Institute of Communications and Computer Systems (ICCS)

"System for Performance Evaluation of Broadband connection Services (SPEBS)"

Version: 1.3 - DRAFT

December 2011

1.1 Introduction

SPEBS (System for Performance Evaluation of Broadband connections) is the Monitoring and Evaluation framework for broadband connections (xDSL services). The proposed system utilizes tools/functionality offered by the Measurement-lab, in order to enable users, regulators as well as ISPs to draw conclusions regarding the quality of the delivered broadband services per country / globally. Users are encouraged to register with M-lab details such as location, broadband connection type and nominal internet access data rate (up/down stream) in order for the system to:

- Produce and publish statistics illustrating broadband connections' quality per internet service provider, per connection type and per region - in time.
- Carry out scientific research in the area of evaluating Internet Protocols' performance (Max Planck Institute for Software Systems, etc).

In particular, this document describes the overall system conceptual architecture including SPEBS enabling components, relationships and interactions between the various parts of the architecture as well as system design and implementation.

1.2 System for Performance Evaluation of Broadband connection Services

1.2.1 Overall System Architecture and architectural strategies

The proposed architecture aims at extending the functionality provided by the M-Lab framework. Therefore, *SPEBS*[1] is introduced, along with the system's interactions with *ISP* and *Locator* entities, the End-User and the Measurement Service. The End-User/ Measurement Service entities/interaction are primarily defined by the M-Lab framework.

The SPEBS architecture is comprised of the following components:

1. SPEBS data store: The SPEBS Data Store is a MySQL database that contains all the necessary end-user information. Specifically the SPEBS Data Store contains
    - End-user account information (e.g. username, e-mail etc)
    - End -user connection information (e.g. ISP, nominal rate, description etc)
    - Measurement information per connection/ end-user
    - Location information for local exchanges
    - IP address ranges per *ISP*

[1] See Appendix A for roles description

- System specific information.

2. SPEBS engine: The SPEBS engine handles the appropriate information exchange among the Measurement Service tools (or End-User measuring applet) and the SPEBS Data Store.
3. SPEBS front – end: The appropriate Interface among the End-User and the SPEBS system.
4. SPEBS Web Services: In order to support interoperable interactions among the proposed SPEBS framework (in particular the SPEBS Data Store) and the various entities (ISPs and Locators) of the overall measurement system architecture, appropriate information exchange services are defined as soap web services.

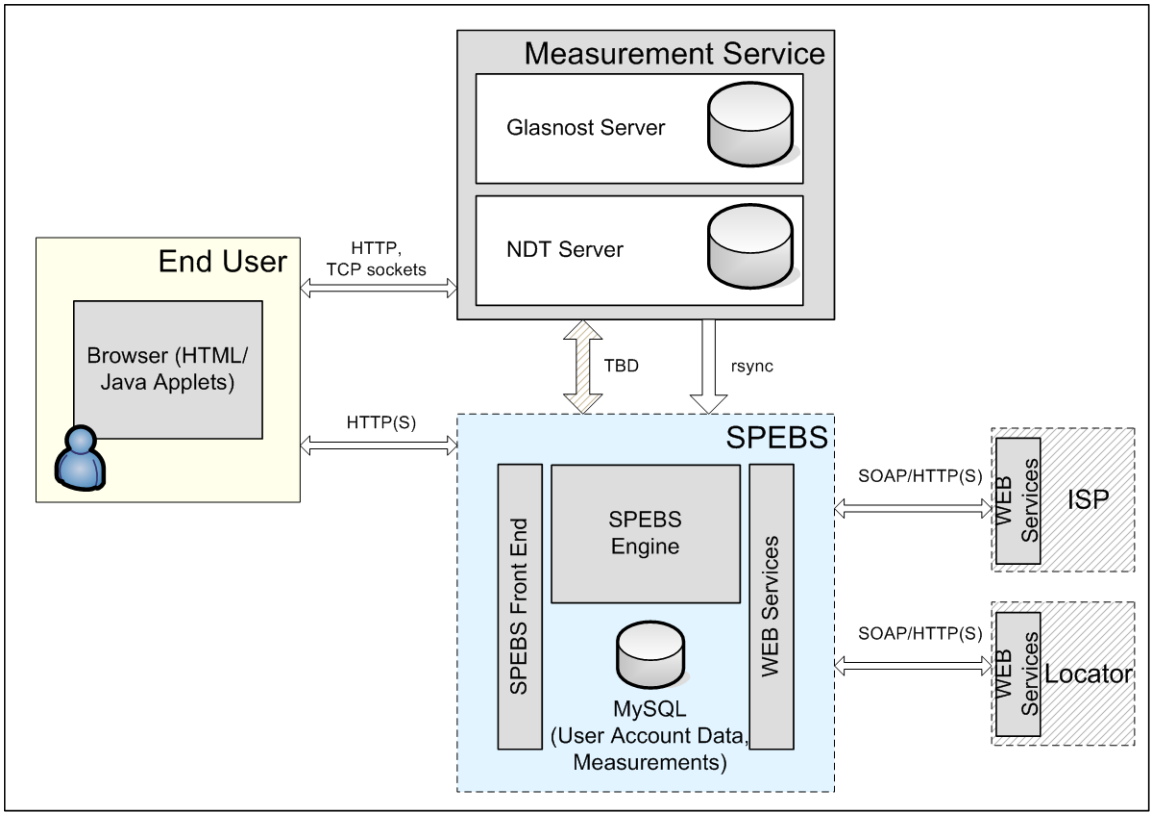

Figure 1: Overall System Conceptual Architecture

1.2.2 Detailed system design

1.2.2.1 Measurement Tools

1.2.2.1.1 Glasnost

Architecture

The parts that comprise the mechanism by which glasnost carries out measurements are as follows:

1. Server-side PHP code that produces the pages that contain the measurement applet and are also responsible for presenting the measurement results.
2. A Java Applet that is launched at the end-user side and carries out the measurement.
3. A server-side software daemon (C) that works in conjunction with the Applet.

A typical measurement is being carried out as follows:

1. The end-user visits the initial glasnost page (bttest.php). The end-user is presented with the choice to carry out a full or simple measurement and the button to commit it.
2. When the user selects the type of measurement and presses the start button, the browser is redirected to the bt.php page, where the measuring applet is presented. Simultaneously, the server executes the bt_client daemon, which will remain alive throughout the duration of the measurement.
3. Upon measurement completion, the applet signals (appletContext.showDocument(url)) the browser to go to bt.php[2], with all the measurement results as additional HTTP GET arguments.

Measurement Parameters

The measurement parameters are placed inside the applet containing tag as *parameters* e.g.:

<param name="ID" value="147.102.220.40">

<param name="port" value="6881">

<param name="up" value="true">

<param name="down" value="true">

<param name="tcp" value="true">

[2] Measurement results are presented to the end-user.

<param name="port2" value="4711">

<param name="duration" value="10">

<param name="repeat" value="2">

<param name="cache_option" value="no">

<param name="cache_version" value="2.0">

<param name="errorpage" value="/glasnost///error.php">

<param name="testpage" value="/glasnost///bt.php">

Measurement Mechanism

The measurement is being carried out at the client side, thus at the end of the measurement the data is available to the Java Applet and the appropriate calculations are being done by the code that comprises it. During the measurement, the applet communicates with the server side daemon through a TCP control channel. This channel is established prior to the commencement of the measurement and remains open throughout its duration. The control channel protocol is not formally defined, but rather follows a simple ad hoc request-response schema where the applet gives instructions and the daemon responds, usually with a simple "ok". Possible control channel commands include signals for the daemon to open certain ports server-side where a particular measurement is about to be done ("setup" command) or messages destined to be stored in the server side log ("log" commands). Here is a typical example of applet and daemon communication:

applet : setup downstream 11 port 6881 .
daemon : ok
applet : log Windows XP,x86,5.1,Sun Microsystems Inc.,1.6.0_17
applet : bt downstream 21 port 6881 .
daemon : ok
applet : log Transferred 1.79830784E8 bytes in 20.0 seconds: 7.19323136E7 bps
applet : shutdown
daemon : ok

User Identifier

To accommodate certain needs, a field that represents the identity of the user with a (possibly encrypted) id may need to be added to the applet parameters. An example for such a field would be:

<param name="user_hash" value="user_anonymized_hash_0001">

This parameter would subsequently be available to the applet code. At the end of the measurement procedure, measurement results must be transmitted to SPEBS. Therefore the following alternative methods are proposed:

1. Transmit the *user_hash* in an additional HTTP GET parameter to bt.php. The bt.php obviously needs to be modified in order to be able to transmit the results given to it to the appropriate third party (in the same or in another host e.g. SPEBS engine) that must receive the measurements.

2. Transmit the *user_hash* directly from the applet to the collecting entity by the applet itself. For example, the applet could conceivably use an http client library (example: from java.net.* or apache http client library) and submit the measurement results directly to a measurement collection server along with their respective *user_hash*. In that case, the mechanism of results submission is independent from the server side code, thus the bt.php code does not need to be modified at all. The submission of the measurement results can either be done to the same server as where the measurement was made, or to an independent server that runs the collection service (e.g. SPEBS engine). For the second option, an additional parameter telling the applet where to submit the results must be supplied.

1.2.2.1.2 Network Diagnostic Tool

Architecture

Contrast to Glasnost, NDT does not use server-side generated HTTP pages for presenting the measurement results. The NDT client program has two distinct implementations and can be used either standalone or inside a web browser as an applet. The parts of the NDT architecture are as follows:

- Web100 kernel patch[3] for linux kernels 2.4.x or 2.6.x coming from the web100 project (http://www.web100.org/), allowing the monitoring of various TCP parameters per connection. A summary of the measuring probes implemented by the patch can be found at http://www.web100.org/download/kernel/2.5.30/tcp-kis.txt while a more

[3] Unfortunately until today the web100 patch has not been incorporated inside the Linux kernel, so to use the web100 tools, one need to recompile the kernel by hand. The only other OS implementing RFC4898 as of today is windows vista (and the newer 7), see http://msdn.microsoft.com/en-us/library/ee391627(VS.85).aspx for details. Similar mechanisms in FreeBSD were implemented by the NewTCP project of Swinburne University's Centre for Advanced Internet Architectures: http://caia.swin.edu.au/urp/newtcp/. Lastly, Sun Solaris implements DTrace which encompasses a large number of kernel probes, possible some of which can be used to monitor TCP connection parameters.

comprehensive description can found at RFC4898 (http://www.ietf.org/rfc/rfc4898.txt). The web100 patch includes both the kernel space code that implements the TCP probes and the library (libweb100) to access the probes from within userland programs. The library is accompanied by programs that can be used directly for monitoring TCP connections in systems with the web100 kernel patch.

- Server-side daemon written in C (web100src) that resides in the NDT server and services NDT clients wishing to carry out measurements.
- Java class (Tcpbw100) able to communicate with a web100srv daemon running in a remote host and carry out measurements. The Tcpbw100 class extends the JApplet class, so it can be embedded in a web page as an applet, but also has a main method so it can alternatively be used standalone.
- Client-side program written in C (web100clt) able to communicate with a web100srv daemon running in a remote host and carry out measurements. Its usefulness is analogous to that of the applet.
- Simple HTTP server (fakewww) used in cases where the applet must be served to the client through HTTP.

A typical measurement is being carried out roughly as follows:

1. Client connects to server using the ndt protocol and carries out the measurement directly. Regardless whether the client is being used as a Java applet or a standalone program, the way of operation is exactly the same
2. Upon measurement completion, the presentation of the results is done by the Tcpbw100 class directly. The Tcpbw100 does not interact with the (hypothetical) web container like in the case of Glasnost.

Measurement Parameters

The only parameter that the Tcpbw100 needs to commit a measurement is the address of a measuring server host running a web100srv daemon. In the case of direct invocation of Tcpbw100, the parameter **must** be explicitly supplied, for example:

java -jar Tcpbw100.jar ndt.iupui.ath01.measurement-lab.org

In the case where the Tcpbw100 is being used as an applet through fakewww, it is actually able to guess the measuring server by calling getCodeBase().getHost() , which returns the server from

which the browser downloaded the applet. Alternatively, an explicit parameter stating the address of the measuring server is allowed as in the following example:

<PARAM NAME="testingServer" VALUE="ndt.iupui.ath01.measurement-lab.org">[4]

Other optional parameters are:

- ***disableStatistics***
- ***disableDetails***
- ***disableMailto***
- ***disableOptions***
- ***enableMultipleTests***
- ***U, H and subject concerning email problem reports***

e.g.

<PARAM NAME="H" VALUE="gmail.com">

<PARAM NAME="U" VALUE="rcarlson501">

<PARAM NAME="subject" VALUE="Trouble report from mlab1.ath01.measurement-lab.org">

Measurement Mechanism

The web100srv daemon listens to ports 3001 (command channel), 3002 (client streaming data to server) and 3003 (server streaming data to client). By using the web100 probes and the libpcap library, web100srv can asses the network bandwidth between the client and server along with other miscellaneous quality indicators. All the measurement data is being made accessible to the client through the command channel. Unfortunately the command channel uses an ad hoc protocol of binary requests and responses that cannot be easily understood by a human (contrary to Glasnost). So, to understand this protocol and consider its possible modification or extension, extensive study of the code implementing it is required. Readers of this document wishing to know more can look up file network.c under src/ on the NDT web100srv source tree and take a look at send_msg() and recv_msg().

User Identifier

[4] note that if the above mentioned parameter is present, it takes precedence by default

Exactly like in glasnost, the addition of a field representing the identity of the user with a (possibly anonymized) identifier can be done by adding a parameter like:

<param name="user_hash" value="user_anonymized_hash_0001">

The parameter will be subsequently available everywhere throughout the applet code. When at the end of the measurement sequence that time comes that the measurements must be transmitted to SPEBS the following methods are available:

1. Transmit the *user_hash* through the control channel. Reverse engineering the control protocol may require considerable effort. Therefore it is proposed that a message containing the results in their entirety be added right before the control channel is about to change. To achieve this, both the web100srv and the Tcpbw100.java must be modified to exchange one more message right before proceeding to close the control TCP channel. As the development and testing of this mechanism requires modifying the daemon of a measuring server, any attempt to do so will naturally require the installation of a GNU/Linux host with the web100 patch. Upon receiving the measurement data, the web100srv daemon would typically proceed to insert the data to the SPEBS engine, for example by using a direct connection to the database, through a web service or any other conceivable means.
2. Transmit the *user_hash* along with the rest of the measurement data using the Java client side code itself, with exactly the same code that could be used by the glasnost proposed modification #2. This solution has the additional advantage that allows the use of common components for both glasnost and NDT, implemented in Java.

1.2.2.2 SPEBS Framework

SPEBS is comprised of four basic components: the SPEBS engine, the SPEBS Data Store for storing measurement results and independent systems' profile information, the SPEBS front-end defining end-user/ SPEBS interaction and finally the necessary web services/clients for communicating with the various ISPs / Locator.

1.2.2.2.1 SPEBS Data Store

The following Entity-Relationship diagram illustrates the SPEBS Data Store schema. SPEBS stores information about ISPs, Local Exchanges, and End-Users, as well as the results of their measurements emerging from the use of MLAB tools.

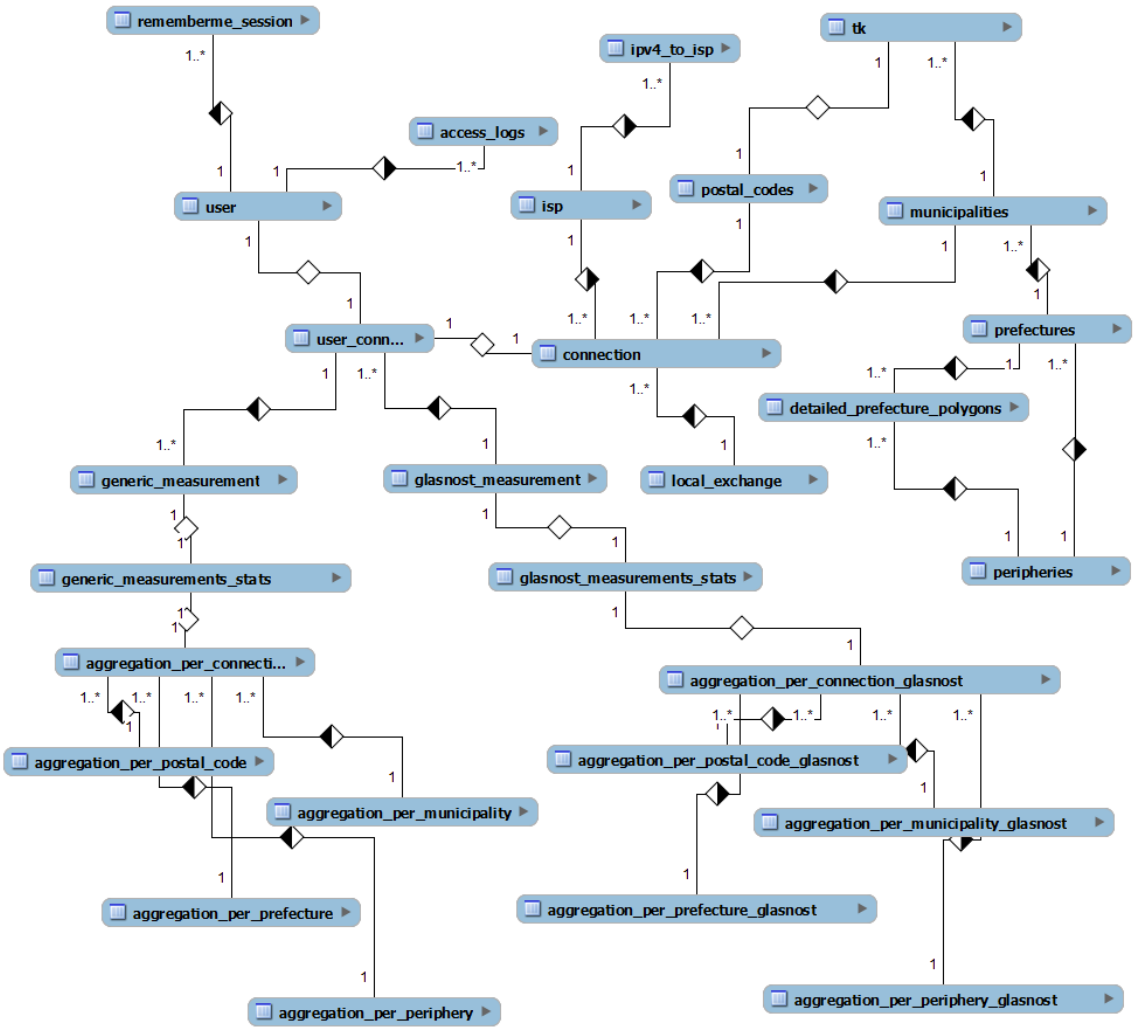

Figure 2: SPEBS E-R diagram

1.2.2.2.2 Web Services

Defined entities exchange data through SOAP messages. Services exposed by each part are accumulated in three separate groups based on the entity where they reside. Each underlying entity provides through internal procedures the appropriate answers to the received services' requests.

regulatorWS services

Locator (e.g. National Regulator) preserves information regarding user location and corresponding local exchange / ISP information. Thus, regulator exposes the following services:

- TrackConnection

  Input : ipaddress, timestamp

  Output: isp_id, exchange_id, local_loop_name

- IdentifyUserConnection

  Input : ipaddress, timestamp

  Output: isp_id, exchange_id, local_loop_name

ispWS services

In addition to what locator knows about the user, an ISP can also provide customer contract information. Thus, each ISP should expose the following services:

- TrackUser

  Input : ipaddress, timestamp

  Output: longitude, latitude, elevation, tolerance, address, postal_code, city, country

- ConfirmUserPosition

  Input : ipaddress, timestamp, longitude, latitude, elevation, tolerance, address, postal_code, city, country

  Output: true/false

- ConnectionInfo

  Input : ipaddress, timestamp

  Output: longitude, latitude, elevation, tolerance, address, postal_code, city, country isp_id, exchange_id, purchased_bandwidth_kbps, local_loop_name, distance_to_exchange, contention_ratio

sapesWS service

The SPEBS system stores information, measurements performed by end users in time and space dimensions, in order to present statistics on broadband connections. SPEBS is not designed to

disseminate this information but *should* assure data accuracy. For that reason, ISPs and Locator *can* update SPEBS Data Store through the following services[5]:

- AddExchange

  Input : unique_name, description, latitude, longitude, address, postal_code, city, country

  Output: human readable text answer

- RemoveExchange

  Input : unique_name

  Output: human readable text answer

- CheckValidExchange

  Input : unique_name

  Output: human readable text answer

- AddISP

  Input : unique_name, description

  Output: human readable text answer

- RemoveISP

  Input : ISPname

  Output: human readable text answer

- DeclareIPspace

  Input : ISPname, start, end, ip_version

  Output: human readable text answer

[5] Services' parameters correspond to database fields

APPENDIX A

Measurement Service: Measurement lab hosts in widely distributed servers appropriate Internet measurement tools. Each tool is allocated dedicated resources on the M-Lab platform to facilitate accurate measurements. All data collected via M-Lab is made available to the research community to allow researchers to build on a common pool of network measurement data. Measurement service represents the service provided by the synergy of M-lab along with researchers that develop tools on testing broadband connections; that is i. Measurement tools ii. Distributed server platform where tools are deployed/operating iii. Measurement logging system iv. Appropriate interface for measurement retrieval (single/group based).

SPEBS: A system that utilizes the measurement service with the unique target to retrieve, group and present results regarding the end user' s broadband connection service performance evaluation. SPEBS primarily enables monitoring of broadband connections.

End-User: An entity that makes use of the measurement service/SPEBS in order to test/monitor an arbitrary set of broadband connections e.g. a person that maintains a contract with an ISP (Internet Service Provide), an ISP that monitors the performance of a pool of broadband connections etc.

Internet Service Provider (ISP): A company that maintains broadband network infrastructure and provides internet connection services to customers. Primarily enables correlation of local loop/ end-user/ location.

Locator: Entity that maintains/provides registration information for networks and related location details. Registration information includes IP addresses and location details include geographical information that is associated by the ISPs with these IP addresses. Appropriate actor to take on this role is the National Regulator.

Geographic Information System (GIS): The GIS enables SPEBS to provide location aware services to the end-user e.g. present performance evaluation results on broadband connections per ISP, per area.