THE WEB APPLICATION OF THE SLAMETER TOOL

Liberios Vokorokos, Ján Juhár, Adrián Pekár, Peter Feciľak
Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics
Technical University of Košice, Letná 9, 042 00 Košice
E-mail: {liberios.vokorokos, jan.juhar, adrian.pekar, peter.fecilak}@tuke.sk

ABSTRACT

In this paper we discuss in general the role of monitoring tools in network traffic engineering and shaping; and current trends in the development of web applications, while our main focus is aimed at the top most layer of the SLAmeter tool, i.e. the web application; which provides the user interface for the tool. The purpose of the SLAmeter network traffic metering tool is to provide information about various aspects of computer network traffic. Since meaningful and configurable data visualization is an indispensable tool for achieving the goal of the monitoring; considering the shortcomings of the previous version; our main focus is aimed at the design and implementation of a modular architecture with a thick client for this web application which serve as the basis for further extension with modules for the evaluation of the collected data; and thus optimizing the process of network traffic monitoring and evaluation with the SLAmeter tool.

Keywords: *Web application, SLAmeter, architecture, client-side MVC, thick client, network traffic monitoring*

1. INTRODUCTION

Most of the monitoring systems share almost identical architecture and work on similar principles. Figure **??** illustrates a common architecture for monitoring systems. Data collection is ensured by continuous measurement of either (i) various data samples from the traffic or (ii) information reflecting the actual state of the inter-networking devices. The observed data are subsequently encapsulated into a special data structure and stored for further use.

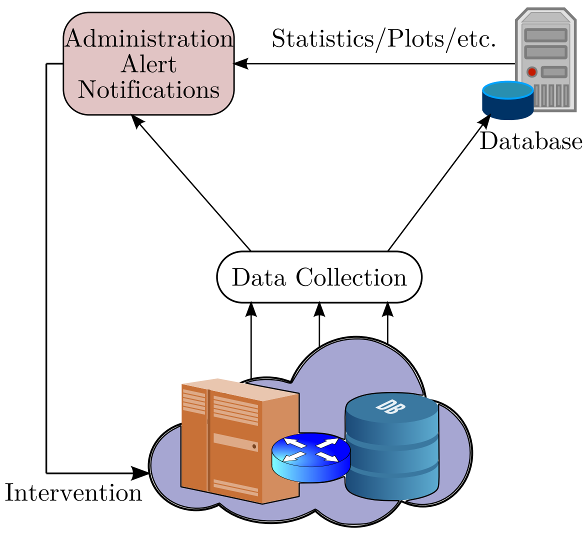


Fig. 1 Common architecture of network monitoring systems.

For now, let's say, this special data structure is a *metric*, however, other metric-like forms (e.g. flow record) also exist. The monitoring system continuously confronts the newly observed data with the ranges of the pre-agreed threshold and in case of any violation it raises an alert[1]. What happens next depends entirely on the implementation details, but basically there are two options: (i) in case of automated fault resolution, the system will take on the basis of certain computations some actions to resolve the potential or real threat or (ii) in case of manual fault resolution, by the analysis of the plots the operator or administrator has to draw a conclusion that should lead to a resolution. For a clearer explanation we shall break down this process into the following functional parts:

1. *Data collection* – Generally, by data collection we mean collection of all measurable information about the network, its components and traffic. However, in our case we will restrict this set only to network traffic properties. Collection of the data includes also process of extracting measured data with key properties into a metric which is subsequently sent either to a database or for further processing using a pre-agreed protocol.

2. *Data storage* – The data in the metrics are grouped and collected by their properties which basically serve as supplement information to the measured sample [**?**]. Relevant information[2] are retrieved from the metrics and summarized to yield a time series of values. Individual points of the series are subsequently evaluated and compared to admissible threshold range. If anomalous condition is detected, further actions are taken based on method of fault resolution.

3. *Visualization* – Network condition visualization is an effective way to deal with faults and errors. The evaluated values can be presented in many different ways to achieve the goal of monitoring, as well as they can be combined for maximizing the efficiency of the fault elimination process. The most useful information to take the accurate mitigating actions during alerts can be obtained exactly from them. Moreover, they also serve as reference point to check the correctness of the mitigating actions.

As we already stated, network monitoring is conditioned by data collection. The way how the measured data are treated

[1]Note, that at the same time, the observed data can be used for adjusting the actual altering threshold as well.
[2]The relevance depends on the desired result or goals of the evaluation.

after their collection, usually depends on the requirements and implementation of the monitoring system. The process of transforming collected data to their visualization is most commonly noted as data analysis. As described in [?] we distinguish between three application areas for data analysis:

- Flow analysis and reporting – is the most basic functionality. It typically provides the functionalities of (i) browsing and filtering flow data, (ii) statistics overview and (iii) reporting and alerting.
- Anomaly detection – the gathered data may be used for analyzing which hosts have communicated (i.e. gathering various summaries, etc.) or for analyzing certain types of threats (i.e. analyzing and modeling network behavior) [?, ?, ?].
- Performance monitoring – aims at observing the status of the running services on the network. Most common metrics for this include Round-Trip-Time (RTT), delay, jitter, packet loss and bandwidth usage. Further on, post-processing the gathered data can show a set of metrics per target service which can be subsequently used for the verification of Service-Level Agreement (SLA) compliance. In addition, performance monitoring can also reveal network events and their impact on end-user experience.

Since monitoring and reviewing the state of the network are critical tasks, the interface of the monitoring tool has a high importance for the user. It goes without saying that any potential insights on the network behavior that could by gained from the collected metrics – if not accessible through the user interface – are virtually non-existent from the user's perspective. Thus, meaningful and configurable data visualization is an indispensable tool for achieving the goal of the monitoring.

In this paper we present a new modular architecture for the web-based user interface of the SLAmeter network monitoring tool. This architecture is built on the previous work regarding the user interface of the SLAmeter tool, however, it significantly enhances its usability and responsiveness, while it also enables the integration of data visualization based on the stored (historical) as well as real-time network metrics' collections.

2. MOTIVATION

The SLAmeter tool [?] is aimed at metering and evaluation of different parameters of computer network traffic for the purpose of determining the level of compliance with the SLA. It uses IP Flow Information Export [?, ?] protocol for network traffic monitoring, that is focused on IP flows (conversations) between network devices. SLAmeter is developed by the MONICA research group in the Computer Network Laboratory of the Technical University of Košice.

As depicted in Figure ??, the SLAmeter tool consists of the following main components:

- *Exporter*, that monitors the network and generates IPFIX messages;
- *Collector*, for collecting IPFIX messages in a database;
- *Database*, for persisting collected data;
- *Evaluator*, that evaluates the data, either persisted or collected in real-time through Analyzer-Collector Protocol.
- *Web application*, that provides a user interface for the tool.

User interface in the for of a web application was chosen due to the ease of access to such an application for its users. Similar trend toward the web applications can be observed for many different types of applications, as the Internet is becoming more accessible and the web technologies are evolving at a rapid pace. Further, our focus will be aimed at the web application of the SLAmeter tool.

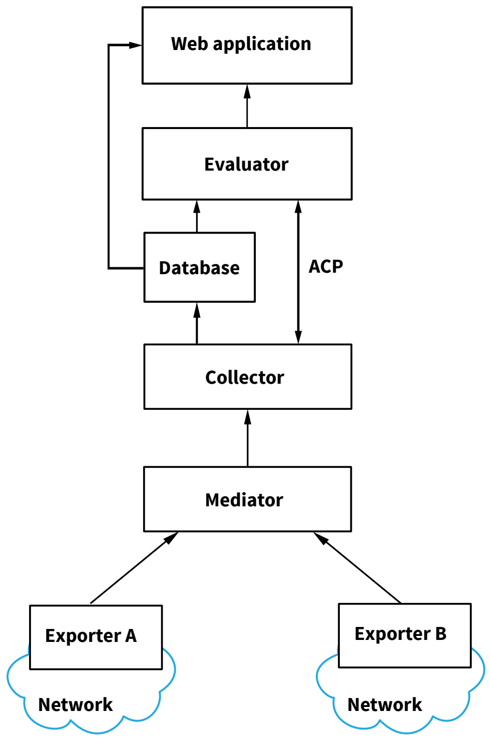


Fig. 2 The architecture of the SLAmeter tool.

The purpose of the web application is to provide a tool that will display the collected and evaluated data from the lower parts of the SLAmeter in a user-friendly way that will allow the users to understand and to interact with collected data through configurable visualizations. Its first version was created by the members of the MONICA group through numerous tasks and works. The modular architecture of the application consists of the server-side framework [?] that contains the modules, as depicted in Figure ??. Each module is responsible for displaying the data provided by the corresponding module in the Evaluator. The resulting interface is composed of the outputs of these modules and of additional templates.

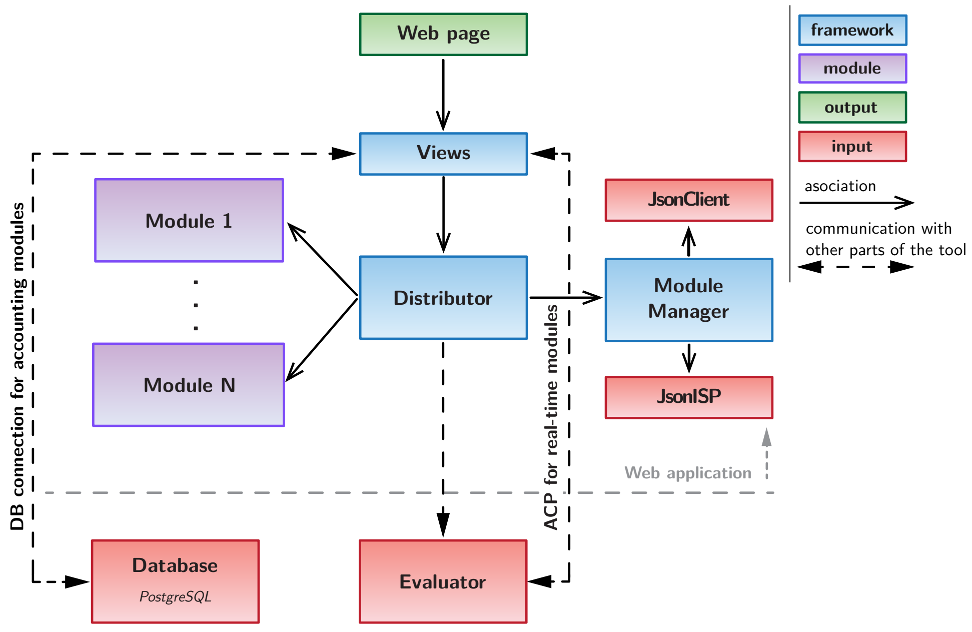


Fig. 3 The former architecture of the web server.

The usage of this web application showed that not all of the requirements on the modularity, extensibility, or even the functionality were met. There are modules created on the server and the charts used for data visualization in the user interface do provide some interactive features as they are rendered in the browser with a JavaScript charting library. However, the filtering criteria by which the data are selected and processed by the Evaluator can not be changed from the user interface – they are hard-coded in the modules of the web server. Moreover, these criteria do not capture the full filtering possibilities of the Evaluator.

Further problems in the web application are caused by the following process that creates a web page:

1. The *Distributor* requests filtering criteria from each of the modules (see Figure **??**).
2. The Distributor connects to the Evaluator and hands over the collected requests.
3. The data retrieved from the Evaluator are subsequently distributed back to the modules.
4. The modules produce the HTML fragments, including the JavaScript code for dynamic chart generation in the user's web browser.
5. These fragments are joined to form a full HTML page that is sent as a response to the user's original request.

The described process emphasizes (i) the need to fully evaluate all of the modules before anything can be sent to the browser and (ii) a strong dependency on the Evaluator as the only possible source of the data. The former creates a long loading time of the web pages due to the computationally intensive evaluation process. Further, it does not allow the filtering criteria configuration to be implemented at module level, as it would require partial page updates. The latter already caused difficulties in the process of extending web application with new modules for accounting and real-time metering [**?**]. These modules required different data sources and their implementation had to bypass the designed communication model, as depicted in Figure **??** in case of the database and the ACP (Analyzer–Collector Protocol) connections [**?**]. Thus, implementation of the web server became ineffective and difficult to maintain.

All the stated shortcomings hinder further development of the SLAmeter tool and they presented a motivation for thorough redesign of the web application. Our main objective was to design and implement an architecture for a new web application, that will result in its better modularity with modules of different kind, that will provide interactive data visualizations and solve the shortcomings of the previous version. We also intended to integrate some of the modules present in the previous version of the web interface into the new.

3. CURRENT TRENDS IN A WEB APPLICATION DEVELOPMENT

Web applications, i.e. the applications that are accessible through the web [**?**], are typical representatives of network-oriented applications. They are built according to the architectural style called *client–server*, which is by [**?**] the most common hierarchical style in this sort of applications. As further stated by [**?**], in this style server provides services and expects request for these services. Client that requires particular service sends the requests. It seems, however, that such strict separation of concerns has been for long time missing from the web applications, as we describe below.

3.1. Moving Towards the Thick Client

Originally, World-Wide Web presented a mean of accessing static, prepared documents, or pages. At that point, separation of concerns between client and server was achieved, as client's requests were denoted only by the URL of the page and responses of the server were stateless. All what the client had to do with received page was to display it. Such architecture is referred to as a *thin client architecture* [?, ?].

When web servers transitioned from static content to dynamic, URLs turned into inputs of applications that the web servers were running. Based on such input web server have been able to generate document for the client. From the client's perspective, though, nothing have changed. In order to produce per-user specific content, state was included in the form of *cookies* – tokens that could be passed back and forth between the client and the server to track the state of the application on the server side. But even with this "hack", user interaction was restricted to navigation through hypertext links. Even though this is sufficient for many (mostly statical, or heavy text-oriented) applications, no truly interactive application can be created this way, mainly because of (i) long latency in user interface reactions due to client–server communication, and (ii) the need to re-create whole web page on the server and client side after each received request and response, respectively.

In the context of World-Wide Web, where web browsers are dominant client applications, inclusion of JavaScript language brought the desired interactivity [?]. XML-HttpRequest (XHR) API that was later introduced into the JavaScript language made possible to move communication with the server to the background, so that it was no longer interrupting flow of the user's activity [?]. A suite of technologies that could be utilized in conjunction with XHR became known as AJAX and it radically changed the way web applications were built. However, it also introduced a few problems with regard to the originally intended separation of concerns in the client–server architecture.

As a consequence of the partial page updates performed in the background, a client requires some application logic that will incorporate updated portions of the interface back to the displayed page. This is possible either directly with JavaScript API for manipulating Document Object Model of the page, or with specialized libraries that greatly simplify such manipulations, e.g., jQuery or Prototype. However, this divides construction of the pages between the server and the client. Furthermore, by [?], often it is not clear which part of the page construction should be implemented where. Situation is further complicated when we take into account non-HTTP communication with the server, most notably through bi-directional WebSocket connection for real-time events.

According to [?], solution to these problems lies in moving all of the presentational logic to the client (web browser), where it can readily respond to user's actions, while data in their structured form can be interchanged between the two sides of the client–server architecture in the background. This can be achieved by implementing full Model–View–Controller framework on the client side – in other words, creating *thick client* where all updates to the user interface are done with JavaScript. The server then becomes responsible for providing *domain services* that are used by the client. This is similar to Service-Oriented Architectural style and therefore is referred to by authors of [?] as Service-Oriented Front-End Architecture (SOFEA).

Over the last few years, there have been a gradual shift towards the thick clients in the area of web applications. Increasing amount of page content is being retrieved through AJAX or WebSocket and web servers provide an API for accessing their content. This fits the service-oriented architecture concept.

3.2. The Technology

As mentioned above, a thick clients should implement full MVC architecture to provide separated presentational layer of a web application. To make development of such clients more effective, a number of libraries or frameworks were introduced. Generally, these are based on the mentioned three-layer architecture common in user interface design. However, they differ in ways the MVC is used, as well as in the scope of the problems they are trying to solve. Basically, there are three categories into which we can divide features provided by most of these tools: (i) the view layer of the application, (ii) the architectural primitives available to the programmer, and (iii) the way to navigate the application.

One of the oldest tools for MVC architecture in the browser is Backbone[3] library that provides basic primitives to achieve model, view, and controller layer separation in the application. Most of the work, however, is left to the programmer. Another popular tool is Knockout[4] that focuses on data binding between model and view through the intermediary *view-model* layer. Features are yet again limited and binding notation in the view is very verbose and driven by custom HTML attributes. Neither of these provide direct tools for navigating the web application and are therefore more suited to enhance pages generated by the server.

Arguably, the most advanced tools are represented by AngularJS[5] and Ember[6] projects. These provide the most complete features of MVC architecture, but are nonetheless different in their intended usage. AngularJS is more like a toolset for building a framework most suited for specific application development, while Ember strives to provide ready-to-use solutions to most of the common problems in the so-called single-page web application development.

AngularJS can manage the whole presentational layer of the web application, or it can be used to enhance existing, server-rendered page with advanced interactive features. It provides templates (views) based on HTML, data bindings between models and templates, and controllers to drive application logic with support for dependency injection. Application navigation is possible by third-party extension that enables switching the current view based on the URL.

[3]http://www.backbonejs.org
[4]http://www.knockoutjs.com
[5]http://angularjs.org
[6]http://emberjs.com

On the other side, Ember requires full control of the page it is used on, which it manages mostly through its router. The router acts as a state machine that, based on the current URL, builds a hierarchy of the views, wires them with the corresponding controllers and manages available model data. Most of this is build on defined naming conventions at the higher and dependency injection at the lower layers of the framework. Templating is provided by Handlebars library that adds syntax for dynamic data bindings and content generation to the HTML language.

4. RELATED WORK

We can find numerous flow data analysis applications that use web application as their user interface, both commercial and open-source. An overview of such applications from the network monitoring perspective can be found in [?]. In this section we focus on the web interfaces and technologies behind them.

Flow data analysis applications rely heavily on data visualizations. These have either textual form of tables or graphical form of various types of charts. Often there is a way to alter displayed visualizations, either by changing the visualization form (e.g., from textual to graphical), or by adjusting data selection used for visualization (e.g., filtering). In the modern web interfaces this implies the use of client-side scripting for interactive behavior, at least for in-background retrieval of pre-rendered content, if not directly for generating the visualizations from raw data. Open-source application ntop[7] uses jQuery with combination of additional libraries to create dynamic user interface with real-time data updates. However, other open-source tools are lacking in this area: WebView NetFlow Reporter or Stager renders all parts of their interfaces on the server, which reduces their interactivity.

On the other side, commercial tools have more advanced web user interfaces. SolarWinds' Network Traffic Analyzer uses, similarly to ntop, custom solution built from smaller libraries to provide dynamic interface. ManageEngine's NetFlow Analyzer goes all the way to the thick client and uses Ember framework for fully client-side driven web interface.

5. ARCHITECTURE OF THE WEB APPLICATION

The existing web application of the SLAmeter tool can be described as a server-rendered web application with a thin client as conceptually described in [?]. Although this application provides partial interactive features, such architecture is not particularly well suited for dynamic user interfaces. It requires the whole pages to be rendered at once, which limits interactivity, responsiveness and performance of data visualizations. In order to increase the performance and to resolve the detected problems, we introduced some changes that go with the current trend.

One way to increase performance of web application with heavy use of the data visualizations is to enable data updates on the single visualization level. This way, each module that provides data visualization can be evaluated independently. If partial page updates through XHR API was used in the case of our application, most of the data displayed on the page would be retrieved that way. For that reason, we will move full presentational layer to the client and use service-oriented web server. Besides avoiding the split of the presentation flow, such an architecture can provide better modularity as well as support for easier inclusion of real-time communication through WebSocket protocol, that is by [?] well suited for monitoring applications, because both XHR and WebSocket will transfer only data that will be subsequently visualized in the web browser.

5.1. Implementation considerations

To enable an effective development of the application, we decided to use some JavaScript framework. AngularJS and Ember, were thoroughly explored and tested in order to select the right one. As a result, Ember was chosen with regard to:

- the way it allows to define the hierarchical structure of views;
- a *router* which helps to manage the application's states and transitions between different view hierarchies;
- its computed properties and a generally richer API.

On the server side, Django framework was used, as in the former web application. Though, this time with the support of the *Django REST Framework* for creating the REST web services for the client–server communication. The Python library gevent-socketio was used for the integration of the WebSocket communication protocol with the server.

5.2. The modular architecture

The work on a new web application of the SLAmeter tool began with the design of the new modular architecture. This architecture should provide the base for various kinds of modules that could be easily integrated if provided with required application interface. For this reason, described architectures for the web server and the web client, while influenced by particular used technology, present general structure, for which the most building blocks are provided in our implemented framework.

On the server side, the *applications with modules* were designed as Django applications that will contain groups of similar modules. One such application in its general form with classes is depicted in Figure **??**. Rationale behind the depicted classes is as follows:

- The *data connectors* retrieve the data from data sources.
- The *modules* have responsibility for requesting data from their respective data sources with the use of the data connector. This functionality is made available for the client through either the REST web services or

[7]http://www.ntop.org

the WebSocket events. *Service views* are used to provide additional information about the modules, such as their name displayed in the user interface or the used filtering criteria. Thus, the module information and the data retrieval create required application interface of each module.

- The *application* groups modules by their type.
- The *application router* is used to generate regular expressions for REST service URLs registered with the application and its modules.
- The *serializers* are used for transformations between the internal data structures and the JSON format for the data interchange with the client.
- The *framework* consists of abstract or base classes that provide a basis for the concrete application implementation.

Base building blocks that were implemented are displayed as framework classes. In addition to the depicted elements, the *core* of the server side, designed to connect the services of all the *applications with the modules*, was implemented.

The architecture of the client side is shown in Figure **??**. This structure is built of the base components providing default implementation that can be easily extended. It contains routes, that manages views shown on the web page, with templates and controllers assigned to these views. The applications, corresponding to those on the server, are further split to sections to allow a better organization of the user interface. The section consists of three panels for dis-

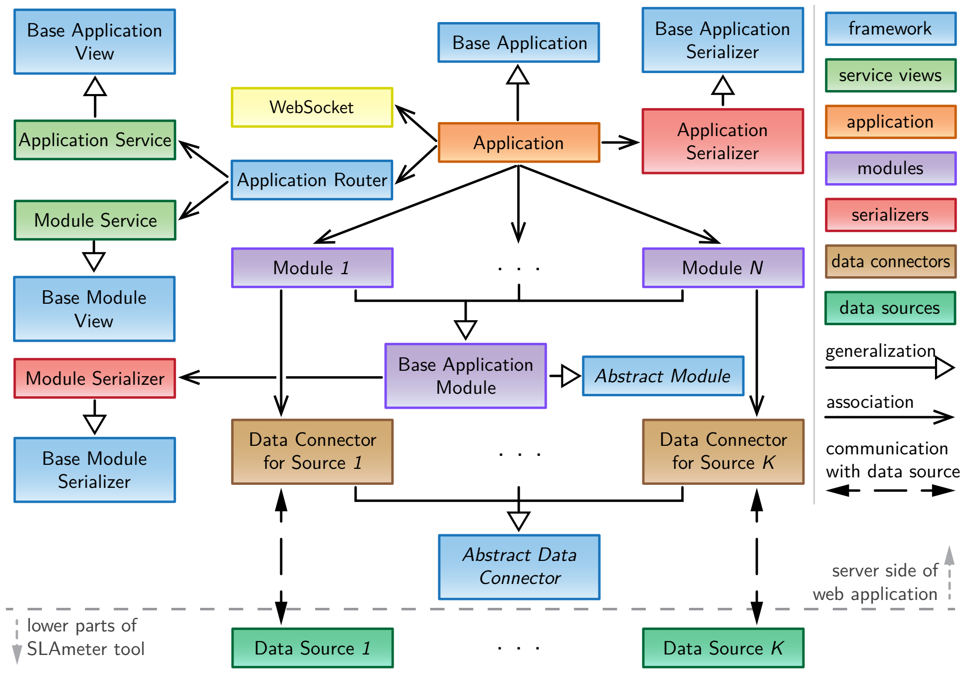


Fig. 4 Class diagram of the general application with the modules on the server.

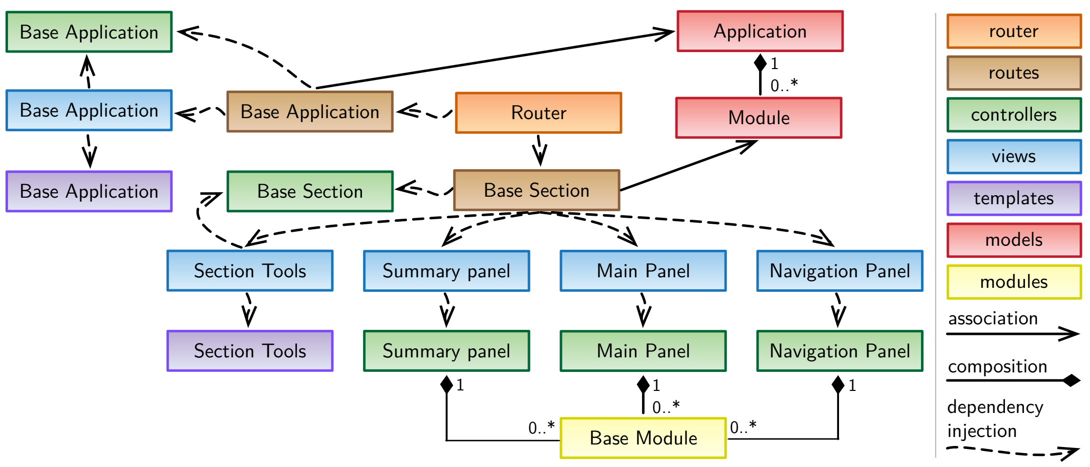


Fig. 5 Class diagram of the general application with the modules in the web client.

playing different kinds of modules – summary and navigational modules in the sidebars and larger main modules for tables and charts in the middle. The models contain data about modules and applications that are retrieved from the server through web services. The individual modules are built of a set of MVC components: (i) one view with a controller, (ii) the main layout template and (iii) partial templates for header, toolbar, and main body. These partial templates are designed to make even extensive adjustments to the base modules as simple as possible.

The described base MVC components are looked up during the run-time construction phase in such a way that allows them to be replaced with specific implementations. This builds upon Ember naming conventions that we have extended to support fallback lookup of dependent JavaScript modules. The lookup begins with the specific module name, continues to more general versions, and ends at the default module implementation in case when no other version is found. This created naming convention for the JavaScript modules allows the injection of customized implementation at the place where it is needed, while it requires no manual configuration.

5.3. The integration of the modules

With the new design and implementation of the architecture for the web application, the modules implemented for the previous version became unusable. Thus, we also devoted time to the integration of the selected modules into the new application.

Modules that were integrated are those that provide statistical and analytical information regarding the network traffic. Some of the implemented modules are displayed in Figure **??**.

The modules were placed within the first *application with the modules*, implementation of which was based on the architectural design described earlier, while the framework classes were used on the server and the base modules on the client side. However, since the new architecture provided several features beyond possibilities of the former one, this did not mean just the re-implementation of the old code. The most notable new feature was the individual module evaluation, which was used to implement a user-configurable filtering criteria on the module level. Figure **??** shows the module *Bandwidth History* with activated time filtering. It is also possible to filter network traffic on the basis of the IP addresses and ports, as these represent filtering capabilities of the Evaluator. Filter parameters can be specified separately for the source and destination, and to select all the traffic flowing to and from a specific address or port, non-directional filtering can be used.

6. DISCUSSION

The main motivations behind this work were (i) to resolve issues present in the original version of the web application and (ii) to enhance usability of the SLAmeter metering tool. To evaluate the implemented solution, we subjected it to use under laboratory conditions, where the web application was used by other members of MONICA group to visualize output of those layers of the SLAmeter tool they were working on.

From the point of interface responsiveness, users reported the load time was significantly decreased compared to the original version they were also familiar with. This can be associated with the used service-oriented architecture with the thick client, where with initial page load the full interface is retrieved in form of static files (and these can be subsequently cached by the browser). Next, the modules begin to evaluate and the retrieved data are immediately displayed. Moreover, the users are informed about the ongoing evaluations and therefore the web application seems to be more responsive.

The ability to easily filter the collected data is essential to the network monitoring tool and this functionality was finally included. Positively was perceived also possibility to re-evaluate individual modules.

The better modularization of the architecture contributed to the ability to adapt the deployed SLAmeter tool to increased load. We could deploy multiple Evaluator instances with intent to distribute the load of the module evaluation. By configuring modules on the web server to use different data sources, which was impossible in original centralized architecture, we could thus achieve better evaluation performance.

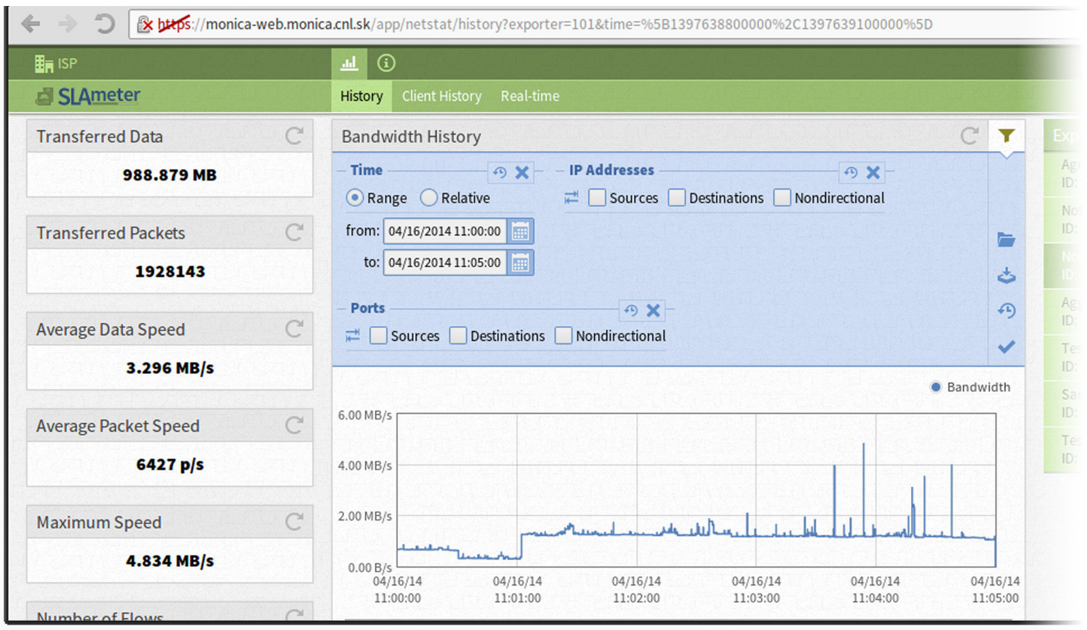
In conclusion, although the real-time modules still have to be implemented, we conducted tests of the client–server communication through a WebSocket protocol on implemented modules to ensure this feature is also ready when needed.

7. CONCLUSION AND FUTURE WORK

In this paper we reviewed some current trends in web applications of the monitoring tools and presented details regarding the new design and implementation of the modular web application of the SLAmeter tool.

The intended objective was achieved. The new architecture does not show weaknesses of the previous version. Its response time was greatly reduced by allowing for evaluation of the modules individually. The better modularization and separation of the concerns will also help when adding a new modules to the application. Moreover, WebSocket support will allow for the implementation of real-time evaluating modules. In addition, the selected modules were also integrated into to the new architecture. These modules extend the functionality with the support of configurable filtering criteria (these are essential to exploit the potential of the metering tool). We believe that thanks to those changes the interface of the SLAmeter can compete with similar tools.

Future work will be aimed at the integration of more modules, either those that can be migrated from the former web application, or entirely new ones. Again, with the new architecture, their functionality of migrated modules can be expanded significantly. As for the new modules that will further extend features of the SLAmeter tool, there are plans for accounting and real-time modules, and also modules for threat monitoring.


~~https~~://monica-web.monica.cnl.sk/app/netstat/history?exporter=101&time=%5B1397638800000%2C1397639100000%5D

Transferred Data: 988.879 MB

Transferred Packets: 1928143

Average Data Speed: 3.296 MB/s

Average Packet Speed: 6427 p/s

Maximum Speed: 4.834 MB/s

Bandwidth History

Time: [✓ Range] Relative

from: 04/16/2014 11:00:00

to: 04/16/2014 11:05:00

IP Addresses: Sources, Destinations, Nondirectional

Ports: Sources, Destinations, Nondirectional

Fig. 6 The web application of the SLAmeter tool.

ACKNOWLEDGEMENT

This work was supported by the Slovak Research and Development Agency under the contract No. APVV-0008-10 and project KEGA 008TUKE-4/2013: Microlearning environment for education of information security specialists.

REFERENCES

[1] ANTL, M.: Framework for the Evaluator and Web Interface of the SLA Meter Tool. Master's thesis, Technical University of Košice, Faculty of Electrical Engineering and Informatics (2012)

[2] CHOVANEC, M., CHOVANCOVÁ, E., DUFALA, M.: Dids based on hybrid detection. In: Proceedings of the 12th IEEE International Conference on Emerging eLearning Technologies and Applications. pp. 79–83. ICETA '14, IEEE (2014)

[3] CLAISE, B., TRAMMELL, B.: Information model for ip flow information export (ipfix). RFC 7012 (September 2013)

[4] CLAISE, B., TRAMMELL, B., AITKEN, P.: Specification of the ip flow information export (ipfix) protocol for the exchange of flow information. RFC 7011 (September 2013)

[5] DANKOVÁ, E., ÁDÁM, N., JAKUBČO, P.: An anomaly-based intrusion detection system. In: In Proceedings of Electrical Engineering and Informatics 2, (EEI'II). pp. 260–264 (2011)

[6] DONG, S., CHENG, C., ZHOU, Y.: Research on ajax technology application in web development. In: E - Business and E -Government (ICEE), 2011 International Conference on. pp. 1–3 (May 2011)

[7] ENNERT, M., CHOVANCOVÁ, E., DUDLÁKOVÁ, Z.: Testing of ids model using several intrusion detection tools. Journal of Applied Mathematics and Computational Mechanics 14(1), 55–62 (2015)

[8] FIELDING, R.T.: Architectural Styles and the Design of Network-based Software Architectures. Ph.D. thesis (2000)

[9] FRATERNALI, P., ROSSI, G., SÃĄNCHEZ-FIGUEROA, F.: Rich internet applications. Internet Computing, IEEE 14(3), 9–12 (May 2010)

[10] HOFSTEDE, R., CELEDA, P., TRAMMELL, B., DRAGO, I., SADRE, R., SPEROTTO, A., PRAS, A.: Flow monitoring explained: From packet capture to data analysis with netflow and ipfix. Communications Surveys Tutorials, IEEE 16(4), 2037–2064 (Fourthquarter 2014)

[11] HUSOVSKÝ, M.: Network Services Usage Accounting in th SLAmeter Tool. Bachelor's thesis, Technical University of Košice, Faculty of Electrical Engineering and Informatics (2013)

[12] JAZAYERI, M.: Some trends in web application development. In: 29th International Conference on Software Engineering: Future of Software Engineering. pp. 199–213. ICSE:FOSE '07 (2007)

[13] LEFF, A., RAYFIELD, J.: Web-application development using the model/view/controller design pattern. In: Proceedings of the 5th IEEE International Conference on Enterprise Distributed Object Computing. pp. 118–127. EDOC '01, IEEE (2001)

[14] LIGUS, S.: Effective Monitoring and Alerting: For Web Operations. O'Reilly Media (2012)

[15] PEKÁR, A., FECIL'AK, P., MICHALKO, M., GIERTL, J., RÉVÉS, M.: Slameter – the evaluator of network traffic parameters. In: Proceedings of the 10th IEEE International Conference on Emerging eLearning Technologies and Applications. pp. 291–295. ICETA '12, IEEE (2012)

[16] PEKÁR, A., RÉVÉS, M., GIERTL, J., FECIL'AK, P.: Overview and insight into the monica research group. Central European Journal of Computer Science 2, 331–343 (2012)

[17] PRASAD, G., TANEJA, R., TODANKAR, V.: Life above the service tier (2007)

[18] PURANIK, D., FEIOCK, D., HILL, J.: Real-time monitoring using ajax and websockets. In: Engineering of Computer Based Systems (ECBS), 2013 20th IEEE International Conference and Workshops on the. pp. 110–118 (April 2013)

Received November 12, 2014, accepted March 16, 2015

BIOGRAPHIES

Liberios Vokorokos (prof., Ing., PhD.) was born on the 17. November 1966 in Greece. In 1991 he graduated (MSc.) with honors at the Department of Computers and Informatics of the Faculty of Electrical Engineering and Informatics at the Technical University of Košice. He defended his PhD. in the field of programming device and systems in 2000; his thesis title was "Diagnosis of compound systems using the Data Flow applications". He was appointed professor for Computers Science and Informatics in 2005. Since 1995 he is working as an educationist at the Department of Computers and Informatics. His scientific research focuses on parallel computers of the Data Flow type. He also investigates the questions related to the diagnostics of complex systems. He is the dean of the Faculty of Electrical Engineering and Informatics of the Technical University of Košice. His other professional interests include the membership in the Advisory Committee for Informatization at the faculty and the Advisory Board for the Development and Informatization at the Technical University of Košice.

Ján Juhár was born in 1989. In 2014 he graduated (MSc.) with distinction at the Department of Computers and Informatics of the Faculty of Electrical Engineering and Informatics at the Technical University of Košice. Currently, he is a PhD student with the focus on software concerns.

Adrián Pekár was born in 1986. In 2011 he graduated (MSc.) with distinction at the Department of Computers and Informatics of the Faculty of Electrical Engineering and Informatics at the Technical University of Košice. Since 2011 his scientific research is focusing on the optimization of measurement platforms based on the IPFIX protocol. He defended his PhD in the field of network traffic characteristics' measurements and monitoring in 2014. Since that, his scientific research was extended to also investigate the questions related to the monitoring and virtualization of could networks.

Peter Feciľak was born in 1983. In 2006 he graduated (MSc.) at Department of Computers and Informatics at Faculty of Electrical Engineering and Informatics, Technical University of Košice. In 2009, he finished his PhD studies at the same department with the focus on optimization of computer networks. Currently, he is working as employee of DCI, FEI, Technical University of Košice. His current teaching and research interests are computer networks, network monitoring, quality of services and smart energy systems.