Simplification of the Real-Time Network Traffic Monitoring

Adrián PEKÁR, Juraj GIERTL, Martin RÉVÉS, Peter FECIĽAK

Department of Computers and Informatics, FEI TU of Košice, Slovak Republic

adrian.pekar@gmail.com, juraj.giertl@tuke.sk, martin.reves@tuke.sk,
peter.fecilak@tuke.sk

Abstract – **This paper deals with real-time monitoring of network traffic, it introduces the Analyzer-Collector Protocol and its Application Programmable Interface. Particular attention is given to the BasicMeter measuring tool and the explanation, why are tools like BasicMeter so important in modern networks.**

Keywords – **real-time monitoring, network traffic, QoS, API, ACP, BasicMeter, ACPapi, exporter, collector, analyzer**

I. Introduction

Today's computer networks, which are based on the best-effort traffic model, are designed to transmit video, sound and data (or some combination of these traffic types) at the same time. This kind of transmission is provided by converged networks, which instead of setting own separate links for data and voice, make do with only one converged link. However, in these networks, to get the desired quality of the applications sensible to latency (e.g. Voice over Internet Protocol - VoIP, Video on Demand - VoD), we need to ensure a given level of Quality of Services (QoS).

The phrase QoS is binded to a set of parameters (packet rate, packet loss, one way delay, jitter, round-trip time, etc.), which are used to define the characteristics of computer network traffic. With measuring these parameters, we can ensure the functionality of the above mentioned real-time applications, the fulfillment of conditions specified in the Service Level Agreement (SLA), prevent network attacks, find out dominate traffic sources, etc. Therefore, real-time monitoring of QoS parameters has a significant role in the administration of converged networks and their services.

By means of Analyzer-Collector Protocol [1], the BasicMeter tool [2] was designed for measuring network parameters and their real-time computing. While the implementation of real-time data collection and computation is in some cases really problematic, there was a need for an Application Programmable Interface for servicing the Analyzer-Collector Protocol (ACPapi). ACPapi was designed to simplify the work of the programmers during the development of real-time data dependent applications. Its main purpose is to serve the ACP, which gives opportunities for communication between the parts of the BM tool, whereby the programmers do not have to care for the implementation details of this communication.

In the following sections an alternative method for real-time network traffic monitoring is presented, starting with a simple introduction of the BasicMeter tool; following with a detailed description of the Analyzer-Collector Protocol and its communication principles; up to the presentation of the ACPapi and its basic implementation details.

II. The BasicMeter Measuring Tool

The architecture of the tool was developed in the Computer Network Laboratory at the Technical University of Kosice within three sub-projects:

- BEEM – Basicmter Exporting and Measuring process
- JXColl – Java XML Collector
- BM Analyzer – BasicMeter Analyzer

Regarding to the roles of these projects (BEEM, JXColl, BM Analyzer), in the following they will be specified as the exporter, collector and analyzer.

The concept of the BasicMeter tool, as described in Figure 1, is in conformance to the IPFIX architecture [3].

Analyzer is a front-end interface, which cares about both, the visualization of the computed data and the control of the architecture's lower parts. However, the analyzer itself is not a part

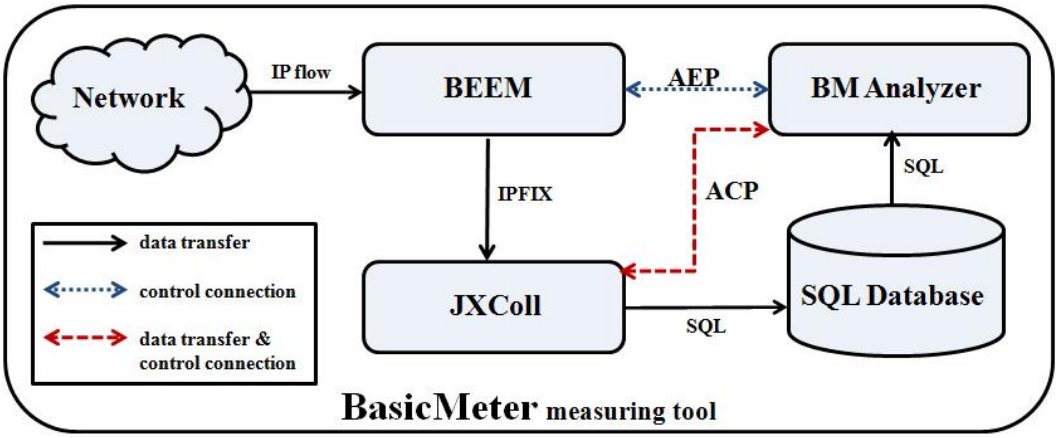


Fig. 1: The architecture of the BasicMeter measuring tool

of the IPFIX architecture, therefore we had to draft and implement our own network protocols for servicing the communication between the analyzer and the other parts of the measuring tool. These protocols are the Analyzer-Collector Protocol (ACP) and the Analyzer-Exporter Protocol (AEP).

AEP was proposed to control the connection between the analyzer and the exporter. With AEP, it is possible to operate the behavior of the export process directly from the customers' application.

In the primary concept of the tool, the measured data should be stored in a database for later analyzes. However, in the case of real-time data computation this method is absolutely inapt. With the growth of data in the database, the processing of the queries is becoming more and more time-consuming. For this reason there was a need for choosing an other way to gain data about network traffic without the ineffective access to the database. Ad hoc was developed the ACP, which allows effective computation of data between the collector and the analyzer. The collector is permanently storing data in the database and if needed, it is simultaneously sending exactly the same data (or a part of it) to the analyzer by the means of ACP.

III. Analyzer-Collector Protocol

ACP is a binary, application layer protocol. TCP is used for communication, but it could be easily replaced with any other reliable, connection-oriented protocol. The communication is bi-directional and works on the base of client-server model, while the client side is represented by analyzer and the server side is represented by the collector. As it was mentioned in the previous section, the primary function of the protocol is to gain network traffic data with avoiding the ineffective database queries. To achieve this, besides data, control messages are also transmitted, which are dedicated for format and accuracy checkout of received data by the analyzer.

Communication Principle

The communication is based on sending queries by the analyzer and sending replies with data by the collector. This type of communication could be easily illustrated by a Finite State Machine (FSM) of both sides. The particular communication phases are represented with machine's states at which every message has its own unique ID.

Figure 2a and 2b depict the state transitions of Analyzer-Collector Protocol. State graph in Figure 2a is representing a Mealy type Finite State Machine. Figure 2b is describing a Moore Machine. For the reason that the outputs of the Moore Machine are representing the particular activities of the Analyzer (e.g. receiving data), they are not present in Figure 2b. For easier understanding of state transitions from old states to new states, the messages that invoke these transitions are also included in Table 1 and 2.

ACP is forwarding:

- data (0)
- control messages (1)

Types of the control messages:

- (A) – authentication
- (0) – setting template
- (1) – setting filter

- (2) – suspending data transmission
- (3) – resuming data transmission
- (4) – setting data transmission type
- (5) – acknowledging received data

States of the ACP:

- S_1 – start
- S_2 – waiting
- S_3 – receiving
- S_4 – transmitting
- S_5 – suspended
- S_6 – stop

Messages sent by analyzer:

- A_0 (A_1) – incorrect (correct) authentication data
- 0_0 (0_1) – incorrect (correct) template
- 1_0 (1_1) – incorrect (correct) filter
- 2_0 (2_1) – data transmission unsuccessfully (successfully) suspended
- 3_0 (3_1) – data transmission unsuccessfully (successfully) resumed
- 4_0 (4_1) – not supported (accepted) data transmission type
- 5 – acknowledging the received data

Messages sent by collector:

- 0 – unsuccessful authentication in state S_1 (after established connection), the requested template was rejected in state S_1
- 1 – successful authentication in state S_2 (after established connection), the requested template was accepted in state S_2
- 10 (11) – filter not accepted (accepted)
- 20 (21) – suspension of data transmission was rejected (accepted)
- 30 (31) – resume of data transmission was rejected (accepted)
- 40 (41) – not supported data transmission type (data transmission type is supported and also accepted)

Table 1: State transitions - collector side

OldState	IncomMsgID	OutgoMsgID	NewState
S_1	A_0	0	S_6
	A_1	1	S_2
S_2	0_0	0	S_2
	0_1	1	S_4
S_4	0_0	0	S_4
	0_1	1	S_4
	1_0	10	S_4
	1_1	11	S_4
	2_0	20	S_4
	2_1	21	S_5
	4_0	40	S_4
	4_1	41	S_4
	5	-	S_4
S_5	3_0	30	S_5
	3_1	31	S_4
S_6	-	-	-

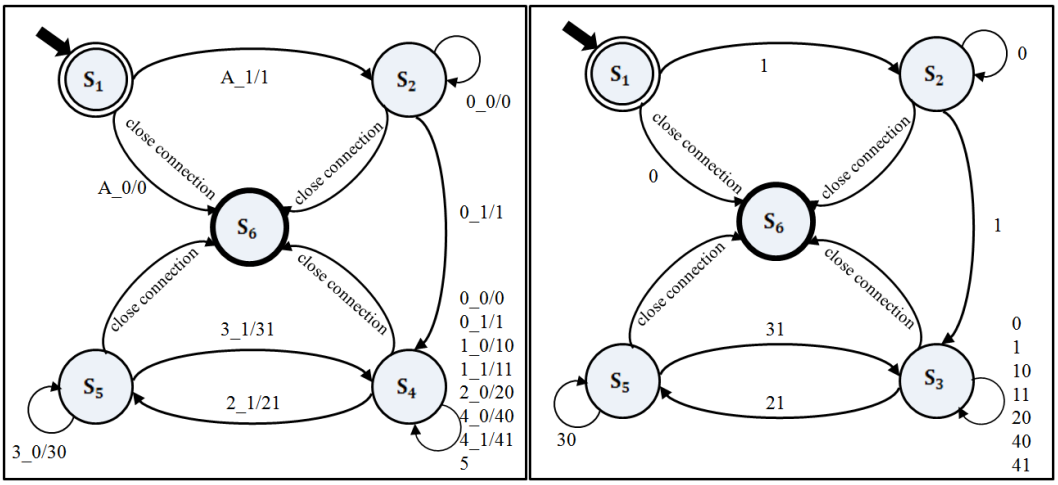
Table 2: State transitions - analyzer side

OldState	OutgoMsgID	IncomMsgID	NewState
S_1	A_0	0	S_6
	A_1	1	S_2
S_2	0_0	0	S_2
	0_1	1	S_3
S_3	0_0	0	S_3
	0_1	1	S_3
	1_0	10	S_3
	1_1	11	S_3
	2_0	20	S_3
	2_1	21	S_5
	4_0	40	S_3
	4_1	41	S_3
	5	-	S_3
S_5	3_0	30	S_5
	3_1	31	S_3
S_6	-	-	-

Description of the Communication

The FSM graphs (Figure 2a and 2b) and the tables (Table 1 and 2) clearly describe the communication via ACP's control messages between the collector and the analyzer. Control messages sent by collector act only as replies to requests generated by the analyzer. However data transmission is running through the same connection, from one point of view it represents the collector's special autonomous activity, which is initiated by the receive of IP flow information from the exporter(s). Regarding to desired simplicity and efficiency of the protocol, the header of the control messages is differing from the header of the data in one byte (value 1 for the control message and value 0 for data).

At the initial state, the collector is awaiting the analyzer's connection at a pre-agreed port (default value for collector is 2138). If there are no problems during the connection establishment, both sides traverse to their S_1 states. Before the bi-directional communication can be opened, the analyzer has to send accurate authentication information to the collector. Therefore as soon as the connection is established, the analyzer sends its login and MD5 password ciphers to the



(a) Collector side (b) Analyzer side

Fig. 2: State transitions

collector. Successful authentication brings both sides to their S_2 states, but if the authentication is unsuccessful, the collector terminates the connection and the analyzer have to reconnect for an other try.

In S_2 state, the collector is awaiting the analyzer's setting of the template message (message 0), which informs the collector about the desired format of the data. At the current state of the ACP, templates are sent as arrays containing the IDs of the information elements, but in the future we count with replacing the arrays with a binary description of these elements. After receiving the template the collector replies with its acceptance (message 1) or rejection (message 0). If the template was rejected, the analyzer has to send another template. During this phase, unless the collector accepts the template and replies with accept message, the communication is suspended. If the template is accepted, the collector traverses to its S_4 state and starts with the data transmission in the format of the accepted template. On the other hand, after the analyzer receives the template accepted message, it immediately traverses to its S_3 state and starts to receive the data from the collector. In this phase the template could be changed at any time, and if it is accepted by the collector, the data would be sent in the format of the new template. Real data, which were received/sent by the analyzer/collector, besides control messages are sent only in states S_3 and S_4.

By default, collector exports all the traffic information obtained from the exporter(s). However, in a case of need, by setting the filter (message 1) it is also possible to receive only specific information. This is an optional step which is accepted by the collector only in its S_4 state, while only one filter can be set in a given time. Current flow filtering is supported upon the next criteria:

- IP address of the measuring point.
  - Format of the criterion: (A.B.C.D/n), where $0 \leq n \leq 32$;
- Source and destination IP address/mask – with the combination of IP address and mask it is possible to specify not only one specific host, but also a whole subnet. It is also possible to specify more subnets or hosts.
  - Format of the criterion: (A.B.C.D/n; A.B.C.D/n - A.B.C.D/n; A.B.C.D/n , A.B.C.D/n), where $0 \leq n \leq 32$;
- Source and destination port – this criterion is allowing to specify upon the transport layer port number also a specific type of network traffic. It is also possible to set more values and range of values.
  - Format of the criterion: (N; N - N; N, N), where N can be a well known port, registered port and dynamic or private port;
- Protocol – currently the recognition is supported by numbers and like in the case of port criterion, it is possible to set random combination of values and range of values.
  - Format of the criterion: (N; N - N; N, N), where N can be any of the known protocol numbers;

The collector is using message 11 to confirm the accepted filter, and message 10 to inform about the filters rejection. The filter can be canceled or replaced with empty filter rule in the collectors S_4 state.

Control message with ID 4 is used to set the type of the data transmission, which brings more customizability and convenience for the developers using the ACPapi to communicate with the collector. Currently ACP supports two types of data transmission:

- sending the data by collector in N-tuples (message 1), where N is the number of the information elements in obtained template
- sending the data by collector one-by-one (message 2)

By default, the collector sends the data in N-tuples, ergo during one cycle of data reception the analyzer receives N values. This type of data transmission can be changed optionally in the collector's S_4 state. With second type of data transmission the analyzer receives one value during one data reception cycle.

There are another two control messages, which are dedicated for suspending and resuming the data transmission. In S_4 state the collector receives the data transmission suspension message (message 2). The receive of this message causes that the collector sends every data from the currently processed packet, suspends the data transmission and replies with message ID 21, which informs the analyzer about the transfer suspension. Both sides of the communication traverse to their S_5 states. From here on, the analyzer won't receive any data. The only method to continue with data transmission is to send the collector the data transfer resume message (message 3), which brings both sides to their previous states. After this message, the collector replies with acceptance message (ID 31) and starts to transmit data from the actual packet received from the exporter(s). Before this message, the analyzer will not continue with receiving data. In general, there is no reason for the collector to discard the transfer suspension (message 20) or resume (message 30) request.

The last control message is destined for synchronizing the data sending/receive by the collector/analyzer. When the collector sends the data, it awaits acceptance from the other side. Unless this acceptance is obtained, the collector will not send any other data. With this method we can prevent receiving inappropriate data (from experience, without this control message the collector sent in some cases concatenated values, which could easily led to data computation errors).

IV. Application Programmable Interface for Servicing the Analyzer-Collector Protocol (ACPapi)

The language chosen was the Java language, which main advantages are in modularity, stability, perspicuity of the source codes, ability of the documentation auto-generation, etc. Except these advantages, Java was chosen also for the reason that the collector was written in Java too. Some of the methods will be used by both parts of the communication, hence of that the implementation in the same language is more preferable to the implementation in different languages. As mentioned before, the ACPapi has to be usable in the case, when the programmers do not want to waste time with designing and implementing their own classes or methods for the communication based on ACP. For this reason, the whole ACPapi was designed as an easy to understand, implement and use API, besides during its creation was also given a tone to its exact documentation and to the annotation of the source code.

An API can be written by means of classes or interfaces. According to [5] the most profound feature of Java interfaces is multiple inheritance. With multiple inheritance, you need just one object to implement an unlimited number of interfaces from an API. In the case of classes, you would need to create one subclass for each API class, and if these classes are related to one another, join their instances by means of delegation. This can significantly increase the amount of occupied memory.

On the basis of this knowledge the ACPapi was written by the means of interfaces, which main advantage expresses in the performance of the final application using ACPapi. On the contrary, with interfaces we also have to count with adding new methods only by filling in the source code.

Because of the above mentioned dual-usability of some methods, the ACPapi was divided into two packages:

- ACPapi package – contains the API interface itself with declarations of the methods, classes with definitions of the methods and other classes
- commonly used package – contains commonly used classes of the analyzer and the collector (classes for working with templates and filters)

Functions defined by Analyzer-Collector Protocol (sending authentication data, templates, filters, etc.) are named upon their functionality. These functions with short description (detailed description could be found in the *Communication Description* subsection) are the following:

- getMd5Digest – method for cipher login and password data
- connectToCollector – method for establishing the connection with the collector
- sendTemplate – method for setting the desired format of the received data

- sendFilter – method for setting filter rules
- SendPause,sendUnPause – methods for suspending or resuming the data transmission
- readCollectorAnswers – method for computing the data sent by the collector
- quit – method for correct termination of the ACP process
- sendTransferType – method for setting the type of the data transmission

Main function of the ACPapi is its data providing for almost real-time computation. The word 'almost' points to the fact, that receiving data by ACPapi brings a certain delay with itself, which results from the data computation by the exporter, their transmission over the network, further computation by the collector and sending them by ACP. Despite of that, from the point of real-time monitoring of network traffic, this delay is absolutely affordable.

Obtaining the data with ACPapi could be implemented in some ways. One method could be storing the data in some data structure (e.g. arrays). Another method could be reading the incoming messages and data in a cycle. Even though that the first method is more comfortable, from the point of real-time data computation the second method was chosen. By reason of proper customizability this method is not part of the ACPapi and is provided only as a design pattern (the source code of the design pattern could be found at the API's website - http://wiki.cnl.sk/Monica/ACPapi). Of course, the data could be obtained by other methods, but all of them need a thorough study of the ACP and its API.

Forasmuch as the protocol is defined for communication of both sides, it has to be supported by the collector too. However, a more detailed description of ACP support by collector is out of the scope of this paper. In a case of need, one can find more information about this on the projects website.

V. Conclusion

Actual researches and developments on the field of monitoring tools are leading to the improvement of their ability of measuring the main characteristics of network traffic. Besides the main effort is put on the maximization of their real-time data monitoring ability. An important feature of these tools is also their flexibility and ability of supporting a wide scale of complex applications. The BasicMeter measuring tool is focusing to the fulfillment of these requirements, which above all is served for measuring QoS parameters.

In the previous sections the main concept of the BasicMeter tool and its method of getting data for (almost) real-time computation by ACPapi was presented. The Application Programmable Interface for servicing the Analyzer-Collector Protocol fulfilled its goals. Its implementation brought its awaited results. Thanks to the proposed design pattern and some improvements to the ACP (setting transfer type or acknowledging the received data) the ACPapi gives reliable, easy to use and implement application interface for the developers of the BasicMeter measuring tool.

Future work should be aimed at the replacement of the actual design pattern of ACPapi and to the optimization of the templates by replacing the arrays with a common binary description of the information elements.

Acknowledgment

This work is the result of the project implementation: Center of Information and Communication Technologies for Knowledge Systems (ITMS project code: 26220120030) supported by the Research & Development Operational Program funded by the ERDF.

References

[1] F. Jakab, R. Jakab, M. Kaščák, and J. Giertl, "Improving Efficiency and Manageability in IPFIX Network Monitoring Platform," Proc. of the 6th International Network Conference, INC 2006, Plymouth, UK, 11.-14.7.2006, Plymouth, University of Plymouth, 2006, 6th., pp. 81-88, ISBN 1-84102-157-1.

[2] F. Jakab, Ľ. Koščo, M. Potocký, and J. Giertl, "Contribution to QoS Parameters Measurement: The BasicMeter Project," in *Conference proceedings of the 4th International Conference on Emerging e-learning Technologies and Applications ICETA 2005*, vol. 4, pp. 371–377, 2005.

[3] G. Sadasivan, N. Brownlee, B. Claise, and J. Quittek, "Architecture for IP Flow Information Export," Internet Engineering Task Force, RFC 5470 (Informational), March 2009. [Online]. Available: http://www.ietf.org/rfc/rfc5470.txt

[4] A. Pekár, "Podpora pre monitorovanie prevádzkových charakteristík siete v reálnom čase," Master's thesis, KPI FEI TU Košice.

[5] J. Tulach, *Practical API Design: Confessions of a Java Framework Architect*. Apress, 2008.