HGI-GD030

HOME NETWORKS KPI EVALUATION FRAMEWORK
1 CONTENTS

2 Important notices, IPR statement, disclaimers and copyright ........................................ 4
2.1 About HGI .................................................................................................................. 4
2.2 This may not be the latest version of This HGI Document ......................................... 4
2.3 There is no warranty provided with This HGI Document .......................................... 4
2.4 Exclusion of Liability ................................................................................................. 4
2.5 This HGI Document is not binding on HGI nor its member companies ....................... 4
2.6 Intellectual Property Rights ...................................................................................... 4
2.7 Copyright Provisions ................................................................................................. 5
    2.7.1 Incorporating HGI Documents in whole or part within Documents Related to Commercial Tenders 5
    2.7.2 Copying This HGI Document in its entirety ....................................................... 5
2.8 HGI Membership ....................................................................................................... 5
3 Introduction .................................................................................................................. 7
4 Overview of the KPI’s ................................................................................................... 8
5 KPI services .................................................................................................................. 11
6 KPI Analysers ............................................................................................................... 12
7 Definition of a KPI ........................................................................................................ 15
8 KPI detailed definition .................................................................................................. 18
9 Best in home test practice ............................................................................................ 28
10 Executable Scripts for KPI assessment ........................................................................ 30
11 Appendix 1 – Service: Windows FullRate Script ......................................................... 31
    11.1 Windows script for HGI KPI Testing .................................................................... 31
    11.2 Required hardware and Initial configuration ....................................................... 31
    11.3 Script architecture ............................................................................................... 32
    Service : Manager code (Manager.vbs) .................................................................... 34
    Service: Client code (172.16.1.x_172.16.1.x.vbs) ..................................................... 35
12 Appendix 2 – Test Harness Scripts running under Ubuntu ........................................... 38
    12.1 Python Script Running Under Ubuntu-Linux for KPI Testing ............................. 38
12.2 Required Platform and Initial Configuration .......................................................... 38
12.3 Test Manager Script Architecture ........................................................................ 39
12.4 HGI_test_manager.py .......................................................................................... 42
12.5 MGEN Scripts ...................................................................................................... 50
12.5.1 Transmitter employed for KPI service “HDVoD” (aka. “HDFilm”): .................. 50
12.5.2 Transmitter employed for KPI service “HDLive”: .......................................... 50
12.5.3 Transmitter employed for KPI service “Headroom”: ..................................... 51
12.5.4 Transmitter employed for KPI service “FullLink_toSTB” (aka. “FullRate”): ........ 52
12.5.5 Transmitter employed for KPI service “FullLink_toTV2”: ............................... 52
12.5.6 Transmitter employed for KPI service “FullLink_toTV3”: ............................... 53
12.5.7 Receiver employed for KPI services “HDVoD” and “Headroom”: ................... 54
12.5.8 Receiver employed for KPI service “HDLive”: .............................................. 54
12.5.9 Receiver employed for KPI service “FullLink” (aka. “FullRate”): ..................... 54
12.6 mgen_stats.py ..................................................................................................... 55
13 Appendix 3 – Analyser: Throughput CCDF .............................................................. 60
14 Appendix 4 - KPI analysis example implementation .............................................. 62
15 Appendix 5 – Test Control Schema XSD ................................................................. 72
16 Appendix 6 – Analysis Specification Schema XSD ................................................. 82
17 Appendix 7 – XML Profile ...................................................................................... 88
2 IMPORTANT NOTICES, IPR STATEMENT, DISCLAIMERS AND COPYRIGHT

This chapter contains important information about HGI and this document (hereinafter ‘This HGI Document’).

2.1 ABOUT HGI

The Home Gateway Initiative (HGI) is a non-profit making organization which publishes guidelines, requirements documents, white papers, vision papers, test plans and other documents concerning broadband equipment and services which are deployed in the home.

2.2 THIS MAY NOT BE THE LATEST VERSION OF THIS HGI DOCUMENT

This HGI Document is the output of the Working Groups of the HGI and its members as of the date of publication. Readers of This HGI Document should be aware that it can be revised, edited or have its status changed according to the HGI working procedures.

2.3 THERE IS NO WARRANTY PROVIDED WITH THIS HGI DOCUMENT

The services, the content and the information in this HGI Document are provided on an "as is" basis. HGI, to the fullest extent permitted by law, disclaims all warranties, whether express, implied, statutory or otherwise, including but not limited to the implied warranties of merchantability, non-infringement of third parties rights and fitness for a particular purpose. HGI, its affiliates and licensors make no representations or warranties about the accuracy, completeness, security or timeliness of the content or information provided in the HGI Document. No information obtained via the HGI Document shall create any warranty not expressly stated by HGI in these terms and conditions.

2.4 EXCLUSION OF LIABILITY

Any person holding a copyright in This HGI Document, or any portion thereof, disclaims to the fullest extent permitted by law (a) any liability (including direct, indirect, special, or consequential damages under any legal theory) arising from or related to the use of or reliance upon This HGI Document; and (b) any obligation to update or correct this technical report.

2.5 THIS HGI DOCUMENT IS NOT BINDING ON HGI NOR ITS MEMBER COMPANIES

This HGI Document, though formally approved by the HGI member companies, is not binding in any way upon the HGI members.

2.6 INTELLECTUAL PROPERTY RIGHTS

Patents essential or potentially essential to the implementation of features described in This HGI Document may have been declared in conformance to the HGI IPR Policy and Statutes (available at the HGI website www.homegateway.org).
2.7 COPYRIGHT PROVISIONS

© 2013 HGI. This HGI Document is copyrighted by HGI, and all rights are reserved. The contents of This HGI Document are protected by the copyrights of HGI or the copyrights of third parties that are used by agreement. Trademarks and copyrights mentioned in This HGI Document are the property of their respective owners. The content of This HGI Document may only be reproduced, distributed, modified, framed, cached, adapted or linked to, or made available in any form by any photographic, electronic, digital, mechanical, photostat, microfilm, xerography or other means, or incorporated into or used in any information storage and retrieval system, electronic or mechanical, with the prior written permission of HGI or the applicable third party copyright owner. Such written permission is not however required under the conditions specified in Section 2.7.1 and Section 2.7.2:

2.7.1 INCORPORATING HGI DOCUMENTS IN WHOLE OR PART WITHIN DOCUMENTS RELATED TO COMMERCIAL TENDERS

Any or all section(s) of HGI Documents may be incorporated into Commercial Tenders (RFP, RFT, RFQ, ITT, etc.) by HGI and non-HGI members under the following conditions:

(a) The HGI Requirements numbers, where applicable, must not be changed from those within the HGI Documents.

(b) A prominent acknowledgement of the HGI must be provided within the Commercial document identifying any and all HGI Documents referenced, and giving the web address of the HGI.

(c) The Commercial Tender must identify which of its section(s) include material taken from HGI Documents and must identify each HGI Document used, and the relevant HGI Section Numbers.

(d) The Commercial Tender must refer to the copyright provisions of HGI Documents and must state that the sections taken from HGI Documents are subject to copyright by HGI and/or applicable third parties.

2.7.2 COPYING THIS HGI DOCUMENT IN ITS ENTIRETY

This HGI Document may be electronically copied, reproduced, distributed, linked to, or made available in any form by any photographic, electronic, digital, mechanical, photostat, microfilm, xerography or other means, or incorporated into or used in any information storage and retrieval system, electronic or mechanical, but only in its original, unaltered PDF format, and with its original HGI title and file name unaltered. It may not be modified without the advanced written permission of the HGI.

2.8 HGI MEMBERSHIP

The HGI membership list as of the date of the formal review of this document is: Advanced Digital Broadcast S.A, Arcadyan, ARM, Bouygues Telecom, Broadcom, BT, Cisco, Deutsche Telekom, devolo AG, Dialog Semiconductor B.V., Digital TV LabsDigital TV Labs, DSP Group, eflow Inc., EnOcean Alliance, Fastweb SpA, Hitachi, Huawei, Ikanos, Imagination Technologies, Intel, KPN, Lantiq, LG, Microsemi, MStar, NEC Corporation, Netgear, NTT, Oki, Orange, Portugal Telecom
3 INTRODUCTION

The Key Performance metrics Indicators (KPI’s) are intended to give a ‘real world’ view of the likely performance of various home networking technologies. The KPI’s are primarily aimed at giving an indication of the in-home connectivity and performance. This provides a more useful alternative to the typical ‘on the box’ theoretical maximum bandwidths often stated for home networking kit and can be used for meaningful communication with vendors. Although there are many variables involved with home testing, these high level metrics are useful indicators of likely performance taking into account domestic attenuation and interference. The KPI’s are technology agnostic and are useful for comparing the results of different in-home solutions.

A KPI is a combination of prescribed tests and subsequent analysis. This document outlines the defined tests and analysis. Both the tests and the analysis are formally defined by XML schemas with the details of the analysis processing presented as pseudo code. The currently defined schema tests and analysis objects are used to generate a ‘default’ set of XML HGI KPIs, these can be found in the latest revision of HGI02174. These range from simple point to point tests with throughput averages for analysis, to, concurrent streams and more complex analysis. Whilst the HGI ‘default’ KPIs aim at providing a balanced view of home IPTV connectivity, other KPIs are possible for other types of assessment.

The KPI’s do not assume any particular test environment. The HGI have made available the XML schemas definitions and XML implementations via the DMS. The appendices of this document also contain a minimal KPI test platform expressed through script-controlled open source components.

This document gives an overview of the KPI’s, and provides a description of the schemas, pseudo code and working basic test scripts. The set of KPI’s is not fixed and is open to additional schema definitions for test and analysis as agreed by the HGI TWG and published as future XML and XSD documents.
4 OVERVIEW OF THE KPI’S

The KPI’s have resulted from previous work in the HGI where in-home networking connectivity and performance was assessed. In order to assess in-home networking connectivity and performance, HGI members conducted in homes assessments in a number of countries, using retail power-line and 5 GHz Wi-Fi units. The assessments showed that connectivity can be highly variable as seen in Error! Reference source not found..

![5 GHz performance over 24 hours](image)

**Figure 1. Throughput over time**

Significantly variable performance is common for both power-line and Wi-Fi systems; albeit for different reasons but the root causes are attenuation coupled with noise or interference. For power-line, significant interference can arise from plug top power supplies and low energy lights, and many other devices. For 5 GHz Wi-Fi, the movement of people, pets or doors can cause significant variations in performance. The KPI’s offer a range of useful analysis methods to characterize the connectivity encountered during testing and offer an easy way to understand the overall performance of the home network. One key principle of the analysis is to extract the test paths that have had no connectivity whatsoever and then perform analysis over room paths that have registered some throughput. Metrics on both these aspects of performance are useful and complementary.
The time spent testing in real homes needs to be minimized (to avoid unduly inconveniencing the occupants). A way of testing multiple services, paths and rooms concurrently was therefore needed. To do this, the testing supports hopping from one traffic type on one path to another, reasonably rapidly (typically every 15 seconds). Longer term testing comprises of a set of throughput ‘snapshots’ built up for each room, path and service. There is clearly a trade-off between obtaining a broad set of room path/service types and having higher granularity records over a more limited number of paths and services. The time taken to cycle round all the test and traffic types is typically ~135 seconds. Error! Reference source not found. shows the average throughput of one room path and traffic type using this arrangement.
Figure 2. However, as mentioned previously, the actual set of nodes, streams and analysis are determined by what the user of the KPI’s is actually trying to determine.

**Concurrent streaming**

*HD Film*

*HD Live (Multicast)*

*Headroom (over and above fixed)*

**Full Rate assessment**

*TCP maximum rate*

The KPI model is defined in 2 XSD schemas, one specifying the entities used in testing and the other for analysis. Conforming XML files provide the actual implementation and define the number of test nodes, traffic types and analysis, as desired by the user. These files can then be used by platform specific script, test harnesses or manual execution to apply the defined test and analysis.

Although any combination of streaming types, rates and test paths can be used,

Figure 2 shows the HGI default network for in-home assessment. The definition for the streaming types (services) is to be found in Section 5.
The KPI test nodes are NOT part of the customer’s own home network. However, to be representative of the real performance the KPI endpoints are placed in the appropriate locations in a test home. The ‘Hub’ is placed as close as possible (together with the technology under test) to the Home Gateway. Similarly, the STB, TV2 and TV3 are placed where the users main, second and third TV’s are (or would be) located. This is important as it produces meaningful analysis, such as evaluating the in home performance of the path between the gateway and main TV.

5 KPI SERVICES

KPI services are defined within the schema as distinct entities and essentially describe the test traffic being streamed. Currently the schema defines the services outlined in

Figure 2 as HD Film, HD Live, Headroom and TCP maximum rate (full rate assessment). The definition of new services is regulated by the HGI. At least one service must be selected as part of the KPI. The simplest available KPI service would be a single path running a TCP maximum rate/Full Link service and would equate to a continual set of ‘ftp’ type assessments. The throughput rates connected with a service are applicable to the application layer, rather than layer 1.

The defined service types are:

- **HD Film**: a 6Mbps fixed rate concurrent UDP stream from the hub to each end point. Although TCP may more typically be used for this type of streaming, we have found that ‘weak’ paths will be progressively TCP throttled by good links. For this reason, we use UDP to assess the maximum available throughput.

- **HD Live**: a 12Mbps fixed rate multicast UDP stream also sent concurrently from the hub to each end point. Some non wired network technologies exhibit poor performance when transmitting multicast. This is usually because they have to either serialize the stream to each end point, or revert to the rate of the ‘weakest’ end point.

- **Headroom**: A maximum rate concurrent UDP stream from the hub to each end point. This service enables a view to be gained of the maximum concurrent throughput when not constrained to a fixed rate. It also indicates the ‘headroom’ over and above the HD Film/HD Live rates. The concurrent streams assume identical DSCP/QoS markings. Some future KPI may include explicit QoS packet markings for advanced testing.

- **FullRate**: A single maximum rate TCP stream. This cycles around each available room path giving a view of the available maximum throughput per path.

Although these are simple traffic types it is intended that more complex streaming functionality can be expressed. Such functionality might include a play-out buffer and stream retry modelling; for this reason the term ‘service’ has been adopted.
At this point, some consideration of the analysis that is applied to these services is appropriate. Room paths that have no or perfect connectivity will always show ‘all bad/all good’ regardless of the analysis method. Other appropriate analysis has to be chosen to fully understand the nature of the paths that have partial connectivity. As an example, Error! Reference source not found. is not readily understood in terms of its ability to deliver a service, as it varies over time. An analyser giving an overall average throughput does not give a meaningful representation of the connectivity seen. For this reason more than one analyser can be associated (via the XML implementation file) to provide various views of the connectivity.

6 KPI Analysers

Currently, there are 3 analysers described in the schema. Aside from the analyser functionality, the schema also specifies how the records generated by the testing are to be collated, processed and what values are given to the user. The schema for the services defines the values that are logged from the service and these define the input for the analysis. Currently, all the services map down to TCP or UDP logs, which all analysers can process. However, if other service are added that don’t use these base recording formats then each current analyser will have to be amended. The currently defined analysers are:

- **Basic Analysis**: As the name implies, this is used to provide a high level over-view of the connectivity. For this analysis, all data records of the defined input aggregation are processed as one total collection to provide the output of:

  Percent Zero Paths: The percentage of paths that had no connectivity throughout the test period. These records are not included in any further processing.

  Zero records: The percentage of the records from paths that exhibited some connectivity where the throughput was zero.

  Average: The cumulative throughput of all the records divided by the number of total records. This includes any Zero records present (but does not include the records from the Zero Paths)

  Deviation:

  \[
  \sigma = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (x_i - \mu)^2}, \quad \text{where} \quad \mu = \frac{1}{N} \sum_{i=1}^{N} x_i.
  \]

  RunTime: The lapsed time from the earliest to the last test record. It reflects the lapsed time spent in each home rather than an summation of the individual path runtimes. It also takes into account crossing the midnight boundary from one day to the next and it is expressed in minutes.
• **Fixed Rate Analysis**: This compares a fixed rate throughput against the incoming record stream and gives a measure of the ‘viewing time between errors’. This analysis uses the delivery bandwidth as a threshold for defining the presence of an error (e.g. a 6 Mbps stream suffers and error if the recorded bandwidth is below 6 Mbps). This analysis is possibly more pessimistic than the real world customer experience as many solutions employ packet loss concealment or retransmission schemes. However it does identify inherently robust technologies. This scope of this also includes the ability to target a particular time span (i.e. 17:00 to midnight as peak viewing).

The outputs are:

- Zero Paths: The percent of paths (from total paths) that had no connectivity throughout the test period.
- RunTime: The lapsed time from the earliest to the latest test record. The run time is not cumulative over all the paths. It also takes into account crossing the midnight boundary from one day to the next and it expressed in minutes.
- ViewtimeOfUpTo30mins:
- ViewtimeOfUpTo1Hour:
- ViewtimeOfUpTo2Hours:
- ViewtimeOfUpTo4Hours:
- All of these quantities express the maximum time without the bandwidth dropping below the service level. In essence this describes the ‘maximum viewable time, without a ‘glitch’. These are expressed as the percent of room paths with a viewable time for each bucket. A value of 100% is expected from the summation of the percentages of Zero, 30min, 1 hour, 2 hours and 4 hours.

• **Throughput Distribution**: The graphical representation of the distribution of the recorded throughputs is a useful and easy way to assimilate performance indicator. We define the “Throughput Distribution KPI” in terms of a complementary cumulative distribution function (CCDF). The CCDF $C(x)$ of a random variable $X$ is defined as

An implementation of the above calculations in form of a Python script is given in Appendix 3.

• **Packet Error Distribution**: This distribution gives analysis of the dropped packets as shown in the example below,

<table>
<thead>
<tr>
<th>Record Time</th>
<th>#packets</th>
<th># pkts</th>
<th>#Instances</th>
<th>Avg. rate</th>
</tr>
</thead>
</table>

© HGI 2015 – All rights reserved
we define Packet Error Time (PET) as the time interval (e.g. in seconds) between two successive error events. An error event is defined as a measurement record with the number of dropped packets $> 0$. In the above example, error events occur for Records 3 and 8. In this case, a sample of PET is obtained from the time stamp of record 4 until the time stamp of record 8, as the time difference:

$$PET = Time(Record \ 8) - Time(Record \ 4)$$

$$= 13:47:43 - 13:35:15 \ (hh:mm:ss)$$

$$= 00:12:28$$

$$= 748 \text{ seconds (12.47 minutes)}$$

We define the “PET Distribution KPI” again as a CCDF. The PET Distribution can be derived with the same approach as described above for the Rate Distribution by just replacing the vector $R$ of rate samples with the vector of PET samples. The scope of the records included in this analysis follows is determined by the user at analysis time. This may be just for one path in a given home, or include all paths in all homes.
If an entire measurement file does not include any error events, the time difference between the first and last measurement record serves as the single PET sample representing this particular file in the PET vector.

If a measurement file includes a single error event only, then two PET samples can be obtained, by counting the time difference between the first record and the error event, and the time difference between the record following the error event and the last record of the file.

The PET distribution can be measured within a specific time-interval of the day, and therefore matched to typical user behaviour, e.g. the main TV viewing hours between 17:00 and 24:00 hours. In this case the data records should be filtered such that only records in that time window apply to the PET analysis.

7 DEFINITION OF A KPI

The discussion so far has outlined the various constituents of a KPI. At the highest level, a KPI consists of a service linked with one or more analysers. These are fully defined in XSD schemas and composed into a KPI instance through an XML file which links a defined service with a defined analysis set. The XML file can also contain multiple KPIs, forming a sequence of service tests and analysis (known as a KPI test set). The HGI defines the following KPI’s test set.
Figure 3 HGI KPI Test set

Within the HGI KPI test set, the four individual KPI’s can clearly be identified (by the service name) of Fullrate, Headroom, HD Film and HD Live with their associated analysers. As can be seen, the analysers are not tied to a specific service. The KPI test set is wrapped with other meta-data to produce a KPI profile. The HGI KPI profile is aimed at providing a balanced range of metrics for assessing in-home IPTV connectivity. Although all these are specified in the XML DMS implementation file, other implementations targeting a different purpose will use a different set. For instance, for a simple view of in-home connectivity to compare technologies, a Full Rate service with basic analysis would be sufficient.

All of these are explored in more detail, in the next section where a walk through the schemas highlights key areas of a KPI. The remainder of this section describes the sequence in processing a KPI.
Figure 4 Utilising a KPI

Figure 4 shows the progression of a KPI. Once configured, the nominated test harness executes the sequence of tests and logs the results with the specified output into log files (normally a file per trialist/technology/service/path). The term trialist is used to identify the test location or home and normally takes the form of a postcode or ZIP location. Once all home testing is complete, the entire dataset is then processed by the analysers specified by the KPI analysis configuration. The test harness and analysers may be fully automated or rely entirely on manual configuration. An analyser is responsible for applying the desired level of aggregation of the recorded dataset. The aggregation provides a view based on ‘all records’ or a certain sub-set. This is discussed further in the next section, with a more detailed description of the schemas.
8 KPI DETAILED DEFINITION

This section provides a walkthrough of the KPI definition, highlighting the key parameters. A KPI draws on entities from two XSD schemas, one for the test control and the other for analysis specification. Overall the schemas and HGI profile, expressed in XML are not easy to study in text form, so this section uses a graphical representation to highlight various key areas. The schemas and KPI specification are in HGI02174

Figure 5. KPI Profile - First level
Figure 5 shows top level entries under the KpiProfile root. The details of the technologies being tested are string comments, to aid readability. The NodeInfo entries are expanded to show the defined nodes their address field used for IP configuration. The KPITest set containing all service and analysis details is discussed next.

![Diagram](image)

**Figure 6. KPI Profile – The KPI**

The second level shown in Figure 6 expands the KpiTestSet into one or more KPI’s. Each KPI entity encapsulates the service and analysis descriptions together with path and error control. These are further expanded.
The service schema expands into the 4 other (current) services. Although only the ‘FullRate’ service is expanded in Figure 7, all the other services follow the same pattern. The protocol field is currently either TCP or UDP but could change if new transmission types are introduced. For instance, a service that is just concerned with IPTV retransmissions might require a protocol type of ‘RET’. Even though RET would actually map down onto TCP/UDP transmissions it would have its own service type to described the logged values (e.g. number of retransmissions per second, peak retransmission etc.). The rate can either be set to a maximum or a fixed level that reflects the content type it represents. The ‘FullRate’ service is shown, requiring the ‘Max’ value in the XML implementation file, whereas ‘HD Film’ would require a ‘Fixed’ value of 6 Mbps. Where fixed rates are used, the transmitter would use a given bandwidth (e.g. content encoding rate) with an additional margin to deal with OS overheads etc. The ‘LoggedInfo’ is based on the Protocol field. The Full rate service used TCP and so in Figure 7, the logged information is just a timestamp and throughput as appropriate.
The desired paths are selected from the 'Paths' schema entity. When implemented in XML one or more of the possible paths can be selected. The selected paths are subject to the path control parameter, which determines if the test is executed sequentially across all the paths (essentially a test per path). If the test is not sequential, concurrent streaming is specified to instruct all transmitters to concurrently send to the designated receivers within one test period.
In general, the HGI KPIProfile will be used by some form of test harness that will implement some form of signaling with the end node test points. If at any point the communication fails, invalid results will be obtained. This might arise from a test not starting, stopping or being unable to obtain the result (from the Rx). Assuming, the test harness is not at fault, the communication failure is due to poor network connectivity that needs to be logged. Control of how this is logged is left to the implementer of the XML file. The default behavior is to ‘InvalidateResultsForAllNodes’, for a serial path test, this just results in 0 Mbps being logged for the point to point test. For concurrent path testing, if any path fails even though others succeed, all paths will record 0 Mbps. This may not be the desired behavior and so there are other options to log the failure which impacts the analysis in a different way.

The review of the KPI continues with a discussion of the analysis schema.
The analysis section of the profile contains one or more analysers, as detailed previously. Only one analyser can be assigned to a service at any time. However, various views of the performance can be gained by changing the analyser in the XML implementation file and re-executing the analysis. The schema provides a structured way to specify the input, output and scope (the level of record aggregation). Figure 10, shows the BasicAnalysis entity, with an expanded input section. The inputs, outputs and scope are applied to all analysers. The input section provides filter specifications to apply on the data. For instance specifying a time filter to only include the records from 17:00 to 23:59 may be useful, rather than including the performance in the early hours of the morning; again this is down to the implementer of the XML to consider. Although the analysis rate parameter is ignored for BasicAnalysis, all analysers follow the input, output and scope pattern.

Figure 10 Analysis Schema - top level
The output shown in Figure 11, is specific for the BasicAnalysis analyser. By default, the processing of the analyser does not include paths that are fully ‘dead’. These are expressed as a separate measure as a percentage of the total number of tested paths. BasicAnalysis amalgamates the records of all the room paths (under the direction of the scoping) and that are not fully dead. It may encounter records that recorded 0 Mbps, these are accumulated and presented as a percent of the whole record set.
Figure 12 Analysis schema – scoping

The scoping of the input log files is shown in Figure 12. The scoping essentially provides the number and granularity of the output reporting. For instance, under Path, ‘AllPaths’ indicates that an output is required having combined the record sets for all paths in a home. Trialist control is very similar producing an output by combining all the records from all trialists, a specific trialist or each trialist. Again for technology, a similar arrangement is used. If the XML has a scope of ‘ForEachTechnology’, ‘ForAllTrialists’ and ‘AllPaths’ for the BaiscAnalyser, the resultant outputs are expressed in Table 1.

<table>
<thead>
<tr>
<th></th>
<th>Average</th>
<th>StdDev</th>
<th>PercentZero</th>
<th>Percent Zero Paths</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tech A</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>Tech B</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>Tech C</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
</tr>
</tbody>
</table>

Table 1 Top level scope example for BasicAnalyser
If ‘ForEachTechnology’, ‘ForEachTrialist’ and ‘ForEachPath’ was configured, the following table shows the resultant outputs:

<table>
<thead>
<tr>
<th>Tech</th>
<th>Trialist</th>
<th>Average</th>
<th>StdDev</th>
<th>PercentZero</th>
<th>Percent Zero Paths</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tech A</td>
<td>TrialistA</td>
<td>HubStb</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>Tech A</td>
<td>TrialistA</td>
<td>HubTv2</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>Tech A</td>
<td>TrialistA</td>
<td>.....</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
</tr>
</tbody>
</table>

Table 2 Detailed scope example for Basic Analyser

The difference between the examples clearly shows the effect of changing the scoping, with the most detailed level shown in Table 2, with an analysis output per tested path.
9  BEST IN HOME TEST PRACTICE

This section provides a few examples to ensure the results from home testing are reliable as possible.

For longer term testing where different technologies are being sequentially tested over a week, trying to ensure the same domestic cycle gives the best results. For instance, identifying repeatable domestic cycles (i.e. weekdays only) gives the best opportunity for a similar test environment.

Longer term testing is used where the performance of a given IPTV service is being assessed rather than the relative performance of a given set of technologies. To compare one technology against another, a useful strategy is to use short term testing. Constructing an XML implementation file that just specifies a single path using a ‘FullRate’ service with 10 second snap shots gives a high granularity view of the performance. Using an hourly test cycle (with this configuration) a reasonable number of records are available for analysis, whilst changes in the operation environment are minimised.

The usage of ‘pass-through’ sockets, when provided, on power-line device’s requires some consideration. Using the pass-through socket for other customer CPE (customer premises equipment), perhaps via a dis-board, ensures any mains borne interference is filtered out before it degrades the power-line signal. Although this gives the best possible performance from the power-line devices, it may not reflect the real world. Many customers are un-aware of the advantage the pass-through gives or will not consider using it as it extends the distance out from the wall. This may cause difficulties as the inserted plug in the pass-through socket may foul the available gap between furniture and the wall. When using the KPIs for power-line testing the usage of pass-through sockets should be specified.

The usage of ‘dis-boards’ is also an area that needs to be specified for power-line. Anti-surge boards should always be avoided as the surge filters cause a significant (artificial) attenuation of the signal. Chained distribution boards also need to be considered, although the practice is considered unsafe, it’s popular! However, a distribution board chain hosting a large amount of customer CPE greatly increases the chance of introducing interference. This may have a significant impact on a power-line device in a dis-board chain if it’s also receiving a heavily attenuated signal from a distant transmitter.

For Wi-Fi, there is little to consider aside from the physical placement of the test devices. At 5 GHz very small changes in placement can have a sizeable impact on the performance and this need to be minimised. For units having internal PCB antennas, placing units in the same orientation is important. However, where a trialist would just ‘drop’ a unit behind a piece of furniture this is not always possible.

Paths that have no connectivity have to be noted and investigated. If the non-connectivity is due to issues with the failure in delivering management messages, rather than the actual connectivity of the path under test, the results should be removed. This may occur if the manager is remote from the end points of the path under test. Paths results that reflect real non connectivity should be included. Testing on paths where non connectivity is present should always be continued through the test cycle time, as they can sometimes ‘come alive’ is the conditions change.
When analysing the results, some pre-inspection should be entertained. It is not uncommon for trialists to unplug equipment at night, forgetting that testing is underway. In addition, trialist may change over test devices one at a time without stopping the tests until they reach the manager. This behaviour causes zero throughputs to be logged where human intervention causes the problem! Clearly the results need to be cleaned of failures due to these and similar errors.

Realising these issues allows the tester to assess what customer environments are acceptable without introducing a skew on the results through exceptional and erroneous test conditions.
10 EXEcutable Scripts for KPI Assessment

The following sections/appendices contain examples of working implementations of a test harness for KPI assessment.

In appendix 1, the test harness is provided by script based windows code to run iPerf on remote machines. Pseudo code is provided for the simple basic analysis and python scripting for CCDF throughput analysis. This demonstrates the KPIs are not tied to any specific test harness. However, executing a KPI in this fashion requires a large amount of manual intervention and using a configurable test harness in more appropriate.

Appendix 2 presents a test harness based on Python scripts which run on PCs under Ubuntu-Linux OS. Instead of using iPerf, the streaming traffic transmitters and receivers employ the open source software toolset MGEN.
11 Appendix 1 – Service: Windows Full Rate Script

11.1 Windows Script for HGI KPI Testing

In any real world assessment, a KPI profile must be translated and applied to a ‘test harness’ and analysis tool set. Both or either of these could be proprietary, however, an entry level assessment can be realised with freely available tools. Usage of ‘Iperf’ (http://en.wikipedia.org/wiki/Iperf) combined with some scripting, can be used as a test harness. The analysis of the ‘basic’ KPI profile can be achieved by taking the output of this process and feeding it into a spread-sheet. This section describes the end to end process in more detail, with the scripts that drive this process shown in appendix 1, 2 and 3.

These scripts use the Microsoft environment to prove the concept. However, other platforms and scripting languages can of course be used. The scripts in the appendices have been used and verified as fit for purpose. However, the HGI have no liability in the matter and the scripts provided are used at your own risk.

11.2 Required Hardware and Initial Configuration

The set up in this section details the resources required and a basic level of configuration for the Windows 7® environment. The ‘basic profile’ requires 4 test nodes and a further machine is required as a manager to co-ordinate the testing between the nodes. As the manager and test nodes are required to execute processes on the remote nodes the appropriate execution permissions and an open access policy is required on all machines.

Assuming that 5 Windows 7 machines are available, they should be configured to use a suitable subnet (i.e. 172.16.0.0/16). As specified in the KPI basic profile, the 4 test nodes should be configured with 172.16.1.1/2/3/4 with the manager using an address in the subnet (e.g. 172.16.1.10).

Iperf, (Windows binaries) should be downloaded and placed into a machine wide accessible directory i.e. c:\Windows. In addition, PSTools (http://technet.microsoft.com/en-us/sysinternals/bb897553.aspx) should also be downloaded and extracted to the same location. This tool-set allows remote execution of Iperf and script files.

To enable remote execution on these machines they need to be configured with a common user account (with administrative privileges). In the network section of control panel the Microsoft client needs to be operating (this is a default configuration). In the network and internet control panel, the advanced sharing settings from the network and sharing center should be selected and ensure that ‘password protected sharing’ is operative, together with ‘public folder sharing’. The final step is to enter the LocalAccountTokenFilterPolicy (DWORD) key into the registry location
HKLM\Software\Microsoft\Windows\CurrentVersion\Policy\System, configuration set to 1. The firewall should also be disabled. It should be noted that with these settings the machines have a low level of security. A final check to ensure that access is possible from the manager is to type at a DOS prompt ‘net use \172.16.1.1\Admin$ /user:username password’. If this command fails then the security is probably not configured correctly. The username and password should match those used in the script files in the appendices (manager.vbs and client.vbs)

11.3 SCRIPT ARCHITECTURE

The general pattern is for the manager to select which end points are to be tested. The manager then remotely executes a batch file (.bat) on the node that is going to be the transmitter. The transmitter’s batch file then executes a local VBScript file. This file opens an Iperf session to the receiver, which opens up a Iperf ‘server’ window. The transmitter script then runs the Iperf test and logs the result (or failure) to the local hard disk. The manager waits for a certain period of time before selecting the next pair of nodes for testing. The manager has no knowledge of the test failure or success.

The sequence used by the manager is hard wired into the manager.vbs script (listed in appendix 1). The script should run on the manager without any changes. However, if a different address range is used this can readily be changed in the first 4 lines of the script and reflected in the script file names. Changing the names of the nodes (the next 4 lines) is not advised, as the names are used to execute the correctly name batch files on the client machines. The script contains the set paths to be used, and is configured for the 6 paths between the 4 nodes. However, this can be altered to reflect adjustments in test node numbers.
The figure above shows the locations of all the scripts used. The manager has the controlling Manager.vbs and this calls the .bat files on the remote machines to progress the tests. Each of the remote .bat files invokes a local .vbs script. This then executes a remote Test.bat file. An example of this is the first test of the basic profile, running between the Hub and Stb nodes. The manager script remotely runs the HubStb.bat on 172.16.1.1, which in turn executes the local 172.16.1.1_172.16.1.2.vbs. This script then executes the Test.bat file on 172.16.1.2 which runs the Iperf server. The 172.16.1.1_172.16.1.2.vbs script runs the test to the remote Iperf server and logs the result in a c:\results\172.16.1.1_172.16.1.2.csv file. The file is appended with every test result for this path. After allowing sufficient time for the test to run the manager then repeats the process, but using the HubTv2.bat file on 172.16.1.1. It continues in this fashion executing all the bat files on the Hub, Stb and Tv2 machines before starting the cycle again.

As can be seen there are 6 client vbs files that represent each test path. 172.16.1.4 only acts as an Iperf server to host a remote end point, with the Test.bat file.
Service : Manager code (Manager.vbs)

```
userId = "uid"
pass = "pass"

HubAddr = "172.16.1.1"
StbAddr = "172.16.1.2"
Tv2Addr = "172.16.1.3"
Tv3Addr = "172.16.1.4"

HubName = "Hub"
StbName = "Stb"
Tv2Name = "Tv2"
Tv3Name = "Tv3"

dim clientAddresses(4)
cclientAddresses(0) = HubAddr
cclientAddresses(1) = StbAddr
cclientAddresses(2) = Tv2Addr
cclientAddresses(3) = Tv3Addr

dim clientNames(4)
cclientNames(0) = HubName
cclientNames(1) = StbName
cclientNames(2) = Tv2Name
cclientNames(3) = Tv3Name

path1 = HubName + " " + StbName
path2 = HubName + " " + Tv2Name
path3 = HubName + " " + Tv3Name
path4 = StbName + " " + Tv2Name
path5 = StbName + " " + Tv3Name
path6 = Tv2Name + " " + Tv3Name
Dim Allpaths(6)
Allpaths(0) = path1
Allpaths(1) = path2
Allpaths(2) = path3
Allpaths(3) = path4
Allpaths(4) = path5
Allpaths(5) = path6

endIndex = uBound(Allpaths) - 1
while true
    For i = 0 to endIndex
        testpath = Allpaths(i)
        endPoints = Split(testpath)
        vbsName = endPoints(0)
        remoteName = endPoints(1)

        ' find the addresses of the vbsName
        vbsAddress = ""
        For vbsIndex = 0 to uBound(clientNames)
            ' code
```

str = clientNames(vbsIndex)
If str = vbsName Then
    vbsAddress = clientAddresses(vbsIndex)
end if
next

' Check we have the security and acces available
WScript.StdOut.WriteLine("Checking access to " & vbsName)
' attach to default admin share
checkRemotestring = "net use \" & vbsAddress & "\Admin$ /user:" & userId & " " & pass
' look at the response
Set objShell = CreateObject("WScript.Shell")
Set exec = objShell.Exec(checkRemotestring)
accessCheck = false
While Not exec.StdOut.EOF
    line = exec.StdOut.ReadLine
    If InStr(1, line, "successful") > 0 then
        accessCheck = true
    end if
wend
if not accessCheck then
    WScript.StdOut.WriteLine("Unable to reach or missing permissions on" & vbsAddress)
    WScript.Quit
else
    WScript.StdOut.WriteLine("Access ok")
    ' ditch the connection
    ditchRemotestring = "net use \" & vbsAddress & "\Admin$ /delete"
    objShell.Exec(ditchRemotestring)
end if

' ok we have access, now run the test script on the vbsAddress machine
remoteFileName = "c:\" & vbsName & remoteName & ".bat"
remoteScriptString = "psexec -i -u " & userId & " -p " & pass & "]\" & vbsAddress & " " & remoteFileName
WScript.StdOut.WriteLine("Running " & remoteFileName)
Set exec = objShell.Exec(remoteScriptString)
While Not exec.StdOut.EOF
    line = exec.StdOut.ReadLine
    WScript.StdOut.WriteLine(line)
wend
WScript.Sleep(20000)
next
wend

**Service: Client code (172.16.1.x_172.16.1.x.vbs)**

remoteEnd = "172.16.1.x"
localEnd = "172.16.1.x"
userId = "uid"
pass = "pwd"
'Check we have the security and access available
checkremoteEndString = "net use \" & remoteEnd & "\Admin$ /user:" & userId & " " & pass
WScript.StdOut.WriteLine("Checking access to " & checkremoteEndString)
Set objShell = CreateObject("WScript.Shell")
Set exec = objShell.Exec(checkremoteEndString)

' look at output for success
remoteEndCheck = false
While Not exec.StdOut.AtEndOfStream
    line = exec.StdOut.ReadLine
    WScript.StdOut.Write(line)
    If InStr(1, line, "successful") > 0 then
        remoteEndCheck = true
    end if
wend
objShell = nil
if remoteEndCheck = false then
    WScript.StdOut.Write("Unable to access or dont have logon permissions for " & server)
    WScript.Quit
end if

WScript.StdOut.WriteLine("Bringing up remote Iperf script" & remoteEnd)

' bring up the server on the remote machine
' the bat file the other end does not block and we don't get a return code to work with
' so we wait and hope it fires up in x seconds
remoteEndString = "psexec -s -i -u " & userId & " -p " & pass & " \" & remoteEnd & " c:\test.bat"
Set objShell = CreateObject("WScript.Shell")
Set exec = objShell.Exec(remoteEndString)
WScript.Sleep(5000)

WScript.StdOut.WriteLine("trying TcpMax to " & remoteEnd)
striPerf = "iperf.exe"
Set objShell = CreateObject("WScript.Shell")
strCommand = striPerf & " -c " & remoteEnd
Set exec = objShell.Exec(strCommand)

' look at what strings we got back for the iperf server
' was it good?
throughPut = "0"
While Not exec.StdOut.AtEndOfStream
    line = exec.StdOut.ReadLine
    compactedString = Replace(line," ","")
mBytes = InStr(line,"MBytes")
mBits = InStr(line,"Mbits")
if(mBytes > 0) and (mBits > 0) then
  throughput = Mid(line, mBytes + 6, mBits - mBytes - 6)
  WScript.StdOut.WriteLine("Throughput " & throughput & "Mbit/sec")
end if
Wend
if throughput = "0" then
  WScript.StdOut.Write("test failed")
else
  WScript.StdOut.Write("Test OK")
end if

' log whatever result we got by creating the file if it does not exist then push in the result
logFileName = "c:\CHTestResults\" & locaEnd & "\" & remoteEnd & ".cvs"
set logObj = CreateObject("Scripting.FileSystemObject")
if Not logObj.FileExists(logFileName) Then
  set logFile = logObj.CreateTextFile(logFileName,true, false)
  logFile.WriteLine("Time, Mbits")
logFile.Close
end if
' 8 means append an existing file
set logFile = logObj.OpenTextFile(logFileName,8, False, 0)
logFile.WriteLine(Now & "," & throughput)
logFile.Close

Test.bat
' all the same with
Start iperf -s

Client.bat (HubStb.bat shown)
' all the same format
Start cscript c:\172.16.1.1_172.16.1.2.vbs
12 Appendix 2 – Test Harness Scripts Running under Ubuntu

12.1 Python Script Running Under Ubuntu-Linux for KPI Testing

This Appendix presents another example of an implementation of a test harness. The test manager described in the following is implemented using Python 2.7.3 scripting language which uses system commands executing under a Linux OS.

The actual KPI services are implemented using the “Multi-Generator” (MGEN, version 5.02) open source software toolset developed by the Naval Research Laboratory (NRL) Protocol Engineering Advanced Networking (PROTEAN) group (http://www.nrl.navy.mil/itd/ncs/products/mgen).

The scripts shown further below have been tested under Ubuntu 12.04 LTS (http://www.ubuntu.com/download/desktop). Again, the scripts shall serve as guideline for implementation. For the scripts provided here, the HGI takes no liability. The scripts may be used at your own risk.

12.2 Required Platform and Initial Configuration

The set up in this section details the resources required and a basic level of configuration for the Ubuntu 12.04 LTS operating system (http://www.ubuntu.com/download/desktop).

As in the example described in Appendix 1, the ‘basic profile’ again requires 4 test nodes and an additional machine as a manager to co-ordinate the testing between the nodes. As the manager and test nodes are required to execute processes on the remote nodes the appropriate execution permissions and an open access policy is required on all machines. In the script HGI_test_manager.py shown below, the test manager accesses the test nodes as root user using SSL. In order to enable access without any manual interaction, it is assumed that the appropriate root user credentials are preconfigured on the manager and on all test nodes.

While the test manager should run e.g. on any Laptop PC, the test nodes do not need to be equipped with a display. Therefore, any inexpensive mini-PCs may be employed as HW platform for the test nodes (e.g. ASUS EeeBox PC’s), capable to operate under a thin Linux OS.

Assuming that 5 Ubuntu machines are available, they should be configured to use a suitable subnet (e.g. 172.16.0.0/16, or 192.168.194.0/24, as used in the following example).

The MGEN Linux executable (downloaded from http://downloads.pf.itd.nrl.navy.mil/mgen/) must be placed in a suitable directory (denoted “/home/user/mgen” in the script) on each of the 4 test nodes.

To enable remote control of the test nodes by the manager (e.g. execution of MGEN), all 5 machines require SSL to be installed and the SSL daemon process to be activated at startup.
12.3 Test Manager Script Architecture

The Python script HGI_test_manager.py shown below which implements the test manager, should be to most extent self-explained to programmers familiar with Python and Linux. This script essentially performs the following actions:

- Setting default directory and file names employed on the manager node and on the test nodes.
- Setting the IP addresses of the test nodes (these must also be configured as fixed IP addresses on each node).
- In the example HGI_test_manager.py script shown below, the IP addresses are set in a structure called “ip_configuration” as follows:

  ```python
  ip_configuration = {
    'TV2' : { 'ip': '192.168.194.10' },
    'TV3' : { 'ip': '192.168.194.11' },
    'HUB' : { 'ip': '192.168.194.12' },
    'STB' : { 'ip': '192.168.194.13' },
  }
  ```

  The manager node may use any IP address of the same subnet 192.168.194.0/24, e.g. 192.168.194.1.
- Defining the sequence of KPI services that are executed one after another and repeatedly within a loop.
- Defining the configuration settings of the test nodes for each specific KPI service. This information is configured in the structure denoted “serv_list”.
- Setting the names of the csv test data output files (denoted “csv_files” in the script).
- Constructing the ssh commands to be executed on the test nodes. The required ssh commands are
  - mgenrx_cmd (defined in utility function “start_mgenrx()”)
  - mgentx_cmd (defined as inline string)
  - mgen_stats_cmd (defined in utility function “start_mgen_stats()”)
- Writing the test results into output files of type “.csv”.


Handling of some diagnostics information. Any detected errors, e.g. failure of execution of an ssh command, is aimed to be captured as an error code in a variable denoted “exceptions”. Note that this implemented in a rudimentary way only in the example script HGI_test_manager.py.

Setting of some arguments that can also be provided in the command line as defined in function main(). Enter the command ./HGI_test_manager –h in a terminal window to obtain help information on applicable command line arguments.

Starting the main performance testing loop defined in function KPI_test_loop() which is called from function main().

MGEN as configured as either streaming traffic transmitter or receiver by means of script files. See the “MGEN User’s and Reference Guide Version 5.0” for a detailed description of the script syntax. The MGEN script files used in the respective mgentx_cmd and mgenrx_cmd commands by the test manager are listed in the section “MGEN scripts” further below.

During each MGEN test run (set to 10 seconds duration in the MGEN transmitter script files), every receiver generates a very detailed log file including one line of text for each received UDP or TCP packet. This file includes e.g. stream identifiers, time stamps when a packet was transmitted and when it was received which allow to analyze jitter and delay, packet sequence numbers that allow to detect missing or out-of-sequence packets. The present HGI_test_manager.py script writes this temporary output file denoted output2.tmp into a ramdisk folder /tmp/ram. On each receiver test node, this file is processed locally with a Python script mgen_stats.py to obtain some meaningful compact statistics. The mgen_stats.py script generates a single line of output for each individual 10-seconds MGEN test run.

Each line includes the following data:

<table>
<thead>
<tr>
<th>ref_time</th>
<th>ave_rate</th>
<th>ave_rate_ins</th>
<th>txtime</th>
<th>rx_pktcnt</th>
<th>oos_pktcnt</th>
<th>ins_pktcnt</th>
<th>ino_pktcnt</th>
<th>lost_pktcnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>hh:mm:ss</td>
<td>kbps</td>
<td>kbps</td>
<td>seconds</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **ref_time**: a reference time in hours:minutes:seconds format indicating the start of the test run
- **ave_rate**: average rate in kilo-bits per second of received data, disregarding packet order (i.e. receiver handles reordering without buffer constraints)
- **ave_rate_ins**: average rate in kilo-bits per second of received data, counting packets received in-sequence only (i.e. no reordering buffer at the receiver)
txtime: time difference in seconds between the first and last received packet

rx_pktcnt: total number of packets received in the test run

oos_pktcnt: number of packets received out-of-sequence

ins_pktcnt: number of packets received in-sequence

ino_pktcnt: number of packets received in correct order but with lost packets in-between

lost_pktcnt: number of packets lost (i.e. dropped packets)

For each test run, the above described output of mgen_stats.py is collected by the manager node from each involved receiver node and written into the final output files in csv format.

It should be noted that this implementation of a test harness make it extremely convenient to change the investigated KPI service scenarios. Any such changes can be completely covered by changing (i.e. adding, deleting, or replacing) information contained in the Python structures serv_list and serv_sequence. If new type of streaming traffic generators or receivers are required, these can easily be defined in form of new MGEN scripts.

In an advanced implementation of the test harness, these data structures in the Python code could be auto-generated from the XML files that define the desired KPI Profile, as outlined in clauses 5 and 6 of this document. However, this feature is not implemented in the example script shown below.
12.4 HGI_TEST_MANAGER.PY

The following Python script implements the test manager as described above.
#!/usr/bin/python
###
# This is a Python version 2.7.3 script implementing the "Test Manager" of HGI-compliant Test Harness
# as described in HGI GWD030 (HGI01901R04)
#
# NOTES:
# - The open source tool "mgen" is used to implement transmitters and receivers
# - Statistics evaluation of results with another Python script denoted "mgen_stats.py"
#
###

# Requirements:
# SSH keys: disable ssh password login for remote host
# IP address allocation: static as defined in ip_configuration
# Programs: mgen, mgen_stats.py: on all test nodes
# gnuplot: on the test manager
# mgen input scripts: as defined in serv_list
#
###

import getopt, sys, os, re, time
import subprocess as sub
import signal
from datetime import date

# default input variables
verbose = False
random_dir = "~/tmp/ram/"

mgen_dir = "~/home/user/mgen/
# path to mgen directory, this directory exist on all
test_nodes
output_folder = mgen_dir + "measurements/"
# default prefix of the folder where the test results
# are stored.
# This is overwritten if the folder is defined in the
# command line
ssh_user = "root"

measurement_cycles = 200

# normal configuration: NOTE: IP addresses apply to test management interfaces, not necessarily the
tested i/f used in MGEN scripts
ip_configuration = {
    'TV2' : { 'ip': "192.168.194.10"},
    'TV3' : { 'ip': "192.168.194.11"},
    'HUB' : { 'ip': "192.168.194.12"},
    'STB' : { 'ip': "192.168.194.13"},
}

# Sequence of service scenarios (= "KPI services") to be executed on the test nodes:
serv_sequence = ['HDVo', 'HDLive', 'Headroom', 'FullLink_HUBtoSTB', 'FullLink_HUBtoTV2', 'FullLink_HUBtoTV3', 'FullLink_STBtoTV2', 'FullLink_STBtoTV3', 'FullLink_TV2toTV3']

global scenario

# List defining the configuration of test nodes for each service scenario:

serv_list = {
    'HDVo': {'mgenx_machine': 'HUB', 'mgenr_machine': 'STB', 'TV2', 'TV3'},
    'HDLive': {'mgenx_machine': 'HUB', 'mgenr_machine': 'STB', 'TV2', 'TV3'},
    'Headroom': {'mgenx_machine': 'HUB', 'mgenr_machine': 'STB', 'TV2', 'TV3'},
    'FullLink_HUBtoSTB': {'mgenx_machine': 'HUB', 'mgenr_machine': 'STB'},
    'FullLink_HUBtoTV2': {'mgenx_machine': 'HUB', 'mgenr_machine': 'TV2'},
    'FullLink_HUBtoTV3': {'mgenx_machine': 'HUB', 'mgenr_machine': 'TV3'},
    'FullLink_STBtoTV2': {'mgenx_machine': 'STB', 'mgenr_machine': 'TV2'},
    'FullLink_STBtoTV3': {'mgenx_machine': 'STB', 'mgenr_machine': 'TV3'},
    'FullLink_TV2toTV3': {'mgenx_machine': 'TV2', 'mgenr_machine': 'TV3'},
}

### names of csv output files to be generated for each service scenario
csv_files = {
    'HDVo': ['HUB2STB', 'HUB2TV2', 'HUB2TV3'],
    'HDLive': ['HUB2STB', 'HUB2TV2', 'HUB2TV3'],
    'Headroom': ['HUB2STB', 'HUB2TV2', 'HUB2TV3'],
    'FullLink_HUBtoSTB': ['HUBtoSTB'],
    'FullLink_HUBtoTV2': ['HUBtoTV2'],
    'FullLink_HUBtoTV3': ['HUBtoTV3'],
    'FullLink_STBtoTV2': ['STBtoTV2'],
    'FullLink_STBtoTV3': ['STBtoTV3'],
    'FullLink_TV2toTV3': ['TV2toTV3'],
}

### list of test nodes that may be operated as receivers
active_rx_nodes = ['STB', 'TV2', 'TV3']

# Function to start mgen receivers
def start_mgenr(mgenx_machine, mgen_dir, mgenxin_fn, mgenx_error_file):
    kw_mgenr = {
        'bufsize': 8,  
        'executable': None,
        'stdin': sub.PIPE,
        'stdout': sub.PIPE,  # mgenr_output_file,  # mgenr_output_file does not seem to be used even if
        'preexec_fn': None,
        'close_fds': False,
        'shell': False,
        'cwd': None,
        'env': None,
        'universal_newlines': False,
        'startfile': None,
        'creationflags': 0,
    }

    mgenx_cmd = "ssh " + ssh_user + '@' + ip_configuration[mgenx_machine]["ip"] + " \
    " + mgen_dir + "mgen input " + mgen_dir + mgenxin_fn + " output " + mgenx_error_file
    "output2.tmp"

    if verbose:
        print "mgen receiver execution " + mgenx_cmd

    try:
        p = subprocess.Popen(mgenx_cmd.split(' '), **kw_mgenr)
err = 0
except:
    err = 1
    print "ssh command or mgen receiver didn't work"
else:
    err = 0
    mgenrx_error_file.close()
    mgenrx_error_filename = mgen_dir + "mgenrx_error.txt"

return err

# Compute mgen measurement summary, write result into "mgen_raw_summary.tmp" and extract relevant information
def start_mgen_stats(mgenrx_node, remote_mgendir, local_mgendir, ramdisk_dir, result_csv_file, ref_time, exception):
    mgen_stats_cmd = "ssh "+ssh_user+"@" + ip_configuration[mgenrx_node]["ip"] + \\nlocal_mgendir + "mgen_raw_summary.tmp"

    if verbose:
        print "mgen_stats.py execution " + mgen_stats_cmd

    try:
        p = os.system(mgen_stats_cmd)
        exception = exception + 100
        print "mgen_stats.py command didn't work, node: " + mgenrx_node
    except:
        exception = exception + 200

    try:
        mgen_stats_output_filename = stats_output_folder + "mgen_raw_summary.tmp"
        mgen_stats_output_file = open(mgen_stats_output_filename, 'r')
    except:
        exception = exception + 200
    else:
        mgen_stats_line = mgen_stats_output_file.readlines()
        mgen_stats_output_file.close()

        mgen_stats_line_split = mgen_stats_line[0].split('>') # get effective transmission time

        if len(mgen_stats_line_split) == 5:
            print "Incorrect format of mgen_raw_summary.tmp"
            exception = exception + 200 # setting rate to zero in case of an exception, in case of an incomplete mgen_summary.tmp file

        mgen_stats_line_split = mgen_stats_line[1].split('>') # get rate

        if len(mgen_stats_line_split) == 4:
            print "Incorrect format of mgen_raw_summary.tmp"
            exception = exception + 200 # setting rate to zero in case of an exception, in case of an incomplete mgen_summary.tmp file

        else:
            print "rx_pktcnt", rx_pktcnt, "oos_pktcnt", oos_pktcnt, "lost_packets", lost_pktcnt,
            mgen_stats_line_split = mgen_stats_line[2].split('>') # get packet counts

            if len(mgen_stats_line_split) == 10:
                rx_pktcnt = mgen_stats_line_split[1].strip(" ")
                oos_pktcnt = mgen_stats_line_split[3].strip(" ")
                ins_pktcnt = mgen_stats_line_split[5].strip(" ")
                oso_pktcnt = mgen_stats_line_split[7].strip(" ")
                ins_pktcnt = mgen_stats_line_split[8].strip(" ")
                lost_pktcnt = mgen_stats_line_split[9].strip(" ")

            else:
                print "Incorrect format of mgen_raw_summary.tmp"
                exception = exception + 200 # setting rate to zero in case of an exception, in case of an incomplete mgen_summary.tmp file
mgen_stats_line_split = mgen_stats_line[3].split('>')  # get histogram
if len(mgen_stats_line_split) == 2:
    if mgen_stats_line_split[8] == 'packet loss histogram':
        loss_histogram = mgen_stats_line_split[1].strip('
').strip(' ').strip('[')
    else:
        print "incorrect format of mgen_raw_summary.tmp"
else:
    print "an incomplete mgen_summary.tmp file"
    exception = exception + 600  # setting rate to zero in case of an exception, in case of
    # an incomplete mgen_summary.tmp file

if exception != 0: ave_rate = str(exception/1000.)  # setting rate to "zero" in case of an
# exception, but with "encoded" error code
joined_line = ref_time + " + ave_rate + "," + ave_rate_ins + " + ttime + " + rx_pktcnt \
+ ":" + oos_pktcnt + ":" + ins_pktcnt + ":" + ino_pktcnt + " +"
lost_pktcnt + "\n"
print "Joined line: " + joined_line
result_csv_file.write(joined_line)
return exception

# main KPI test processing loop
def KPI_test_loop():
    # setup directories and csv files
    csv_file_folder = []
    for ksc in range(8, len(serv_sequence)):
        scenario = serv_sequence[ksc]  # ['HDVoD', 'HDLive', 'Headroom', 'FullLink...']
        scenario_split = scenario.split(' ')
        print "scenario_split: " + scenario_split + 
        csv_file_folder.append(stats_output_folder + scenario_split[0] + 
        "\n" + os.path.exists(csv_file_folder[ksc]))
        print 'csv file folder: ' + csv_file_folder[ksc]
        num_receivers = len(serv_list[scenario]['mgenrx_machine'])
        for nrx in range(8, num_receivers):
            csv_output_file = open(csv_file_folder[ksc] + csv_files[scenario][nrx] + 
            ",", 'a')
            csv_output_file.write(ref_time, ave_rate (kbps), ave_rate_ins (kbps), ttime (s),
            rx_pktcnt, oos_pktcnt, ins_pktcnt, ino_pktcnt, lost_pktcnt 
            csv_output_file.close()

    # measurement cycle loop
    for i in range(8, measurement_cycles):
        print "measurement cycle: " + str(i)

    # cycling through the HGi service scenarios as defined in serv_sequence
    for ksc in range(8, len(serv_sequence)):
        scenario = serv_sequence[ksc]  # ['HDVoD', 'HDLive', 'Headroom', 'FullLink...']
        # start mgen receiver(s)
        num_receivers = len(serv_list[scenario]['mgenrx_machine'])
        exceptions = []
        for nrx in range(8, num_receivers):
            exceptions.append(8)
        mgenrx_input_filename = serv_list[scenario]['mgenrx_input_filename'][0]  # same file
        mgenrx_machine = serv_list[scenario]['mgenrx_machine'][nrx]
        mgenrx_error_filename = mgen_dir + "mgenrx_error.txt"
        mgenrx_error_file = open(mgenrx_error_filename, 'w')
        print "mgenrx_machine: " + mgenrx_machine
exceptions[nrx] = start_mgenrx(mgenrx_machine, mgen_dir, ramdisk_dir,
  mgenrx_input_filename, mgenrx_error_file);

print "exceptions after start_mgenrx: " + str(exceptions)

time.sleep(1)  # some waiting time in seconds, to enable TCP connection establishment

# start mgen transmitter(s)
num_transmitters = len(serv_list[scenario]['mgentx_machine'])
for ntx in range(0, num_transmitters):
    mgentx_input_filename = serv_list[scenario]['mgentx_input_filename'][ntx]
    mgentx_machine = serv_list[scenario]['mgentx_machine'][ntx]
    mgentx_cmd = "ssh "+ssh_user+"@" + ip_configuration[mgentx_machine]['ip'] + \
    " + mgen_dir + "+mgen_input " + mgen_dir + mgentx_input_filename
    if verbose:
        print "mgen transmitter execution " + mgentx_cmd

test_ref_time = time.strftime("%H:%M:%S", time.gmtime())
try:
    os.system(mgentx_cmd)
    err = 0
except OSError:
    print "mgen transmitter did not start"
    err = 10

# kill mgen rx processes on all receiving machines that have been started and get results
for nrx in range(0, num_receivers):
    mgenrx_machine = serv_list[scenario]['mgenrx_machine'][nrx]
    if exceptions[nrx] == 0:
        os.system("ssh "+ssh_user+"@" + ip_configuration[mgenrx_machine]['ip'] + "killall mgen"
    # update exception code with result from mgentx
    exceptions[nrx] = exceptions[nrx] + err  # !!!! works only for single transmitter !!!!!!!!
    # !!!! as association between rx and tx node not taken into account
    # get results from receiving nodes after mgen has finished and write output file(s)
    # now different result file for each receiver and each service
scenario !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    csv_output_file = open(csv_file_folder[ksc] + csv_file[scenario][nrx] + ".csv", 'a')
    err_mgen_stats = start_mgen_stats(mgenrx_machine, mgen_dir, stats_output_folder, ramdisk_dir, csv_output_file, test_ref_time, exceptions[nrx])
    csv_output_file.close()
    exceptions[nrx] = exceptions[nrx] + err_mgen_stats
    print "exceptions after start_mgen_stats: " + str(exceptions)

print "exceptions at end of this test run: " + str(exceptions)
time.sleep(1)  # to make sure that there are no packets with large delay still on the way

# main routine: processing input arguments, preparing test manager, starting KPI test processing loop
def main():
    try:
        opts, args = getopt.getopt(sys.argv[1:], "hf:c:v")
    except getopt.GetoptError, err:
        # print help information and exit:
        print "error code: " + str(err)
        usage()
        sys.exit(2)
    for o, a in opts:
        if o == "-v":
            global verbose
            verbose = True
        elif o in ("-h"):
usage()
    sys.exit()

        else o in ('-c'):
            global measurement_cycles
            measurement_cycles = a
        elif o in ('-f'):
            global output_folder
            output_folder = str(a) + "/"
            if not os.path.exists(output_folder):
                print "no such folder ..." + a + " make dir ...
                os.makedirs(output_folder)
            else:
                assert False, "unhandled option"

        signal.signal(signal.SIGINT, signal_handler)

        # prepare ramdisk on all potential rx nodes used in the test (don't care if already existent)
        num_receivers = len(active_rx_nodes)
        for nrx in range(0, num_receivers):
            mgenrx_node = active_rx_nodes[nrx]
            print "create ramdisk on rnxnode: " + mgemrx_node
            os.system("ssh "+ mgemrx_node + " @" + ip_configuration[mgemrx_node]["ip"] + " mkdir -p " + ramdisk_dir"
            os.system("ssh "+ mgemrx_node + " @" + ip_configuration[mgemrx_node]["ip"] + " mount -t tmpfs -o
            size=512M ttmpfs " + ramdisk_dir"

        now = date.today()
        print 'today date: ' + now.strftime("%Y-%m-%d")

        global stats_output_folder # on local machine (test manager)

        stats_output_folder = output_folder + now.strftime("%Y-%m-%d") + "-" + time.strftime("%H-%M-%S",
        time.gmtime()) + "/

        if not os.path.exists(stats_output_folder):
            print "no such folder ..." + stats_output_folder + " make dir ...
            os.makedirs(stats_output_folder)

        print 'Output Folder: ' + stats_output_folder

        KPI_test_loop()

        # Handling abort of the test manager with Ctrl+C
        def signal_handler(signal, frame):
            print 'You pressed Ctrl+C ...'

            num_receivers = len(serv_list[scenario]['mgemrx_machine'])
            for nrx in range(0, num_receivers):
                mgemrx_machine = serv_list[scenario]['mgemrx_machine'][nrx]
                os.system("ssh "+ mgemrx_machine + " @" + ip_configuration[mgemrx_machine]["ip"] + " killall mgem")

            num_transmitters = len(serv_list[scenario]['mgemtx_machine'])
            for ntx in range(0, num_transmitters):
                mgemtx_machine = serv_list[scenario]['mgemtx_machine'][ntx]
                os.system("ssh "+ mgemtx_machine + " @" + ip_configuration[mgemtx_machine]["ip"] + " killall mgem")
            sys.exit(0)

        # output of some command line help text
        def usage():
            print "This script creates evaluation data with mgem"
            print "-v verbose screen output flag (currently not used)"
            print "-c number of loop cycles (note: each cycle takes about 15 seconds per serv_sequence"
            print "-f base folder for storage of output files in csv format"
            print "example usage: ./HGI_test_manager.py -c 200 -f/home/user/mgi_measurements/"

            if name == "__main__":
                main()
12.5 MGEN SCRIPTS

The following MGEN script files need to be available in the directory /home/user/mgen of the respective transmitter and receiver nodes.

12.5.1 TRANSMITTER EMPLOYED FOR KPI SERVICE “HDVoD” (AKA. “HDFILM”):

Transmitter is started on the Hub node (192.168.194.12).

It generates 3 concurrent unicast UDP streams to STB, TV2 and TV3 nodes for a duration of 10 seconds. Each stream has a rate of 4000 packets/s with a payload packet size of 1472 bytes/packet (i.e. the nominal streaming data rate is 510 x 1472 x 8 = 6,005760 Mbps).

```
# MGEN script  "UDP_3unicast6_tx.mgn"
# 510 x 1472 x 8 = 6,005760 Mbps
#
# transmit three concurrent UDP unicast flows

0.0 ON 1 UDP SRC 5001 DST 192.168.194.10/5001 PERIODIC [510 1472]
0.0 ON 2 UDP SRC 5002 DST 192.168.194.11/5001 PERIODIC [510 1472]
0.0 ON 3 UDP SRC 5003 DST 192.168.194.13/5001 PERIODIC [510 1472]
10.0 OFF 1
10.0 OFF 2
10.0 OFF 3
```

12.5.2 TRANSMITTER EMPLOYED FOR KPI SERVICE “HDLive”:
The transmitter is started on the Hub node (192.168.194.12). It generates a single UDP multicast stream to multicast IP address 224.100.0.0 at a rate of 12.01 Mbps for a duration of 10 seconds.

```
# MGEN script "UDP_1multicast12_tx.mgn"
# 1020*14728*8 =: 12,011520  Mbps
# transmits a single UDP multicast flow of 10 seconds duration
0.0 ON 1 UDP SRC 5001 DST 224.100.0.0/5001 PERIODIC [1020 1472]
10.0 OFF 1
```

**12.5.3 Transmitter employed for KPI service “Headroom”:**
The transmitter is started on the Hub node (192.168.194.12). It generates 3 concurrent unicast UDP streams to STB, TV2 and TV3 nodes for a duration of 10 seconds. Each stream is set to maximum expected rate of 94.2 Mbps. Note that this rate would need to be increased if any of the tested transmission links may be capable to carry a throughput higher than 94.2 Mbps.

```
# MGEN script "UDP_3unicast95_tx.mgn"
# 8000 x 1472 x 8 = 94,208000 Mbps
# transmit three concurrent UDP unicast flows
0.0 ON 1 UDP SRC 5001 DST 192.168.194.10/5001 PERIODIC [8000 1472]
0.0 ON 2 UDP SRC 5002 DST 192.168.194.11/5001 PERIODIC [8000 1472]
```
12.5.4 **Transmitter employed for KPI service “FullLink_toSTB” (aka. “FullRate”):**
Depending on the path to be tested (cf. Figure 2 in clause 2), the transmitter is started on the Hub node (192.168.194.12) for testing the shown path 1 (HubtoSTB). It generates a single TCP unicast stream to be received by the STB under IP address 192.168.194.13 at a potential maximum rate of 93.44 Mbps for a duration of 10 seconds.

```
# MGEN input script TCPmax_toSTB.mgn
#
# nominal maximum rate 8000 x 1460 x 8 = 93,44 Mbps
#
TXBUFFER 65536
# Note that the target node must be listening to the destination tcp address
0.0 ON 1 TCP SRC 9000 DST 192.168.194.13/9000 PERIODIC [8000 1460]
10 OFF 1
```

12.5.5 **Transmitter employed for KPI service “FullLink_toTV2”:**
Depending on the path to be tested (cf. Figure 2 in clause 2), the transmitter is started on the Hub node or on the STB node for testing the shown path 2 (HubtoTV2) or path 4 (STBtoTV2), respectively. It generates a single TCP unicast stream to be received by the TV2 node under IP address 192.168.194.10 at a potential maximum rate of 93.44 Mbps for a duration of 10 seconds.
# MGEN input script TCPmax_toTV2.mgn

# max rate 8000 x 1460 x 8 = 93.44 Mbps

TXBUFFER 65536

# Note that the target node must be listening to the destination tcp address
0.0 ON 1 TCP SRC 9000 DST 192.168.194.10/9000 PERIODIC [8000 1460]
10 OFF 1

### 12.5.6 TRANSMITTER EMPLOYED FOR KPI SERVICE “FULLLINK_TOTV3”:

Depending on the path to be tested (cf. Figure 2 in clause 2), the transmitter is started on the Hub node, the STB node or the TV2 node for testing the shown path 3 (HubtoTV3), path 5 (STBtoTV3), or path 6 (TV2toTV3) respectively. It generates a single TCP unicast stream to be received by the TV3 node under IP address 192.168.194.11 at a potential maximum rate of 93.44 Mbps for a duration of 10 seconds.

# MGEN input script TCPmax_toTV3.mgn

# max rate 8000 x 1460 x 8 = 93.44 Mbps

TXBUFFER 65536

# Note that the target node must be listening to the destination tcp address
0.0 ON 1 TCP SRC 9000 DST 192.168.194.11/9000 PERIODIC [8000 1460]
10 OFF 1
12.5.7 **Receiver employed for KPI services “HDVoD” and “Headroom”:**
MGEN is started using the same receiver script file on the STB (192.168.194.13), TV2 (192.168.194.10) and TV3 (192.168.194.11) nodes.

```bash
RXBUFFER 100000
# Listen to UDP socket 5001 starting at 0 seconds into test
0.0 LISTEN UDP 5001
```

12.5.8 **Receiver employed for KPI service “HDLive”:**
MGEN is started using the same receiver script file on the STB (192.168.194.13), TV2 (192.168.194.10) and TV3 (192.168.194.11) nodes.

```bash
RXBUFFER 100000
# Listen to UDP socket 5001 starting at 0 seconds into test
0.0 LISTEN UDP 5001
# Join multicast stream at 0 seconds into test
0.0 Join 224.100.0.0
```

12.5.9 **Receiver employed for KPI service “FullLink” (aka. “FullRate”):**
MGEN is started using the same receiver script file on any of the receiving nodes, i.e. for path 1 on STB (192.168.194.13), for paths 2 and 4 on TV2 (192.168.194.10) and for paths 3, 5 and 6 on TV3 (192.168.194.11).

```
#################################################
# MGEN script   TCP_port9000_rx.txt
#################################################
RXBUFFER 65536
# Listen to TCP socket 9000 starting at 0 seconds into test
0.0 LISTEN TCP 9000

12.6 MGEN_STATS.PY
The following Python script processes the MGEN receiver output file to obtain the most meaningful statistical parameters, as described above, which are stored by the test manager in the csv output files.
#!/usr/bin/python
# Obtains rate, loss, latency... statistics from mgen output file
# Import modules
import getopt, sys, os, re, time
import subprocess as sub
import signal
from datetime import date

verbose = False
mgen_output_filename = "~/home/wgranzow/mgen/output2.txt"

global bytecount
bytecount = 0

global tctime

tctime = 0.0

rx_pktcnt = 0  # received packets
ins_pktcnt = 0  # in-sequence packets
ino_pktcnt = 0  # in-order packets
lost_pktcnt = 0  # out-of-sequence packets
oolo_pktcnt = 0  # out-of-order packets
packet_loss_hist
packet_loss_hist = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]  # array with 21 entries for  # packet loss histogram

ave_rate_kbps
ave_rate_kbps = 0.0

def usage():
    print "This script calculates rate, loss, latency statistics from an mgen logging data file"
    print "It assumes that the data file is located in the specified directory"  
    print "You can also specify a different directory using the -d option"
    print "Example usage: ./mgen_stats.py -v -f /home/wgranzow/mgen/mgen_output.txt"

# Analysis of mgen output file
def analyze(prev_line_split, curr_line_split, tctime, max_seqnum, bytecount, rx_pktcnt, ins_pktcnt, /
            ino_pktcnt, lost_pktcnt, ooolo_pktcnt, packet_loss_hist):

# NOTE: In this version, bit rate is calculated for 2 scenarios
# a) Every received packet is counted, independent of sequence number (corresponding to
#    infinite reordering buffer in the application)
# b) Every out-of-order received packet is discarded, only in-sequence packets are counted
#    (corresponding to no reordering buffer at all)

# Packet size in bytes (previously sent packet)
pktsize = int(prev_line_split[0].strip("size="))
# print "pktsize: ", pktsize

# Total number of (all) packets/bytes received without sequence check
bytecount = bytecount * pktsize
# print "bytecount: ", bytecount

# Arrival time previous packet = ["hh", "mm", "ss.ssssss"]
prev_pkttime = prev_line_split[0].split(":")

# Arrival time current packet = ["hh", "mm", "ss.ssssss"]
curr_pkttime = curr_line_split[0].split(":")
# calculation of pkt interarrival time = itime
prev_hh = int(prev_pkttime[0])
prev_mm = int(prev_pkttime[1])
prev_ss = float(prev_pkttime[2])

curr_hh = int(curr_pkttime[0])
curr_mm = int(curr_pkttime[1])
curr_ss = float(curr_pkttime[2])

prev_time = 3600.* prev_hh + 60.* prev_mm + prev_ss
curr_time = 3600.* curr_hh + 60.* curr_mm + curr_ss

itime_secs = curr_time - prev_time
if itime_secs < 0:
    itime_secs = itime_secs + 3600.*24

prev_mm = int(prev_pkttime[1])
prev_ss = float(prev_pkttime[2])
curr_mm = int(curr_pkttime[1])
curr_ss = float(curr_pkttime[2])

# accumulated total txtime in secs, applies also to rate calculation for packets received in-sequence
txtime = txtime + itime_secs

ave_rate_kbps = 0.008* float(bytecount)/txtime

# sequence numbers seqnum current and previous packets
prev_seqnum = int(prev_line_split[4].strip(‘seq’))
curr_seqnum = int(curr_line_split[4].strip(‘seq’))

# missing packets:
# delta_seqnum = curr_seqnum - prev_seqnum - 1  !!!! CHECK CHANGE!!!
delta_seqnum = curr_seqnum - max_seqnum - 1

# in case of roll-over of sequence numbers, i.e. prev_seqnum > curr_seqnum, no count as packet loss
# note that this can apparently only happen when the next iteration is started too early, such
# that the last packets from the previous round are still received after start of the mgmt receiver
if delta_seqnum == 0:  # new packet received in sequence
    max_seqnum = curr_seqnum
    ins_pktcnt = ins_pktcnt + 1
elif delta_seqnum < 0:  # new packet received out-of-sequence, to be discarded
    oos_pktcnt = oos_pktcnt + 1  # this is number of packets which are discarded at the receiver
    # (not counted into average rate)
    else:
        lost_pktcnt = lost_pktcnt + 1
        lost_seqnum = lost_seqnum + 1
        packet_loss_hist[lost_pktcnt] = packet_loss_hist[lost_pktcnt] + 1
        else:

return ave_rate_kbps, txtime, max_seqnum, bytecount, rx_pktcnt, ins_pktcnt, oos_pktcnt, packet_loss_hist

# main program starts here

try:
    opts, args = getopt.getopt(sys.argv[1:], “hurf:”)
except getopt.GetoptError, err:
    print “error code: “ + str(err)
    usage()
    sys.exit(2)

for o, a in opts:
    if o == “-h”:
        verbose = True
elif o in ("-h"):
    usage()
    sys.exit()
elif o in ("-v"):
    mgen_output_filename = str(a)
else:
    assert False, "unhandled option"

if verbose: print "Reading mgen file: " + mgen_output_filename
mgen_output_file = open(mgen_output_filename, 'r')
morelines = True
num_lines = 0
while morelines:
    mgen_line = mgen_output_file.readline()
    if len(mgen_line) == 0: # EOF reached
        break
    else:
        mgen_line_split = mgen_line.split(' ')
        num_lines = num_lines + 1
        if num_lines == 1:
            if mgen_line_split[1] != 'START':
                if verbose: print "no START event detected in line 1"
                break
        elif num_lines == 2:
            if mgen_line_split[1] != 'LISTEN':
                protocol = mgen_line_split[2].strip("proto")
                if protocol == "UDP":
                    line2save = 3
                elif protocol == "TCP":
                    line2save = 4
                else:
                    if verbose: print "unknown transport protocol: " + protocol
                    break
        else:
            if verbose: print "no LISTEN event detected in line 2"
            break
        else:
            if mgen_line_split[1] == 'RECV':

                #print str(num_lines) + ' ' + str(mgen_line_split)
                if num_lines == line2save:
                    prev_line_split = mgen_line_split
                    # test if first packet has sequence number 1, if not, count packet loss
                    prev_seqnum = int(prev_line_split[4].strip(' seq'))
                    max_seqnum = prev_seqnum
                    if prev_seqnum != 1:
                        if verbose: print "first packet has seq number: ", prev_seqnum
                        lost_pktcnt = prev_seqnum - 1
                        lost_pktcnt = lost_pktcnt + lost_pktcnt
                        if lost_pktcnt < 20:
                            packet_loss_hist[lost_pktcnt] = packet_loss_hist[lost_pktcnt] + 1
                        else:
                    else:
                        prev_line_split = mgen_line_split
                        #if num_lines > 6:
                        #    break
                        if mgen_line_split[1] == 'ACCEPT':
                            if verbose: print "Accept event detected in line ", num_lines
                        elif mgen_line_split[1] == 'JOIN':
                            ###### NOTE: this is a quick hack, not yet fully tested
                            line2save = 4 # This is UDP Multicast
                        else:
                            if verbose: print "RECV event detected in line: ", num_lines
                            break
else:
    mgen_output_file.close()
if verbose: print "no REC event detected in line " + str(num_lines)
if mgen_line_split[0].strip('"\n') == 'STOP':
    if verbose: print "STOP event detected in line " + str(num_lines)
    # Could be further tested if STOP is last line in the file
if mgen_line_split[0] == 'OFF':
    if verbose: print "OFF event detected in line " + str(num_lines)
    break

mgen_output_file.close()  # while loop ended

# received bit rate for in-sequence packets
ins_rate_kbps = 0
if rx_pktcnt != 0:
    ins_rate_kbps = ave_rate_kbps * (float(ins_pktcnt + ins_pktcnt)/float(rx_pktcnt))

# print results on stdout
print "Bytes received", bytecount, " >in> ", txtime, " >seconds"
print "average data rate (kbits)", ave_rate_kbps,
      "in-sequence (kbits)="
      ins_rate_kbps,
      "loss_packets="
      lost_pktcnt
      # lost_pktcnt includes discarded packets
      # lost_pktcnt + ins_pktcnt + ins_pktcnt + 1 == max_seqnum
      # (at EOF of mgen output file)
print "rx_pktcnt>", rx_pktcnt,
      "oos_pktcnt>", oos_pktcnt,
      "ins_pktcnt>", ins_pktcnt,
      "lost_packets>", lost_pktcnt
print "packet loss histogram", packet_loss_hist
sum = oos_pktcnt + ins_pktcnt + ins_pktcnt
if verbose: print "sum = oos_pktcnt + ins_pktcnt + ins_pktcnt: ", sum
      # sum == rx_pktcnt or error!
13 Appendix 3 – Analyser: Throughput CCDF

#!/usr/bin/python

# Rate (Throughput) Distribution (ccdf) plot

# It is assumed that the array R includes the samples of data for which
# the ccdf shall be obtained.
# The following 4 lines represent an example to make this script executable

R = []
L=1000
for k in range(L):
    R.append(abs(random.gauss(25,4)))

#--- Begin Rate Distribution Script ---
len_R = len(R)
R.sort()
C = []
for k in range(len_R):
    C.append(float(k+1)/len_R)
C.reverse()
#--- End Rate Distribution Script ---

# following lines produce a gnuplot output file and plot on screen under Linux/Ubuntu OS
# requires gnuplot to be installed on the execution platform
ccdf_file = open("Rate_ccdf.gp", 'w')

line_title = "plot 'Rate_ccdf.gp' using 1:2 with lines"

ccdf_file.writelines(  
['"set title 'ccd of Throughput' \n",  
"set xlabel 'Rate' \n",  
"set ylabel 'Prob.(R > Rate)' \n",  
"set style data lines \n",  
"set yrange[0:1] \n",  
"set xrange[0:50] \n",  
"set key top right \n",  
"set grid \n",  
line_title,  
"exit \n\n"])

for k in range(len_R):
    ccdf_file.write(str(R[k]) + ", " + str(C[k]) + 
")

ccdf_file.close()

os.system("gnuplot -p Rate_ccdf")
14 Appendix 4 - KPI Analysis Example Implementation

This section provides an example of executable Python 2.7.3 script which performs the KPI analysis for the FullRate, HD Film (HD Vod), HD Live and Headroom KPI services as described in section 3. This script implements the Basic Analysis, Throughput Distribution and Packet Error Distribution KPI analysers as defined in section 4. This example script has been tested under Ubuntu-Linux version 12.04. It should be seen as an implementation example. It does not serve as reference code for the KPI analysers defined in section 4 of this document. Note that this implementation does not strictly use the parameter names as defined in the main body of this document. However, the correspondence should be obvious to a programmer.

The Python script KPI_analyser.py shown below requires as input a full collection of result log files (one per room path) in csv format, as e.g. created by the test harness described in Appendices 1 and 2. The directory structure of this file collection is assumed to be formatted as illustrated in figure A.4-1 below. The right side in the figure shows the actual file names, in this example comprised of the IP addresses of the transmitting and receiving test nodes (cf. Figure 2).

![File System Required by the Described KPI Analyser](image-url)
The root of the file system is denoted in this example as TestCampaign_2013. It includes the set of result log files for several Trialists participating in the test campaign, denoted here Trialist_1, Trialist_2, etc. For each trialist, the test results obtained in different homes are collected in subfolders, home_1, home_2, etc. Within each of these home folders, there is a subfolder for each tested technology, for instance Technology_A representing a PLC technology by vendor A and Technology_B representing a WiFi technology by vendor B. Each of these technology subfolders includes at least one subfolder whose name represents a time reference in dd-mm-yyyy_hh-mm format indicating when the test was started. It includes a folder for each tested KPI service, denoted in this example FullLink, HD_Live, HD_Vod and HeadRoom. Each of these folders includes another folder that serves as a descriptor of the applied streaming scenario used for each test case. These subfolders TCPMax, NASFixedMulticast, NASFixedUdp and NASUdpMax include the result log files in csv format.

The script KPI_analyser.py processes the log files in the following steps:

Based on input parameters given in the command line, a sub-collection of result log files is selected for the KPI analysis. The selection can be made based on names used for trialist, home, technology, KPI service, and csv files. The selected csv files with their full pathnames are written into a temporary file “ListOfAnalysedTestFiles.tmp”.

The files listed in “ListOfAnalysedTestFiles.tmp” are opened one after another and processed line-by-line (each line representing a test record) by the function analyser(). In the present implementation, the KPI parameters are calculated in a single pass through the record collection. Note that in the example script two different ways of running mean, variance, and standard deviation calculation are implemented. The one referred to as “Knuth’s Art of Computer Programming algorithm” avoids possible numerical problems when the number of analysed records is very large.

After a successful run of the script, the KPI results are printed on the screen. Depending on the selected KPI service, graphical plots of the Throughput CCDF and/or Packet Error Distribution are shown on the screen (using the gnuplot utility).
#!/usr/bin/python
#
# KPI Analyser
#
# This script represents an example implementation of a basic KPI analyser.
# The analyser operates on the output log files produced by the test harness.
# These log files are assumed to be stored under logFile_root_dir.
# The input files employed for the analysis can be filtered for certain search criteria
# in the file name, type .KPI_analyser -h to obtain help on file filter criteria and
# other parameters
#
# This script implements the following KPI analysis:
#
# Basic Analysis:
# - Percent Zero Path
# - Zero Records
# - Average Rate (with and w/o zero records included)
# - Standard Deviation of the data rate
# Fixed Rate Analysis:
# - Zero Path
# - Run Time
# - Note: the "viewtimeOfUpToxx" KPI parameter not implemented
# in this example
#
# Throughput Distribution:
# - CCOF of the throughput
#
# Packet error distribution:
# - Packet Error Time (PET) as the time interval between successive
#   error events
#
import getopt, sys, os, re, time
import subprocess as sub
import signal
from datetime import date
from math import sqrt

verbose = False

logFile_root_dir = '/home/username/HGiTestResults/

trialist = "All" # select trialist or take "all"
home = "Home" # select between "Home", "home2", ..., or take "all"
device = "Vendor" # select between Vendor1, Vendor2, Vendor3, ...
service = "HDVod" # select between "HDVod", "HDLive", "HdRoom", "FullLink"
datafile_name = "all" # select test path to analyze, e.g. hub_stb, hub_sv2, hub_tv3, ... (etc)
# (depending on convention used for the respective filenames)

#error-free-seconds (efs) measurement
# Note: to avoid ambiguity of date change, the measurement should end at 24:00 hours at the latest.
# Measurement may be restarted on the following day at start_efs_time
start_efs_hours = 0. # time for start of efs measurement, to be defined in startline
start_efs_time = 3600.*start_efs_hours

end_efs_hours = 24. # time for end of efs measurement (in hours), to be defined in startline
# must be (not tested here): start_efs_hours <= end_efs_hours <= 24.

#NOTE: the StopAt24h flag MUST be set to True if start_efs_hours > 0
# continues efs measurement across date boundary REQUIRES setting of start_efs_hours = 0.
# and end_efs_hours = 24.
StopAt24h = False

if StopAt24h == False:
    start_efs_hours = 0.
    start_efs_time = 0.
    end_efs_hours = 24.
    if verbose:
        print "----- start_efs_hours at 00:00, measurement not stopped at 24:00 h "
        print end_efs_time = 3600.*end_efs_hours # end time in seconds of the day

acceptable_errors = 0 # number of acceptable packet losses, to be defined in startline

total_acc_hrs = 0.0
total_avg_rate = 0.0
total_msg_rate = 0.0
startflag = False
acc_num_lines = 0
mean = 0.0
var = 0.0
startflag_nz = False
mean_nz = 0.0
var_nz = 0.0
total_rate_cnt = 0
long(total_rate_cnt)
total_num_lines = 0
long(total_num_lines)
total_empty_rate_cnt = 0
long(total_empty_rate_cnt)
total_zero_rate_cnt = 0
long(total_zero_rate_cnt)

# Complementary cumulative distribution function of rate measurements (tail distribution)
ccdf = []

# Function defining help on usage of this script:
def usage():
    print "This script performs KPI analysis from logging data files obtained with test harness"
    print "-v verbosity"
    print "-r Log file root evaluation folder which includes the files to analyse"
    print "-d Filters for directory and file names, to select log data files for evaluation:"n
    print "-t Trialist: e.g. trialista, trialistb, ..., or all"
    print "-n Home: e.g. home1, home2, ..., or all"
    print "-o Technology/Device: e.g. Vendor1, Vendor2, Vendor3"
    print "-s Service: FullLink, HVDv0, HDTlive, HeadRoom"
    print "-N Datafile(s): Filter to allow statistics evaluation for specific file name."
    print "-p: correspondig to the tested path, e.g. hub,sb, hub,tx2, hub,tx3"n
    print "Example usage: ./KPI_analysers.py -v -F /media/381B-8EF/HGI Kpi/QC-Atheros/"

# Function for statistical analysis of a single results log file (csv format)
def analyse(service, filename, startflag, prev_mean, prev_var, acc_num_lines, startflag_nz, \n    prev_mean_nz, prev_var_nz):

    # Assumptions on the format of the results log file
    # 1st line is a header in all files, shall be ignored in the analysis
    # time (time) in all files (i.e. reference time stamp when the measurement was performed)
    # rate (Mbps) = line(4) for UDP services (i.e. service = "HVDv0", "HDTlive" or "HeadRoom"
    #       = line(1) for TCP services (service = "FullLink")

    acc_rate = 0.0
    ave_rate = 0.0
    msg_rate = 0.0
    empty_rate_cnt = 0
    zero_rate_cnt = 0
    rate_cnt = 0
    unacceptable_error_cnt = 0

    efs_flag = False
    efs_mem = []
    date_flag = False

    HGfile = open(filename, 'r')
    HGfile_line = HGfile.readline()  # first line should be title
    if len(HGfile_line) == 0:
        print "File is empty. " + filename
        return

    rate_index = 4
    pktrloss_index = 2

    if service == "FullLink":
rate_index = 1
pkt_loss_index = -1
morelines = True
num_lines = 0
losscnt = 0  # included for print statement only

while morelines:  # this should be measurement data
    HGfile_line = HGfile.readline()
    #print "HGfile_line", HGfile_line
    if len(HGfile_line) == 0:  # should be reached only when EOF is detected
        print "File ..., " + filename + ": " + str(num_lines) + " lines processed"  
        break
    else:
        # time stamp ['hh', 'mm', 'ss']
        HGfile_line_split = HGfile_line.split(' ', '!
        line_len = len(HGfile_line_split)
        #print "line_len": ", line_len
        timestamp = HGfile_line_split[0].split('!')
        #print "timestamp": , timestamp
        hrs = int(timestamp[0])
        mins = int(timestamp[1])
        secs = int(timestamp[2])
        num_lines = num_lines + 1  # counting lines with measurement data only
        # (i.e. at least timestamp available)

        # calculate test duration
        if num_lines == 1:  # keep start time of the test
            start_time = 3600 * hrs + 60 * mins + secs
            # just included for printing statement, not needed here otherwise
            # prepare efs measurement start
            mean_prev_nz = prev_mean_nz
            mean_nz = prev_mean_nz
            else:
                # need to calculate time increment for each result line since the last line does not
                # show if the test did run for several days (i.e. more than 24 hours)
                diff_seconds = next_time - start_time
                data_flag = False
                if diff_seconds < 0:
                    data_flag = True  # date change occurred, e.g. time changed from 23:59 -> 0:00
                    diff_seconds = diff_seconds + 3600 * 24
                start_time = next_time
                acc_hrs = acc_hrs + diff_seconds / 3600.

        # prepare efs measurement start
        if start_time > start_efs_time:
            efs_flag = True
        else:
            if not(data_flag): efs_flag = False

        if line_len < rate_index + 1:  # HGI test harness may produce inconsistent result records
            # when connectivity between devices under test has failed.
            # This condition captures this scenario
            rate = 0.
            empty_rate_cnt = empty_rate_cnt + 1
            if efs_flag:
                efs_mon.append(efs)  # we keep them all in case we want to get histogram later
            efs = 0.  # restart efs accumulation for any zero rate result line

        else:
            rate = float(HGfile_line_split[rate_index])
            if rate == 0:
                zero_rate_cnt = zero_rate_cnt + 1
            else:
if efs_flag:
    efs_mem.append(efs)  # we keep them all in case we want to get histogram later
    efs = 0.  # restart efs accumulation for any zero rate result line
else:
    rate_cnt = rate_cnt + 1
    ave_rate = ave_rate + rate
    msg_rate = msg_rate + rate*rate
    cdf.append(rate)  # zeros not included

### calculation of mean and variance with Knuth's algorithm (see also below)
if startflag_num == False:
    startflag_num = True
    mean_num = prev_mean_num = rate
    var_num = prev_var_num = 0.
else:
    mean_num = prev_mean_num + (rate - prev_mean_num)/rate_cnt
    var_num = prev_var_num + (rate - prev_mean_num)*(rate - mean_num)
    prev_mean_num = mean_num
    prev_var_num = var_num

if pktloss_index == 0:
    losscnt = int(HGiFile.line_split(pktloss_index))
if losscnt <= acceptable_errors:  # no or limited packet loss, i.e. 0 or <= n
    if efs_flag:
        efs += diff_secs
        # packet loss has occured
    else:
        efs = 0.  # reset for next round evaluation
        unacceptable_error_cnt = unacceptable_error_cnt + 1

# prepare efs measurement stop
if start_time > end_efs_time:
    if efs_flag:
        efs_mem.append(efs)

    efs_flag = False
    efs = 0.  # reset for next round evaluation

if date_flag:
    # note: at 24:00 hours measurement ends at the latest,
    # unless STOPAT24H is set "False"
    if verbose:
        print "date_flag True, efs_flag\", efs_flag
    if STOPAT24H:
        if efs_flag:
            efs_mem.append(efs)
        else:
            efs = 0.  # reset for next round evaluation

### alternative calculation of mean and variance of the rate with better accuracy
### Knuth's (Art of Computer Programming) running mean/var algorithm
### setting of rate > 0 for empty lines must be kept above if to be taken into the average
    acc_num_lines = acc_num_lines + 1
if startflag == False:
    startflag = True
    mean = prev_mean = rate
    var = prev_var = 0.
else:
    mean = prev_mean + (rate - prev_mean)/acc_num_lines
    var = prev_var + (rate - prev_mean)*(rate - mean)
    prev_mean = mean
    prev_var = var

if num_lines > 0:
    ave_rate = ave_rate/num_lines
    msg_rate = msg_rate/num_lines
    if efs > 0:  # last efs count may not have been saved yet
        if verbose:
            print "efs_mem\", efs_mem
        if len(efs_mem) > 0:
            max_efs = max(efs_mem)  # efs_mem may be empty array ([]) here
        else:
            max_efs = 0
    else:
        max_efs = 0

if end_efs_time < end_time:
    efs = 0.  # reset for next round evaluation
try:
    opts, args = getopt.getopt(sys.argv[1:], "hF:HM:S:D:N:")
except getopt.GetoptError, err:
    # print help information and exit:
    print "error code: " + str(err)
    usage()()
    sys.exit(2)

for o, a in opts:
    if o == "-h":
        verbose = True
    elif o in ("-v", "-V"):
        usage()
        sys.exit(2)
    elif o in ("-F"):  # trialist
        trialist = str(a)  # should be in ["all", "trialist1", "trialist2", ..., "trialist3"]
    elif o in ("-M"):
        home = str(a)  # should be in ["all", "home1", "home2", ..., "home8"]
    elif o in ("-D"):
        device = str(a)  # should be in ["Vendor1", "Vendor2", ..., or whatever identifier
    # for the tested device is used in the directory structure where the
    # csv result files are stored
    elif o in ("-S"):
        service = str(a)  # should be in ["MDVod", "MDLive", "HeadRoom", "FullLink"]
    elif o in ("-N"):
        datafile_name = str(a)  # should be in ["all", "hub_stb", "hub_tv2", "hub_tv3", ...
    elif o in ("-F"):  # logfile_root_dir
        logfile_root_dir = str(a)
    else:
        assert False, "unhandled option"

print "Reading directory structure and files from: " + logfile_root_dir

# get all csv files stored under logfile_root_dir
grepcmd = ""
if trialist != "all":
grepcmd = "| grep -- " + trialist + " "  # filter trialist, unless it is set to "all"
if verbose:
    print "grepcmd for trialist filter: " + grepcmd
if home != "all":
grepcmd = "| grep -- " + home + " "  # filter home, unless it is set to "all"
if verbose:
    print "grepcmd for home filter: " + grepcmd
# always filter device and service
grepcmd = grepcmd + " | grep -- " + device + " " + " | grep -- " + service + ""
# if -N option is not "all" (=default), filter for datafile_name
if datafile_name != "all":
grepcmd = grepcmd + "| grep -- " + datafile_name + ""
if verbose:
    print "grepcmd for all filters: " + grepcmd
cmd = "find \ + logFile_root_dir + " -name *.csv -print " + grep cmd + " >ListOfAnalysedTestFiles.tmp"
if verbose:
    print "file find command: " + cmd

os.system(cmd)

all_efm = []  # counter for paths with no error free transmissions
# read files from the list and process one by one
HGifiles = open("ListOfAnalysedTestFiles.tmp", 'r')
# data text file for output statistics
# Some statistical data computed with this script per input file is written together with the
# input filename into a text file "HG1_data_stats.txt" (one line per input file)
global HG1_stats_out_file
HG1_stats_out_file = open("HG1_data_stats.txt", 'w')
headline = "Filename, num_lines, rate_cnt, zero_rate_cnt, empty_rate_cnt, unacceptable_error_cnt, \
acc_hrs, max_efm, ave_rate, stddev \n"
HG1_stats_out_file.write(headline)
morefiles = True
while morefiles:
    HGifilename = HGifiles.readline()
    HGifilename = HGifilename.strip("\n")
    if len(HGifilename) == 0:
        break
    else:
        print "Processing ... " + HGifilename
        service_this_file = HGifilename.split('/')[1]
        if "FullLink" in HGifilename.split('/')[1]:
            service_this_file = "FullLink"
        elif "HGWid" in HGifilename.split('/')[1]:
            service_this_file = "HGWid"
        elif "HOLive" in HGifilename.split('/')[1]:
            service_this_file = "HOLive"
        else:
            service_this_file = "HeadRoom"

        # function 'analyse()' calculates statistics per csv file
        acc_hrs, ave_rate, msg_rate, rate_cnt, zero_rate_cnt, empty_rate_cnt, num_lines, startflag, mean, var, acc_num_lines, max_efm, unacceptable_error_cnt, mean_nz, var_nz = 
        analyse(service_this_file, HGifilename, startflag, mean, var, acc_num_lines, startflag_nz, mean_nz, var_nz)

        max_efm = max_efm/60.
        if max_efm > 0.:
            all_efm.append(max_efm)  # efm's of all paths and files
        else:
            zero_efm_cnt = zero_efm_cnt + 1

        print "acc hours", acc_hrs, " ave_rate (Mbps)"", ave_rate, " rate_cnt>", rate_cnt, " zero_rate_cnt", zero_rate_cnt, " empty_rate_cnt", empty_rate_cnt, " num_lines">, num_lines
        sqrt_var = sqrt(var/float(acc_num_lines - 1))
        print "mean rate", mean
        print "sqrt(var)", sqrt_var
        print "max error free minutes", max_efm
        print "unacceptable_error_cnt", unacceptable_error_cnt

        fraction_records_inerror = 100.*(empty_rate_cnt+zero_rate_cnt+unacceptable_error_cnt)/num_lines
        print "fraction_records_inerror (%)", fraction_records_inerror

        print "mean_nz, var_nz: ", mean_nz, var_nz, mean, var

    # we calculate statistics over all files here;
    # if following depends how rate statistics is assessed (empty lines counted separately or not)!
    if service_this_file == "FullLink":
        total_acc_hrs = total_acc_hrs + acc_hrs/6.
else:
    total_acc_hrs = total_acc_hrs+ acc_hrs/3.

# parameters for analysis with zeros included
total_ave_rate = total_ave_rate * num_lines * ave_rate
total_msg_rate = total_msg_rate * num_lines * msg_rate

total_rate_cnt = total_rate_cnt * rate_cnt
total_num_lines = total_num_lines * num_lines
total_empty_rate_cnt = total_empty_rate_cnt * empty_rate_cnt
total_zero_rate_cnt = total_zero_rate_cnt * zero_rate_cnt

HGIfiles.close()  # while loop ended
HGI_stats_out_file.close()

# parameters for analysis with zeros excluded (_nz - no zeros)
ave_rate_nz = total_ave_rate/total_rate_cnt
msg_rate_nz = total_msg_rate/total_rate_cnt
arg_root = msg_rate_nz - ave_rate_nz
if arg_root > 0:
    std_dev_nz = sqrt(arg_root)
else:
    std_dev_nz = "NaN"

# the following includes some examples of graphical representation of the statistical results
#-------------------------------------------------------------
#-------- ccdf plot ----------------------------------------
len_ccdf = len(ccdf)
ccdf.sort()
prob = []
for k in range(len_ccdf):
    prob.append(float(k+1)/len_ccdf)

prob.reverse()

print len_ccdf, len_ccdf
ccdf_file = open("ccdf.txt", 'w')
line_title = "plot 'ccdf.txt' using 1:2 with lines t '+' + " ' " + service + " \n"
ccdf_file.writeln(["set title 'ccdf of Throughput' \n", "set xlabel 'Rate' \n", "set ylabel 'Prob.(R > Rate)' \n", "set style data lines \n", "set yrange[0:1] \n", "set xrange[0:50] \n", "set key top right \n", "set grid \n", "exit \n"])
for k in range(len_ccdf):
    ccdf_file.write(str(ccdf[k]) + ", " + str(prob[k]) + "\n")
ccdf_file.close()
system("gnuplot -p ccdf.txt")

#-------- max error free transmission time plot ------------------------
# this is calculated only for fixed rate services such as HDVod and HDLive
if service in ["HD Vod", "HDVod", "HD Live", "HDLive"]:
    len_all_efm = len(all_efm)
    all_efm.sort()
    prob = []
    for k in range(len_all_efm):
        prob.append(float(k+1)/len_all_efm)
prob.reverse()

line_title = "plot 'efm.gp' using 1:2 with lines t * + "'" + service + "' \n"efm_file = open("efm.gp", 'w')
efm_file.writelines(["set title 'CDF of error free transmission time' \n", "set xlabel 'Time (minutes)' \n", "set ylabel 'Prob.(T > Time)' \n", "set style data lines \n", "set yrange[0:1] \n", "set key top right \n", "set grid \n", "line_title, \n", "exit \n"])

for k in range(len_all_efm):
efm_file.write(str(all_efm[k]) + "", " + str(prob[k]) + "\n")
efm_file.close()

os.system("gnuplot -p efm.gp")

# Statistics calculated over all selected input files is printed onto the screen

print "----- statistics over all files - zeros included: --------------------"
print "accumulated test runtime (hours)"", total_acc_hrs
print "ave_rate (Mbps)", ave_rate, " rate_cnt", total_rate_cnt, "
" zero_rate_cnt", total_zero_rate_cnt, 
" empty_rate_cnt", total_empty_rate_cnt, " num_lines", total_num_lines
print "Zero", 100 * float(total_zero_rate_cnt+total_empty_rate_cnt)/float(total_num_lines)
print "Std. Dev. of rate variation"", std_dev

sqrt_var = sqrt(var/float(acc_num_lines - 1))

print "----- Mean rate (Knutx)"", mean
print "Sqrt(var) (Knutx)"", sqrt_var

print "----- statistics over all files - zeros excluded: --------------------"
print "accumulated test runtime (hours)"", total_acc_hrs
print "ave_rate_nz (Mbps)", ave_rate_nz
print "Std. Dev. (nz) of rate variation"", std_dev_nz

sqrt_var_nz = sqrt(var_nz/float(rate_cnt - 1))

print "Mean rate_nz (Knutx)"", mean_nz
print "Sqrt(var) (Knutx)"", sqrt_var_nz

print "all efm"
print all_efm
print "Number of paths with no error free transmission"", zero_efm_cnt

print "total_num_lines", total_num_lines, " acc_num_lines", acc_num_lines
15 Appendix 5 – Test Control Schema XSD

<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"

targetNamespace="http://www.homegatewayinitiative.org"
xmlns="http://www.homegatewayinitiative.org"
xmlns:Analysis="http://www.homegatewayinitiative2.org"
elementFormDefault="qualified">

<xs:import namespace="http://www.homegatewayinitiative2.org"
schemaLocation="KPIAnalysisSchema.xsd"/>

<xs:element name="KpiProfile">

<xs:complexType>
<xs:sequence>
<xs:element name="Name" type="xs:string"/>
<xs:element name="TechnologiesUsed" minOccurs="1" maxOccurs="unbounded"
type="TechnologySpec"/>
<xs:element name="NodeInfo">
<xs:complexType>
<xs:sequence>
<xs:element name="Hub">
<xs:complexType>
<xs:sequence>

</xs:complexType></xs:element></xs:sequence></xs:complexType></xs:element>
<xs:element name="PhysicalLocation" type="xs:string"/>
<xs:element name="Address" type="xs:string" fixed="172.16.1.1"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="Stb">
<xs:complexType>
<xs:sequence>
<xs:element name="PhysicalLocation" type="xs:string"/>
<xs:element name="Address" type="xs:string" fixed="172.16.1.2"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="Tv2">
<xs:complexType>
<xs:sequence>
<xs:element name="PhysicalLocation" type="xs:string"/>
<xs:element name="Address" type="xs:string" fixed="172.16.1.3"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="Tv3">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="PhysicalLocation" type="xs:string" />
            <xs:element name="Address" type="xs:string" fixed="172.16.1.4" />
        </xs:sequence>
    </xs:complexType>
</xs:element>

<!-- NodeInfo -->
</xs:element>

<xs:element name="KpiTestSet" minOccurs="1" maxOccurs="unbounded" type="KpiTest" />

<!-- KpiProfile -->
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:complexType name="KpiTest">
    <xs:sequence>
        ...
    </xs:sequence>
</xs:complexType>
<xs:element name="TestHomes" type="xs:string"/>

<xsl:element name="KPI" minOccurs="1" maxOccurs="unbounded">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Service" type="Services"/>
            <xs:element name="Description" type="xs:string"/>
            <xs:element name="Duration" type="xs:integer"/>
            <xs:element name="PathControl" type="TestPathControl"/>
            <xs:element name="Paths" minOccurs="1" maxOccurs="unbounded" type="TestPaths"/>
            <xs:element name="OnControlError" type="ControlError"/>
            <xs:element name="DelayFromStart" type="xs:integer"/>
            <xs:element name="Analysis" type="Analysis:KpiAnalyser"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:complexType name="TestPaths">
    <xs:sequence>
        <xs:element name="HubToStb" type="xs:string"/>
        <xs:element name="HubToTv2" type="xs:string"/>
        <xs:element name="HubToTv3" type="xs:string"/>
        <xs:element name="StbToTv2" type="xs:string"/>
        <xs:element name="StbToTv3" type="xs:string"/>
        <xs:element name="Tv2ToTv3" type="xs:string"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="TestPathControl">
    <xs:choice>
        <xs:element name="SerialisedPathStreaming" type="xs:string"/>
        <xs:element name="ConcurrentPathStreaming" type="xs:string"/>
    </xs:choice>
</xs:complexType>

<xs:complexType name="NodesInSystem">
    <xs:choice>
        <xs:element name="Hub" type="xs:string"/>
        <xs:element name="Stb" type="xs:string"/>
        <xs:element name="Tv2" type="xs:string"/>
    </xs:choice>
</xs:complexType>
<xs:element name="Tv3" type="xs:string"/>
</xs:choice>
</xs:complexType>

<xs:complexType name="StreamSpec">
    <xs:choice>
        <xs:element name="Udp" type="Udp"/>
        <xs:element name="Tcp" type="Tcp"/>
        <xs:element name="Multicast" type="Multicast"/>
    </xs:choice>
</xs:complexType>

<xs:complexType name="Services">
    <xs:choice>
        <xs:element name="FullRate" type="Tcp"/>
        <xs:element name="Headroom" type="Udp"/>
        <xs:element name="HDLive" type="Multicast"/>
        <xs:element name="HDVod" type="Udp"/>
    </xs:choice>
</xs:complexType>
<xs:complexType name="Udp">
  <xs:sequence>
    <xs:element name="Protocol" type="xs:string" fixed="UDP"/>
    <xs:element name="TransferSize" type="xs:integer"/>
    <xs:element name="Rate" type="RateSpec"/>
    <xs:element name="LoggedInfo" type="UdpRecordFormat"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Tcp">
  <xs:sequence>
    <xs:element name="Protocol" type="xs:string" fixed="TCP"/>
    <xs:element name="TransferSize" type="xs:integer"/>
    <xs:element name="Rate" type="RateSpec"/>
    <xs:element name="LoggedInfo" type="TcpRecordFormat"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Multicast">
  <xs:sequence>
    <xs:element name="Protocol" type="xs:string" fixed="Multicast"/>
    <xs:element name="MulticastAddress" type="xs:string" fixed="225.1.2.3"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="RateSpec">
  <xs:choice>
    <xs:element name="Fixed" type="xs:integer"/>
    <xs:element name="Max" type="xs:string" fixed="Max"/>
  </xs:choice>
</xs:complexType>

<xs:complexType name="UdpRecordFormat">
  <xs:sequence>
    <!-- DD:MM:YY HH:MM:SS-->
    <xs:element name="TimeStamp" type="xs:string" fixed="TimeStamp"/>
    <xs:element name="NosRecvdPacket" type="xs:string" fixed="NosRecvdPacket"/>
    <xs:element name="Drops" type="xs:string" fixed="Drops"/>
    <xs:element name="DropClusters" type="xs:string" fixed="DropClusters"/>
  </xs:sequence>
</xs:complexType>
<xs:element name="Mbps" type="xs:string" fixed="Mbps"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="TcpRecordFormat">
<xs:sequence>
<!-- DD:MM:YY HH:MM:SS-->
<xs:element name="TimeStamp" type="xs:string" fixed="TimeStamp"/>
<xs:element name="Mbps" type="xs:string" fixed="Mbps"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="ControlError">
<xs:choice>
<xs:element name="InvalidateResultsForAllNodes" fixed="InvalidateResultsForAllNodes"/>
<xs:element name="InvalidateResultsForNodeOnly" fixed="InvalidateResultsForNodeOnly"/>
<xs:element name="RecordInErrorFileOnly" fixed="RecordInErrorFileOnly"/>
<xs:element name="RecordInErrorFileOnlyAndInvalidForAllNodes" fixed="RecordInErrorFileOnlyAndInvalidForAllNodes"/>
</xs:choice>
</xs:complexType>
<xs:complexType name="TechnologySpec">
  <xs:sequence>
    <xs:element name="TechName" type="xs:string"/>
    <xs:element name="TechType" type="xs:string"/>
    <xs:element name="Placement" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
</xs:schema>
16 APPENDIX 6 – ANALYSIS SPECIFICATION SCHEMA XSD

<?xml version="1.0"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.homegatewayinitiative2.org"
    xmlns="http://www.homegatewayinitiative2.org"
    elementFormDefault="qualified">

    <xs:element name="KpiAnalysis">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Analyser" type="KpiAnalyser" minOccurs="1" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:complexType name="KpiAnalyser">
        <xs:choice>
            <xs:element name="BasicAnalysis" type="Basic"/>
            <xs:element name="FixedRateAnalysis" type="FixedRate"/>
        </xs:choice>
    </xs:complexType>

</xs:schema>
<xs:complexType name="Basic">
  <xs:sequence>
    <xs:element name="Inputs" type="InputSelections"/>
    <xs:element name="Outputs" type="BasicOutputs"/>
    <xs:element name="Scope" type="ScopeControl"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="FixedRate">
  <xs:sequence>
    <xs:element name="Inputs" type="InputSelections"/>
    <xs:element name="Outputs" type="FixedOutputs"/>
    <xs:element name="Scope" type="ScopeControl"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="InputSelections">
  <xs:sequence>
    <xs:element name="From" type="TimeControl"/>
  </xs:sequence>
</xs:complexType>
<xs:element name="Until" type="TimeControl"/>
<xs:element name="AnalysisRate" type="RateControl"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="ScopeControl">
<xs:sequence>
<xs:element name="Technology" type="TechnologyControl"/>
<xs:element name="Trialist" type="TrialistControl"/>
<xs:element name="Path" type="PathControl"/>
</xs:sequence>
</xs:complexType>

<!-- Placeholders for output values -->
<xs:complexType name="BasicOutputs">
<xs:sequence>
<xs:element name="Average" type="xs:string"/>
<xs:element name="StdDeviation" type="xs:string"/>
<xs:element name="Runtime" type="xs:string"/>
<xs:element name="PercentZero" type="xs:string"/>
<xs:element name="PercentZeroPathsExcluded" type="xs:string"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="RateControl">
    <xs:choice>
        <xs:element name="None" type="xs:string" fixed="None"/>
        <xs:element name="Rate" type="xs:integer"/>
    </xs:choice>
</xs:complexType>

<xs:complexType name="TimeControl">
    <xs:choice>
        <xs:element name="None" type="xs:string" fixed="None"/>
        <xs:element name="When" type="xs:time"/>
    </xs:choice>
</xs:complexType>

<!-- Placeholders for output values-->
<xs:complexType name="FixedOutputs">
    <xs:sequence>
        <xs:element name="PercentZeroPathsExcluded" type="xs:string"/>
        <xs:element name="ExclusiveViewtimeOfUpTo30mins" type="xs:string"/>
        <xs:element name="ExclusiveViewtimeOfUpTo1Hour" type="xs:string"/>
    </xs:sequence>
</xs:complexType>
<xs:element name="ExclusiveViewtimeOfUpTo2Hours" type="xs:string"/>
<xs:element name="ExclusiveViewtimeOfUpTo4Hours" type="xs:string"/>
<xs:element name="Runtime" type="xs:string"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="TrialistControl">
  <xs:choice>
    <xs:element name="ForEachTrialist" type="xs:string" fixed="ForEachTrialist"/>
    <xs:element name="ForAllTrialists" type="xs:string" fixed="ForAllTrialists"/>
    <xs:element name="ForSpecificTrialist" type="xs:string" fixed="ForSpecificTrialist"/>
  </xs:choice>
</xs:complexType>

<xs:complexType name="PathControl">
  <xs:choice>
    <xs:element name="AllPaths" type="xs:string" fixed="AllPaths"/>
    <xs:element name="Hub-Stb" type="xs:string" fixed="Hub-Stb"/>
  </xs:choice>
</xs:complexType>
<xs:element name="Hub-Tv2" type="xs:string" fixed="Hub-Tv2" />
<xs:element name="Hub-Tv3" type="xs:string" fixed="Hub-Tv3" />
<xs:element name="Stb-Tv2" type="xs:string" fixed="Stb-Tv2" />
<xs:element name="Stb-Tv3" type="xs:string" fixed="Stb-Tv3" />
<xs:element name="Tv2-Tv3" type="xs:string" fixed="Tv2-Tv3" />
</xs:choice>
</xs:complexType>

<xs:complexType name="TechnologyControl">
  <xs:choice>
    <xs:element name="ForEachTechnology" type="xs:string" fixed="ForEachTechnology" />
    <xs:element name="ForAllTechnologies" type="xs:string" fixed="ForAllTechnologies" />
    <xs:element name="ForSpecificTechnology" type="xs:string" />
  </xs:choice>
</xs:complexType>

</xs:schema>
17 APPENDIX 7 – XML PROFILE

<?xml version="1.0"?>

<KpiProfile xmlns:Ana="http://www.homegatewayinitiative.org"
 xsi:schemaLocation="http://www.homegatewayinitiative.org KPIProfileSchema.xsd"
 xmlns:TechType="http://www.homegatewayinitiative.org"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

Profile</Name><TechnologiesUsed><TechName>T+W</TechName><TechType>Plt</TechType><Placement>in brd</Placement><TechnologiesUsed><TechName>Celeno</TechName><TechType>Wi-Fi</TechType><Placement>as a user would place it</Placement><TechnologiesUsed><NodeInfo><Hub><PhysicalLocation>By the users main Tv</PhysicalLocation>
</Hub></NodeInfo>
</Address>172.16.1.1</Address>
</Stb><PhysicalLocation>By the users main Tv</PhysicalLocation>

<Address>172.16.1.2</Address>
</Stb><PhysicalLocation>By the users second TV (or where this would be)</PhysicalLocation>

<Address>172.16.1.3</Address>
</Tv2><PhysicalLocation>By the users third TV (or where this would be)</PhysicalLocation>

<Address>172.16.1.4</Address>
</Tv3></NodeInfo>

<!-- Full rate entry-->

<TestHomes>IP5 3RE, an other postcodes for test, used as info only</TestHomes><KPI><Service><FullRate></Service><Protocol>TCP</Protocol><TransferSize>65536</TransferSize><Rate><Max/></Rate><LoggedInfo><TimeStamp/></LoggedInfo><FullRate></Service><Description>All paths</Description><Duration>15</Duration><PathControl><SerialisedPathStreaming/></PathControl><Paths><HubToStb>Yes</HubToStb><HubToTv2>Yes</HubToTv2><HubToTv3>Yes</HubToTv3><StbToTv2>Yes</StbToTv2><StbToTv3>Yes</StbToTv3><Tv2ToTv3>Yes</Tv2ToTv3></Paths><OnControlError><OnControlErrorError><DelayFromStart>0</DelayFromStart><Analysis><Ana:BasicAnalysis><Ana:Inputs><Ana:From><Ana:None/></Ana:From><Ana:Until><Ana:None/></Ana:Until><Ana:AnalysisRate><Ana:None/></Ana:AnalysisRate><Ana:Inputs><Ana:Outputs><Ana:Average/></Ana:Average></Ana:Outputs><Ana:Scope><Ana:Technology><Ana:PercentZeroPathsExcluded/></Ana:PercentZeroPathsExcluded></Ana:Technology><Ana:Trialist><Ana:ForAllTechnology/></Ana:Trialist></Ana:Technology></Ana:Trialist><Ana:ForAllTr...
HD Vod Entry udp to stop TCP issues with concurrent streaming

```
<KPI><Service><HDVod><Protocol><LogInfo><TimeRecvdPacket/>Rates/<Mbs/>><LogInfo><TimeRecvdPacket/>Rates/<Mbs/>
```

HD Live Entry

```
<KPI><Service><HDLive><Protocol><MulticastAddress><TransferSize><Rate>/Fixed/>><LogoInfo><TimeRecvdPacket/>Rates/<Mbs/>><LogInfo><TimeRecvdPacket/>Rates/<Mbs/>
```

Headroom concurrent paths maximum rate

```
<KPI><Service><Headroom><Protocol><TransferSize><Rate>/Max/>
```

© HGI 2015 – All rights reserved