet.fi.uba.ar/ignacio.alvarez-hamelin/

David Samaniego
Universidad de Buenos Aires
Av. Paseo Colon 850
Buenos Aires C1063ACV
Argentina

Phone: +54 11 5285-0716
Email: dsamanie@fi.uba.ar

Alfredo A. Ortega
Universidad de Buenos Aires
Av. Paseo Colon 850
Buenos Aires C1063ACV
Argentina

Phone: +54 11 5285-0716
Email: ortegaalfredo@gmail.com

Ruediger Geib
Deutsche Telekom
Heinrich-Hertz-Str. 3-7
Darmstadt 64297
Germany

Phone: +49 6151 5812747
Email: Ruediger.Geib@telekom.de