spring
Internet-Draft
Intended status: Informational
Expires: December 24, 2016

R. Leipnitz, Ed.
R. Geib
Deutsche Telekom
June 22, 2016

A scalable and topology aware MPLS data plane monitoring system draft-leipnitz-spring-pms-implementation-report-00

Abstract

This document reports round-trip delay measurements captured by a single MPLS Path Monitoring System (PMS) compared with results of an IPPM conformant measurement system, consisting of three different Measurement Agents. The measurements were made in a research backbone with an LDP control plane. The packets of the MPLS PMS use label stacks similar to those to be used by a segment routing MPLS PMS. The measurement packets of the MPLS PMS remained in the network data plane.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 24, 2016.

Copyright Notice

Table of Contents

1. Introduction

Deutsche Telekom has implemented an MPLS Path Monitoring System (PMS). The PMS operates on MPLS networks with LDP control plane. Forwarding follows the principles of Segment Routing, i.e. the packets sent by the PMS use stacked transport labels to execute a combination of MPLS paths and finally return to the PMS. The PMS is connected to a research backbone of Deutsche Telekom spanning parts of Germany. One of the new network monitoring features enabled by Segment Routing are round-trip delay measurements purely executed in data plane. Deutsche Telekom captured delays between three IPPM standard conformant Measurement Agents and compared these with delays measured along identical backbone paths by a single PMS. To prove that the same delays were measured the IPPM results were then compared with the PMS results by applying IPPM methodology as specified by [RFC6576]. Some results passed this test, while others did not. The results of both systems seemed to differ by very small and relatively stable latencies. As the research network only offered single paths between the involved routers, processing of different flows in parallel forwarding instances of the routers along the paths offered an explanation. The PMS was used to execute some

measurements whose results at least are not contradicting that assumption.

The results reported here show that a PMS [I-D.ietf-spring-oam-usecase] can be built and operated (also as part of an LDP based MPLS network). To set up packets with proper label stacks, the PMS needs to be aware of the MPLS topology of the network. MPLS topology awareness within an LDP based network requires reasonable effort. Segment Routing will significantly simplify detection of the MPLS topology. Delay measurements where picked here to give an example of a feature which can be supported by a PMS. Others are possible, like checking continuity of arbitrary segmented routed MPLS paths [I-D.ietf-spring-oam-usecase].

The remaining document is organized as follows: Section 2 briefly informs about the PMS and IPPM measurement system implementation. Section 3 introduces the measurement set up within the research network. Section 4 briefly discusses the test by which the measurements were compared. Section 5 informs about the test results and Section 6 about an IPPM error calibration. Section 7 sums up the document.

2. Measurement system implementation

Deutsche Telekom operates an IPPM standard conformant performance measurement system called Perfas+. Deutsche Telekom intends deployment of an MPLS PMS to monitor the IP performance in network segments connecting roughly 1000 edge routers to the IP-backbone. 11 MPLS PMS are supposed to execute backbone to edge performance monitoring. Had the monitoring system been based on IPPM, one IPPM system had been required per edge router.

2.1. A PMS based round-trip delay measurement system

Deutsche Telekom has implemented an MPLS PMS. The PMS is part of an MPLS research and development backbone of Deutsche Telekom. This backbone only supports LDP routing. The PMS works with an LDP control plane. Detecting the MPLS topology of an LDP based MPLS network is more complex, than doing this by Segment Routing. The PMS consists of the following logical components:

- An MPLS Label detection system. It is collecting MPLS routing information from all MPLS routers of the MPLS network by management plane access (see e.g. [LDP-TE], [BCP-TX])

- An MPLS topology database.

- A measurement system able to compose packets executing any combination MPLS Label Switched Paths (MPLS LSP) which are part of the MPLS topology database. The measurement system further is able to measure delays, if the final address information of the measurement packet directs the packet back to the PMS after the MPLS LSPs to be measured have been passed.

- An IGP topology detection system. It is passively listening to IGP routing.

- A measurement system which is complying to [RFC4379].

Note that the final two MPLS PMS functionalities are required if ECMP routed paths should be detected and addressed by [RFC4379] functions. No ECMP routed paths are present between the sites involved in the measurement set up. The role of these components is reduced to detection of operational issues, should the measurement not work as expected.

While the control plane of the network monitored by the PMS is LDP based, the measurement packets used to execute MPLS LSPs apply the forwarding mechanisms as within a Segment Routing network.

2.2. Perfas+ IPPM measurement system

IPPM conformant one-way delay measurements were performed by Perfas+ Measurement Agents. Three Perfas+ Measurement Agents are connected to edge routers at three different sites of the research network. Perfas+ is one of the few IPPM implementations with proven conformance to some standard IPPM metrics, like one-way delay [RFC6808]. Two of the Perfas+ Measurement Agents were synchronized by NTP only. Due to this restriction, the comparison with the PMS measurements are limited to round-trip times (round-trip delays, RTD). As no ECMP routed paths are active between the sites used for test execution, two back and forth Perfas+ one-way delay measurements between two sites were added to result in an RTD value.

3. Test set up

The test set up is shown in the figure below. The PMS and Perfas+ Measurement Agent 1 (PerfMA 1) are connected to the same LER.

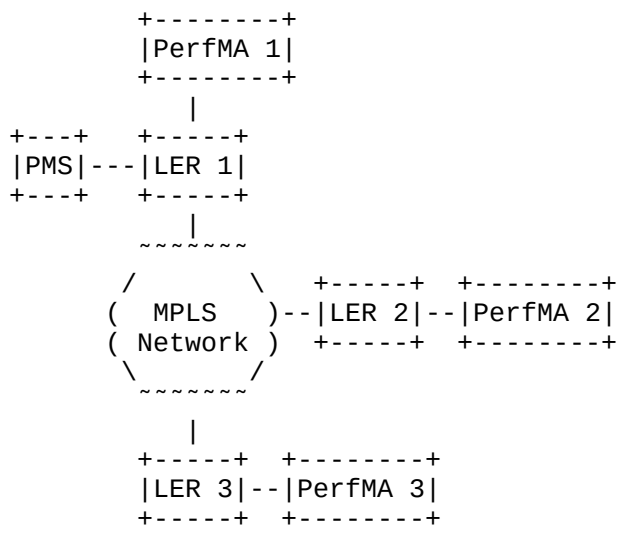

Figure 1: Test set up

The Perfas+ Measurement Agents (MAs) measure the one-way delay to each of the remote Perfas+ MAs. The PMS measures the round-trip delay from LER 1 to LER 2 and back as well the round-trip delay from LER 1 to LER 3 and back. The measurements start and terminate at the PMS, but this segment is omitted here. The round-trip delay from LER 2 to LER 3 is measured along two path combinations by the PMS. The first measurement path is LER 1 to LER 2 to LER 3 and back exactly that way. The round-trip delay LER 1 to LER 2 captured earlier by the PMS is subtracted from the result. The other measurement is LER 1 to LER 3 to LER 2 and back exactly that way. Here, the PMS round-trip delay LER 1 to LER 3 is subtracted to receive the round-trip delay LER 2 to LER 3.

There is a small LAN section causing limited additional latencies for the IPPM measurement. The measurements were executed with an IP packet size of 64 Byte. Perfas is attached by an IP-VPN. The PMS label stack is differing slightly. The assumption is that both differences have minor impact. Note that IPPM metrics expect similar results if differences in measurement set up can be neglected. The sending interval is 10 seconds periodic. A measurement mean is calculated from 10 consecutive measurement packets. The measurements were repeated for 8 hours, resulting in 288 mean values collected per round-trip delay measurement path and measurement system.

The resulting round-trip delays are divided by two and indicate the one-way delay. This seems sound, as there is no path diversity in

the research network and the low standard deviation of the results (single digit [us] figures in all cases, see test results below) indicate that no link was congested.

4. Measurement Result Evaluation

IPPM WG applies the Anderson-Darling-K-Sample (ADK) test to compare up to which temporal resolution the results of two measurements share the same statistical distribution [RFC6576]. To decide, whether Perfas+ and the PMS were measuring identical data, the round-trip delays captured along identical measurement paths were compared by an ADK test. (The ADK test source code is given at Appendix A). Note that the ADK test does not judge accuracy (i.e. it does not test whether the result is close to the true value?), ADK rather judges precision (that the test estimates whether the same value was measured by repeated measurements). As applied here, an RTD sample of Perfas+ was compared with one of the PMS captured along the same path.

To illustrate, how sensible the ADK test is to changes in a measurement environment, a PMS round-trip delay test was set up where all configurations were identical and only packet size was variable. Obviously all paths are identical, so any difference in results is caused by the packet size only (64, 128 and 256 Byte were picked). The ADK test indicated a reasonably high probability that results do not follow the same distribution in roughly half of the cases (i.e. ADK test said that the distribution of round-trip delays captured with packet size of 64 bytes follows a different distribution than the round-trip delays captured with a packet size of 128 Byte).

5. Measurement results

5.1. Round-trip delay measurement and ADK test results

The one-way delays between Perfas MA 1 and Perfas MA 2 calculated on basis of the round-trip Delay and the ADK test results comparing them to the measurement results captured by the PMS are shown in Table 1.

Test metric	PERFAS+	PMS
minimum [us]	691.5	695.5
maximum [us]	701	704.5
mean [us]	695.4	699.6
median [us]	695.5	699.5
standard deviation [us]	1.4	1.7
ADK value		278.445
ADK value with adjustment of mean		1.701
ADK value with adjustment of median		1.982

Perfas+ and PMS OWD measurement results for path LER 1 to LER 2 and ADK test results

Table 1: Perfas+ and PMS OWD measurement results for path LER 1 to LER 2 and ADK test results

The ADK test result is surprisingly good and was not expected a priori. As mentioned, ADK is a very sensible test. When IPPM WG worked on [RFC6808], the packets used by two different IPPM implementation only passed ADK after a network emulator was inserted into the measurement path. As IPPM puts more emphasis on precision than on accuracy, correcting tests samples to result by the same mean for small and constant differences is plausible. Still, the smallest temporal resolution of the standard deviation by which ADK was passed when used to compare two IPPM implementations for [RFC6808] was single digit milliseconds. No network emulator has been used when comparing Perfas+ and the PMS. After adjusting the means, ADK is passed by a temporal resolution of the standard deviation of single digit microseconds!

The one-way delays between Perfas MA 1 and Perfas MA 3 calculated on basis of the round-trip Delay and the ADK test results comparing them to the measurement results as captured by the PMS are shown in Table 2.

Test metric	PERFAS+	PMS
minimum [us]	2991.5	2983
maximum [us]	3008.5	2994.5
mean [us]	2995.7	2988.1
median [us]	2995.5	2988
standard deviation [us]	1.9	2.1
ADK value		231.638
ADK value with adjustment of mean		1.886
ADK value with adjustment of median		2.026

Perfas+ and PMS OWD measurement results for path LER 1 to LER 3 and ADK test results

Table 2: Perfas+ and PMS OWD measurement results for path LER 1 to LER 3 and ADK test results

After adjustment of the means values, also here the ADK test is passed. Comparing Table 1 with Table 2 readers figure can see, that once mean the one-way delay measured by Perfas+ is lower, while in the other case the mean one-way delay captured by the PMS is lower. This behavior was visible in all our measurements. The delays measured per path by one system were always bigger than that of the other along the same path (for all single 10 sample mean values of the time series).

We now compare the one-way delays between Perfas MA 2 and Perfas MA 3 calculated on basis of the round-trip delay and the ADK test results comparing them to the measurement results as captured by the PMS are shown in Table 3.

Test metric	PERFAS+	PMS over LER 2	PMS over LER 3
minimum [us]	3606.5	3551	3542.5
maximum [us]	3659	3568	3558
mean [us]	3611.9	3560.1	3549,8
median [us]	3609	3560	3549,5
standard deviation [us]	8.3	2.9	2.9
ADK value		231.144	231.094
ADK value with adjustment of mean		54.591	56.589
ADK value with adjustment of median		8.915	10.054

Perfas+ and PMS OWD measurement results for path LER 2 to LER 3 and ADK test results

Table 3: Perfas+ and PMS OWD measurement results for path LER 2 to LER 3 and ADK test results

In this case, the ADK test fails (the cause is the difference of the standard deviation, not the mean or median difference). Note that in terms of mean values the difference is around 50 us between Perfas and PMS. The relative error is 1,75%. While ADK indicates that both distributions deviate, human perception may confirm that both results capture delays along the same path.

It is interesting however, that the two PMS measurements deviate in the mean values. And again, the one showing the lower delay does so sample mean measurements. A brief test investigating this symptom was performed. Test and results follow in the next section.

5.2. PMS delay measurements with IP-address variation

The PMS allows to send measurement packets with different destination IP-addresses (routing based on IP-addresses only occurs from LER 1 to PMS and only in this direction). While the IP-address varied, the MPLS Label stack and thus the MPLS path was kept identical. This measurement can only be configured by CLI configuration. Per IP destination address, the mean-value of 10 round-trip delay times was captured. After some measurements the IP-addresses showing the biggest round-trip delay difference were selected for further testing. With these IP-addresses, the test was repeated at different days and daytimes. Overall we had at least 10 more measurement values of every of these IP-addresses. The PMS is connected with two interfaces to two different LERs of the same site. Both interfaces

and LERs respectively were used to perform the measurements. As has been mentioned already, the network does not have ECMP-paths. Table 4 shows the results of the two measurements with the biggest difference in results. The mean delays measured with IP-address a.b.c.0 were the smallest. They were always smaller than those delays captured with IP-address a.b.c.32, which were the biggest. The difference of the mean values from the measurement over the first interface was 19.5 us and 14.4 us over the second interface.

Interface / IP-address	mean [us]	median [us]
one / a.b.c.0	1413.2	1412
one / a.b.c.32	1432.7	1433
two / a.b.c.0	1446.4	1446
two / a.b.c.32	1460.8	1460.5

Table 4: Destination-IP-address variation

Parallel hardware processing within some or all of the routers passed on the measurement paths may be a plausible explanation. Investigating the cause for this behavior was however not the main aim of the test activities documented here. Further activities related to this issue are left to interested research.

6. Error Calibration

Section 3.7. and following of [RFC2679] recommend an error calibration of the (IPPM) measurement clients. The one-way delay of a back-to-back connection of two PERFAS+ clients is measured. Table 5 shows the characteristics of this calibration measurement. The negative values for the one-way delay shown in the table, are physically impossible. The standard deviation is very high. It was decided to calibrate with the round-trip delay which is shown in Table 6. Referring to section 3.7.3 of [RFC2679] there is a systematic error and a random error. The systematic error is the median of the measurement with 49.5 us. The random error is the difference between the median and the 2.5% percentile, which is 17 us. (The random error is the larger absolute value between the median and the 2.5% percentile and the 97.5% percentile; the calculation is |49.5 - 32.5| > |49.5 - 59.5|). The resolution of the PERFAS+ Measurement Agents is 1 us, so the absolute random error is 19 us. So measurement error is 49.5 +/- 19 us. (The synchronization error is 0, as two one-way delays are added, making this error disappear). There was no possibility to calibrate the PMS. The error is assumed to be the same like that of PERAS+, because the PMS is based on the same hardware (and possibly the same host-system).

Test metric	PERFAS+
minimum [us]	-55
maximum [us]	39
mean [us]	-38
median [us]	-23.1
standard deviation [us]	29.4

Table 5: measurement results of one-way delay of back-to-back connection from two PERFAS+ clients at 64 Bytes

Test metric	PERFAS+
minimum [us]	26
maximum [us]	205
mean [us]	49.1
median [us]	49.5
standard deviation [us]	7.6
2.5% percentile [us]	32.5
97.5% percentile [us]	59.5

Table 6: measurement results of both one-way delays of back-to-back connection between two PERFAS+ clients at 64 Bytes

7. Summary

By an IPPM measurement system like PERFAS+ three physical measurement clients are needed to measure the round-trip delay between all sites. With the PMS the same measurements can be performed with only one client. In theory one PMS could monitor a whole MPLS-enabled backbone. The GPS receivers of two IPPM measurement agents were not available, hence the one-way delay could not be captured with the IPPM system PERFAS+. Otherwise a direct comparison with calculated one-way delay values based on the PMS measured values would have been possible. This could be done in future. The results shown in Section 4 indicate, that the PMS measurements equal those captured by an IPPM conformant measurement system. The ADK test is successful by comparing the measurement values of the round-trip delays for packets with a size of 64 bytes. The network does not include an impairment generator (which was required within a test set up to compare independent IPPM implementations, see [RFC6808]). An impairment generator as part of the test set up will have a positive effect on the measurements and the measurements with bigger packet size will also succeed at a temporal resolution above [us] level.

8. Acknowledgements

Joachim Mende, Marc Wieland, Ralf Widera and Jens Wyduba helped to implement and operate the LDP PMS in our research network. In memoriam of Holger Zarwel, who gave our project unconditional support.

9. IANA Considerations

This memo includes no request to IANA.

10. Security Considerations

A PMS monitoring packet should never leave the domain where it originated. It therefore should never use stale MPLS or IGP routing information. If the Label Switch Path is broken, a packet with the destination address 127.0.0.0/26 should not be routed, it should be discarded. The PMS must be configured with a measurement interval (or sum of all measurement stream intervals) that does not overload the network. Too many measurement streams with a big packet size could overload a link.

11. References

11.1. Normative References

[RFC2679] Almes, G., Kalidindi, S., and M. Zekauskas, "A One-way Delay Metric for IPPM", RFC 2679, DOI 10.17487/RFC2679, September 1999, <http://www.rfc-editor.org/info/rfc2679>.

[RFC4379] Kompella, K. and G. Swallow, "Detecting Multi-Protocol Label Switched (MPLS) Data Plane Failures", RFC 4379, DOI 10.17487/RFC4379, February 2006, <http://www.rfc-editor.org/info/rfc4379>.

[RFC6576] Geib, R., Ed., Morton, A., Fardid, R., and A. Steinmitz, "IP Performance Metrics (IPPM) Standard Advancement Testing", BCP 176, RFC 6576, DOI 10.17487/RFC6576, March 2012, <http://www.rfc-editor.org/info/rfc6576>.

[RFC6808] Ciavattone, L., Geib, R., Morton, A., and M. Wieser, "Test Plan and Results Supporting Advancement of RFC 2679 on the Standards Track", RFC 6808, DOI 10.17487/RFC6808, December 2012, <http://www.rfc-editor.org/info/rfc6808>.

11.2. Informative References

[BCP-TX] NANOG, "Best Practices for Determining Traffic Matrices in IP Networks V 4.0", 2008.

[I-D.ietf-spring-oam-usecase] Geib, R., Filsfils, C., Pignataro, C., and N. Kumar, "A Scalable and Topology-Aware MPLS Dataplane Monitoring System", draft-ietf-spring-oam-usecase-03 (work in progress), April 2016.

[LDP-TE] VDE-Verlag, "Traffic Matrices for MPLS Networks with LDP Traffic Statistics", 2004.

Appendix A. ADK2 Test Source Code

The following C++ source code is a modified version of the Code at [RFC6576]. This version allows to test two files containing values with the ADK2. It is not necessary that the values are sorted, because in the first step the values get sorted.

```
/*
Copyright (c) 2012 IETF Trust and the persons identified
as authors of the code. All rights reserved.

Redistribution and use in source and binary forms, with
or without modification, is permitted pursuant to, and subject
to the license terms contained in, the Simplified BSD License
set forth in Section 4.c of the IETF Trust's Legal Provisions
Relating to IETF Documents (http://trustee.ietf.org/license-info).
*/

/* Routines for computing the Anderson-Darling 2 sample
* test statistic.
*
* Implemented based on the description in
* "Anderson-Darling K Sample Test" Heckert, Alan and
* Filliben, James, editors, Dataplot Reference Manual,
* Chapter 15 Auxiliary, NIST, 2004.
* Official Reference by 2010
* Heckert, N. A. (2001). Dataplot website at the
* National Institute of Standards and Technology:
* http://www.itl.nist.gov/div898/software/dataplot.html/
* June 2001.
*/

// this code is a modified version of the code in RFC6576
```

```
// use '-std=c++11' for compiling

#include <iostream>
#include <fstream>
#include <vector>
#include <sstream>
#include <iterator>

#include <algorithm>


using namespace std;

/* This function reads the values and sorts this in an ascending
* order.
* The format is: one value per line followed by a line break.
* A blank line at the end of the file will crash the program.
*/
vector<double> read_file_sort (string filename) {
    vector<double> vec;
    // variable for one line of the file and the value
    string line;
    double tmp;

    ifstream file;
    file.open(filename, ios::in);
    if (!file) {
        cout << "Error in file " << filename << endl;
    }
    else {
        // read file in a vector
        while(!file.eof()) {
            getline (file, line);
            tmp = stod (line);
            vec.push_back(tmp);
        }
        // sort the vector ascending
        sort(vec.begin(), vec.end());
    }
    file.close();
    return vec;
}

 int main(int argn, char *argv[]) {

    if (argn != 1 && argn != 3) {
        cout << "wrong invocation" << endl;
        cout << "start with " << argv[0] << " file1 file2" << endl;
```

```
            cout << "start with " << argv[0] << " without parameter, if \
            the files are named file1.csv and file2.csv" << endl;
            return 1;
        }

        vector<double> vec1, vec2;
        double adk_result;
        static int k, val_st_z_samp1, val_st_z_samp2,
                   val_eq_z_samp1, val_eq_z_samp2,
                   j, n_total, n_sample1, n_sample2, L,
                   max_number_samples, line, maxnumber_z;
        static int column_1, column_2;
        static double adk, n_value, z, sum_adk_samp1,
                      sum_adk_samp2, z_aux;
        static double H_j, F1j, hj, F2j, denom_1_aux, denom_2_aux;
        static bool next_z_sample2, equal_z_both_samples;
        static int stop_loop1, stop_loop2, stop_loop3,old_eq_line2,
                   old_eq_line1;

        static double adk_criterium = 1.993;

        string filename1 = "file1.csv";
        string filename2 = "file2.csv";

        // if called with filenames
        if (argn == 3) {
            filename1 = argv[1];
            filename2 = argv[2];
        }

        // sort the two files i a vector
        vec1 = read_file_sort(filename1);
        vec2 = read_file_sort(filename2);

        k = 2;
        n_sample1 = vec1.size() - 1;
        n_sample2 = vec2.size() - 1;

        // -1 because vec[0] is a dummy value
        n_total = n_sample1 + n_sample2;

        /* value equal to the line with a value = zj in sample 1.
         * Here j=1, so the line is 1.
         */
        val_eq_z_samp1 = 1;

        /* value equal to the line with a value = zj in sample 2.
         * Here j=1, so the line is 1.
```

```
        */
       val_eq_z_samp2 = 1;

       /* value equal to the last line with a value < zj
        * in sample 1.  Here j=1, so the line is 0.
        */
       val_st_z_samp1 = 0;

       /* value equal to the last line with a value < zj
        * in sample 1.  Here j=1, so the line is 0.
        */
       val_st_z_samp2 = 0;

       sum_adk_samp1 = 0;
       sum_adk_samp2 = 0;
       j = 1;

       // as mentioned above, j=1
       equal_z_both_samples = false;

       next_z_sample2 = false;

       // assuming the next z to be of sample 1
       stop_loop1 = n_sample1 + 1;

       // + 1 because vec[0] is a dummy, see n_sample1 declaration
       stop_loop2 = n_sample2 + 1;
       stop_loop3 = n_total + 1;

       /* The required z values are calculated until all values
        * of both samples have been taken into account.  See the
        * lines above for the stoploop values.  Construct required
        * to avoid a mathematical operation in the while condition.
       */
       while (((stop_loop1 > val_eq_z_samp1)

              || (stop_loop2 > val_eq_z_samp2)) && stop_loop3 > j) {
           if (val_eq_z_samp1 < n_sample1+1) {
               /* here, a preliminary zj value is set.
                * See below how to calculate the actual zj.
                */
               z = vec1[val_eq_z_samp1];

               /* this while sequence calculates the number of values
                * equal to z.
                */
               while ((val_eq_z_samp1+1 < n_sample1)
                   && z == vec1[val_eq_z_samp1+1] ) {
```

```
                val_eq_z_samp1++;
                }
        }
        else {
        val_eq_z_samp1 = 0;
        val_st_z_samp1 = n_sample1;

        // this should be val_eq_z_samp1 - 1 = n_sample1
        }

        if (val_eq_z_samp2 < n_sample2+1) {
            z_aux = vec2[val_eq_z_samp2];

            /* this while sequence calculates the number of values
             * equal to z_aux
             */

            while ((val_eq_z_samp2+1 < n_sample2)
                    && z_aux == vec2[val_eq_z_samp2+1] ) {
                val_eq_z_samp2++;
            }

            /* the smaller of the two actual data values is picked
             * as the next zj.
             */

            if(z > z_aux) {
                z = z_aux;
                next_z_sample2 = true;
            }
            else {
                if (z == z_aux) {
                equal_z_both_samples = true;
                }

                /* This is the case if the last value of column1 is
                 * smaller than the remaining values of column2.
                 */
               if (val_eq_z_samp1 == 0) {
                z = z_aux;
                next_z_sample2 = true;
                }
            }
        }
        else {
            val_eq_z_samp2 = 0;
            val_st_z_samp2 = n_sample2;
```

```
            // this should be val_eq_z_samp2 - 1 = n_sample2
          }

           /* in the following, sum j = 1 to L is calculated for
            * sample 1 and sample 2.
            */
          if (equal_z_both_samples) {

              /* hj is the number of values in the combined sample
               * equal to zj
               */
              hj = val_eq_z_samp1 - val_st_z_samp1
                + val_eq_z_samp2 - val_st_z_samp2;

              /* H_j is the number of values in the combined sample
               * smaller than zj plus one half the number of
               * values in the combined sample equal to zj
               * (that's hj/2).
               */
              H_j = val_st_z_samp1 + val_st_z_samp2 + hj / 2;

              /* F1j is the number of values in the 1st sample
               * that are less than zj plus one half the number
               * of values in this sample that are equal to zj.
               */

              F1j = val_st_z_samp1 + (double)
                    (val_eq_z_samp1 - val_st_z_samp1) / 2;

              /* F2j is the number of values in the 1st sample
               * that are less than zj plus one half the number
               * of values in this sample that are equal to zj.
               */
              F2j = val_st_z_samp2 + (double)
                   (val_eq_z_samp2 - val_st_z_samp2) / 2;

              /* set the line of values equal to zj to the
               * actual line of the last value picked for zj.
               */
              val_st_z_samp1 = val_eq_z_samp1;

              /* Set the line of values equal to zj to the actual
               * line of the last value picked for zj of each
               * sample.  This is required as data smaller than zj
               * is accounted differently than values equal to zj.
               */
              val_st_z_samp2 = val_eq_z_samp2;
```

```
            /* next the lines of the next values z, i.e., zj+1
             * are addressed.
             */
            val_eq_z_samp1++;

            /* next the lines of the next values z, i.e.,
             * zj+1 are addressed
             */
            val_eq_z_samp2++;
        }
        else {

            /* the smaller z value was contained in sample 2;
             * hence, this value is the zj to base the following
             * calculations on.
             */
            if (next_z_sample2){
                /* hj is the number of values in the combined
                 * sample equal to zj; in this case, these are
                 * within sample 2 only.
                 */
                hj = val_eq_z_samp2 - val_st_z_samp2;

                /* H_j is the number of values in the combined sample
                 * smaller than zj plus one half the number of
                 * values in the combined sample equal to zj
                 * (that's hj/2).
                 */
                H_j = val_st_z_samp1 + val_st_z_samp2 + hj / 2;

              /* F1j is the number of values in the 1st sample that
             * are less than zj plus one half the number of values in
               * this sample that are equal to zj.
               * As val_eq_z_samp2 < val_eq_z_samp1, these are the
               * val_st_z_samp1 only.
               */
                F1j = val_st_z_samp1;

              /* F2j is the number of values in the 1st sample that
             * are less than zj plus one half the number of values in
             * this sample that are equal to zj.  The latter are from
               * sample 2 only in this case.
               */

                F2j = val_st_z_samp2 + (double)
                     (val_eq_z_samp2 - val_st_z_samp2) / 2;

             /* Set the line of values equal to zj to the actual line
```

```
                * of the last value picked for zj of sample 2 only in
                * this case.
                */
                val_st_z_samp2 = val_eq_z_samp2;

               /* next the line of the next value z, i.e., zj+1 is
                * addressed.  Here, only sample 2 must be addressed.
                */

                val_eq_z_samp2++;
                if (val_eq_z_samp1 == 0) {
                    val_eq_z_samp1 = stop_loop1;
                }
            }
           /* the smaller z value was contained in sample 2;
            * hence, this value is the zj to base the following
            * calculations on.
            */

            else {

                /* hj is the number of values in the combined
                 * sample equal to zj; in this case, these are
                 * within sample 1 only.
                 */
                hj = val_eq_z_samp1 - val_st_z_samp1;

                /* H_j is the number of values in the combined
                 * sample smaller than zj plus one half the number
                 * of values in the combined sample equal to zj
                 * (that's hj/2).
                 */

                H_j = val_st_z_samp1 + val_st_z_samp2 + hj / 2;

            /* F1j is the number of values in the 1st sample that
             * are less than zj plus; in this case, these are within
             * sample 1 only one half the number of values in this
             * sample that are equal to zj.  The latter are from
             * sample 1 only in this case.
             */

                F1j = val_st_z_samp1 + (double)
                    (val_eq_z_samp1 - val_st_z_samp1) / 2;

            /* F2j is the number of values in the 1st sample that
             * are less than zj plus one half the number of values
             * in this sample that are equal to zj.  As
```

```
                * val_eq_z_samp1 < val_eq_z_samp2, these are the
                * val_st_z_samp2 only.
                */

                   F2j = val_st_z_samp2;

               /* Set the line of values equal to zj to the actual line
                * of the last value picked for zj of sample 1 only in
                * this case.
                */

                   val_st_z_samp1 = val_eq_z_samp1;
                   /* next the line of the next value z, i.e., zj+1 is
                    * addressed.  Here, only sample 1 must be addressed.
                    */
                   val_eq_z_samp1++;

                   if (val_eq_z_samp2 == 0) {
                       val_eq_z_samp2 = stop_loop2;
                   }
               }
           }

           denom_1_aux = n_total * F1j - n_sample1 * H_j;
           denom_2_aux = n_total * F2j - n_sample2 * H_j;

           sum_adk_samp1 = sum_adk_samp1 + hj
                           * (denom_1_aux * denom_1_aux) /
                           (H_j * (n_total - H_j)
                           - n_total * hj / 4);
           sum_adk_samp2 = sum_adk_samp2 + hj
                           * (denom_2_aux * denom_2_aux) /
                           (H_j * (n_total - H_j)
                           - n_total * hj / 4);

           next_z_sample2 = false;
           equal_z_both_samples = false;

           /* index to count the z.  It is only required to prevent
            * the while slope to execute endless
            */
           j++;
       }

       // calculating the adk value is the final step.
       adk_result = (double) (n_total - 1) / (n_total
              * n_total * (k - 1))
               * (sum_adk_samp1 / n_sample1
```

```
                + sum_adk_samp2 / n_sample2);


        /* if(adk_result <= adk_criterium)
         * adk_2_sample test is passed
         */
        //return adk_result <= adk_criterium;
        cout << "Result: " << adk_result << endl;
    }
```

Authors' Addresses

Raik Leipnitz (editor)
Deutsche Telekom
Olgastr. 67
Ulm 89073
Germany

Email: r.leipnitz@telekom.de

Ruediger Geib
Deutsche Telekom
Heinrich Hertz Str. 3-7
Darmstadt 64295
Germany

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