Journal of Applied Computing Research, 2(2):68-74
July-December 2012

Multiflow: Multicast clean-slate with anticipated route calculation on OpenFlow programmable networks

Lucas Bondan, Lucas F. Muller, Maicon Kist
Universidade Federal do Rio Grande do Sul. Av. Bento Gonçalves, 9500, 911501-970, Porto Alegre, RS, Brazil
lbondan@inf.ufrgs.br, lfmuller@inf.ufrgs.br, mkist@inf.ufrgs.br

Abstract. Increasingly popular, Internet applications for multimedia broadcasting require multipoint communication, in order to reduce network traffic rates. However, the widespread adoption of traditional multicast protocols is still held back by the current Internet structure, where the responsibility for management of multicast groups is distributed among network devices. By using distributed algorithms, such protocols generate delays in processing control groups events. In this paper we propose a clean-slate approach for multimedia multicasting, where the end to end calculation of the best route is performed to decrease delays in group configuration. The prototype developed implements this approach using OpenFlow technology. Results obtained through experimentation show a performance gain in relation to traditional IP multicasting.

Key words: Multicast, SDN, OpenFlow.

Introduction

Increasingly popular, Internet applications for multimedia broadcasting require communication among many hosts. In applications like Internet Protocol TV (IPTV) the content provider transmits data often identical to numerous subscribers of the service. These applications traditionally rely on IP multicast to perform multipoint communications, avoiding the waste of bandwidth when sending repeated data over multiple unicast connections.

Considering the decentralized nature of the Internet, where each router executes part of the routing algorithm, multicast routing protocols like Distance Vector Multicast Routing Protocol (DVMRP) and Multicast Open Shortest Path First (MOSPF) are not efficient to perform changes in the multicast tree. In such protocols routers are expected to exchange information with each other and then update their routing tables, process which may take some time to reach a consistent state. Furthermore, the Internet Group Management Protocol (IGMP), responsible for controlling entry and exit operations in groups of hosts, needs to communicate with many routers in order to notify events related to the multicast groups (Paul and Raghavan, 2002)

Therefore, in this paper we propose a clean-slate approach to multicast communication on which the calculations of routes between source and destination are made beforehand, with the purpose of speeding up the processing of multicast events. A prototype has been implemented, employing the concept of Software Defined Networks (SDN) with OpenFlow (Mckeown *et al.*, 2008) technology and experiments have been carried out in network topologies emulated in Mininet environment (Lantz *et al.*, 2010).

The remainder of this paper is organized as follows. In the first section, "Software Defined Networks" the concepts comprising the proposal of SDN and the OpenFlow technology are exposed. Then, in the section "Related Work" other proposals found in literature related with traffic management employing multicast protocols are presented. In the section "Multiflow" we present the prototype developed, describing its architecture. After that, in the section "Prototype Evaluation" the methodology for evaluating the prototype, named Multiflow, is depicted. In the section "Results" we present the results obtained from

the evaluation. Finally, conclusions and future remarks are discussed in the last section, "Final remark and further work".

Software Defined Networks

With the increasingly number of devices connected to networks, more applications are being developed to improve the communication between those devices. This growth brings not only the need for more processing power, but also a bigger complexity in the development of applications that help to provide a higher network performance.

The SDN concept proposes separating the data plan from the control plan, completely breaking all paradigms in current networks, fully removing the complex routines that manage data traffic from the network devices. The SDN architecture has, usually, a centralized operational system, which controls the actions that have to be taken by the interconnection devices.

Figure 1 illustrates the SDN concept (Yap *et al.*, 2010). It is possible to observe that the network is controlled by a centralized network operational system (c) by the OpenFlow software (a), which proposes a new approach for network mechanisms well established, such as protocols for traffic control and data delivery (b). In the context of SDN, a popular technology to provide a centralized data plan control is the OpenFlow protocol, described in the following subsection.

OpenFlow

OpenFlow is a protocol where the technology enables the coexistence of new protocols (experimental) and production traffic in the same network. This characteristic allows the abstraction and virtualization of networks, allowing the control of the network traffic through data flows (Kanaumi *et al.*, 2010).

The OpenFlow protocol is based on programmable switches that combine flexibility to develop new network applications and ease adapt of legacy switches to this new protocol.

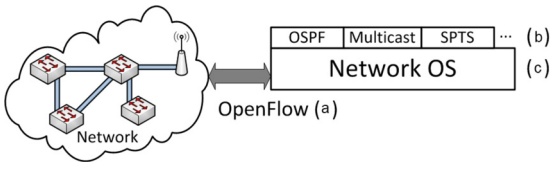

Figure 1. SDN operation concept.

OpenFlow switches are capable to realize forwarding of network packets through rules defined in their flow tables. Furthermore, there is a centralized element (controller) connected to all OpenFlow switches. The controller executes applications to manage the addition and removal of data flows in all switches through a control protocol (Mckeown *et al.*, 2008). Thereby, the OpenFlow controller acts like an operational system for network management and control, offering a platform based in the reutilization of components and different layers of network abstraction (API - Application Programming Interface).

The OpenFlow protocol defines the communication standard between all switches and the controller. The main function of the controller is perform the addition, removal and updating of entries in the flow table (Mckeown *et al.*, 2008). Utilizing this infrastructure, in the moment that a switch receives a packet, it must consult each of its flow tables if there is an entry defined for that packet in that flow. If yes, the actions are realized. Otherwise, when no entry is found in any flow table, it is sent to the controller, which will define an action to be executed for the packet in a specific flow. This process generally ends by adding a new entry in the flow table of the switch that received the packet (Open Network Foundation, 2012). This flexibility gives the controller the capability to configure flows (and flow rules) with a great level of detail.

With the programming flexibility offered by the controller in OpenFlow networks, multicast routing protocols can be completely reconsidered without the use of distributed algorithms. This paper explores this possibility and proposes an innovative multicast routing protocol.

Related work

In the context of an evolutionary approach for the Internet the work of Keshav and Paul (1999) proposed a centralized multicasting. They utilized a hierarchical structure of domains associated with gateways, switches and root controllers, adopting the concept of detaching the data and control flows. This is similar of what is proposed in this paper, but without the flexibility of SDN. In the same context, Ratnasamy *et al.* (2006) proposed using the existing unicast routes to distribute multicast packets, building an overlayed network. However, this solution doesn't receive

support from the network infrastructure, acting solely in the application level.

Recently, other approaches were utilized, along with the clean-slate line, as well as the proposal in this paper. The idea behind all clean-slate approaches is to reformulate the traditional manner which certain applications operate, often breaking paradigms related to widely used solutions. Martinez *et al.* (2007) used the Multiprotocol Label Switching (MPLS) over Virtual Private Network (VPN) to manage multicast traffic. However, this combination has high complexity because of the way the network is organized, creating scalability problems for this solution. Finally, Yap *et al.* (2010) have suggested high level primitives (API) based in OpenFlow to provide a more friendly development of multicasting networks. These primitives have a simplified implementation of the OpenFlow multipoint protocol, but does not consider questions such as changes in multicast groups.

None of the previous researches consider the approach adopted in Multiflow, a scalable, clean-slate multicast protocol with group management, knowledge of the all routes between network devices and preoccupation with the processing time of events.

Multiflow

Multiflow is a multicast, clean-slate approach to programmable networks, in which hosts can join and leave multicast groups dynamically, where the efficiency in the processing time of group events is fundamental. In the following subsections the Multiflow architecture is described, as well as the mechanisms to define routes and process group events.

Prototype description

Multiflow is the application that executes in the OpenFlow controller. The NOX controller (Gude *et al.*, 2008) was chosen to create the prototype due to its scalability. NOX is implemented in C and Python programming languages and is composed by modules that describe new functionalities. These modules are mostly developed in Python. NOX offers an initial set of modules, as for example APIs for handling UDP and IP packets. Also, the Mininet API (Lantz *et al.*, 2010) was used in order to build the virtual network topology where all tests were conducted.

Operation

Once the network topology is defined, the Multiflow application remains alert to any occurrence of IGMP packets in the network. Based in the IGMP protocol, Multiflow identifies packets whose destination is a multicast group. The flow sheet in Figure 2 illustrates the operations performed by Multiflow, showing the operations made when a host announces itself as a data provider for a specific multicast group, when a client enters in a multicast group and when a client leaves a group.

To start the transmission to a determined multicast group, the data server sends a packet of the type IGMP Query. Once this packet is identified, Multiflow recognizes the multicast group addressed in the packet as active in the network and store it in a list of active groups.

Clients interested in joining multicast groups must send an IGMP Join packet addressed to the desired group. When an IGMP Join packet is identified, the Multiflow application recognize the interest of the client in joining a multicast group and then initializes the best route discovery algorithm between the server(s) and the client(s). This calculation is possible only because Multiflow has knowledge of all possible routes in the network. The calculation of the best route employ the Dijkstra algorithm (Djkstra, 1959), widely used with this purpose. The Dijkstra algorithm is based in the concept of weight of edges between node connections. In this paper, the weight of each edge of a graph is defined as the distance between two nodes connected in the network. The best route is considered the list of edges that minimizes the sum of weights.

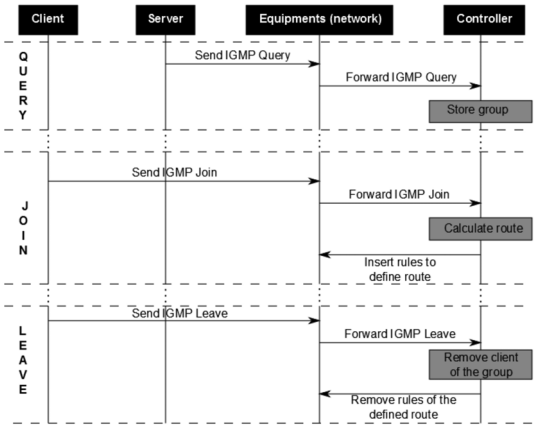

Figure 2. Multiflow operation flow diagram.

Once the best route is found, the controller insert in each node of each edge the necessary flow rules to correctly forward the packets between server(s) and client(s).

Another event present in the operation of the Multiflow, is when a client leaves a multicast group. This event occurs when a client, losing interest in a group, send an IGMP Leave packet. When this packet is received, the Multiflow application searches for the route that was previously added between the client and the server and removes the respective rules of each switch (node) of the route, this way cancelling the forwarding of packets from the given group to this client.

Prototype evaluation

This section presents details of the methodology used to evaluate the Multiflow implementation. The focus of this methodology is to identify the impact of the controller in the routing performance and in the formation of multicast groups. The following of this section presents the experimental setup and the network topology utilized.

Experimentation setup

To realize the experiments, a virtual network with switches that are compatible with the OpenFlow was utilized (Lantz *et al.*, 2010). From this, the definition of active hosts and to which group a host is a client is determined by an ordered execution of a set of bash scripts developed using the open-source software Mausezahn (MZ version 0.40) (Haas, 2012). By using those scripts, each host in the virtual network has the capability to generate IGMP query, join and leave packets.

To test the behaviour of multicast applications, an application in the client/server model was developed in the C programming language. This client application receives UDP packets from a machine running the server application.

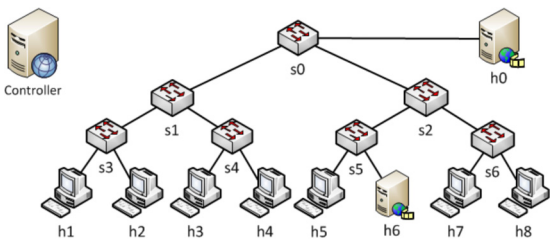

Figure 3. Evaluation topology.

In this scenario, the server application is the source of a multicast group. The server sends packets to a specific multicast group. In all other hosts, one instance of the client is executed. The client application is capable of providing statistical information about the performance of the solution, calculating the time span from when an IGMP Join is sent to a group from the first data packet received from this group. In other words, the client application calculates the time spend by the controller to calculate the best route from a client to the server, configure all necessary nodes and forward the first data packet from the server to that client.

Setup scenario

For the evaluation, a tree topology network was used, with seven switches and nine hosts, of which two are responsible to generate content for the multicast group and the rest of them represent clients interested or not in a given group. Figure 3 illustrates the generated topology in a tree format.

Within this scenario two important parameters are evaluated: the impact of the Multiflow controller in routing control performance and the impact of the performance in the formation of multicast groups. Multiflow was compared with another controller, OpenMcast, with offers network behaviour resembling to what is observed in normal networks, that does not make use of the SDN technology. The OpenMcast controller is detailed as follows.

OpenMcast

The OpenMcast controller was developed to simulate in a SDN environment the multicast operations realized in a standard network, based on the operation of the traditional IGMP protocol, in which is realized the propagation of control packets in the network. Once the server host sends an IGMP Query, the controller identifies the packet and propagates it through all ports in the switch (except the port that has received the packet). This flood operation is done by all switches in the network, with the purpose that all hosts take knowledge about the existence of the server of the multicast group at which the server in bound and the port from which each switch has access to the server. This operation is realized for all servers in the network.

Once all active multicast groups are known in the network, client hosts start their respective

join operations, sending IGMP Join packets. All IGMP Join packets are forwarded across the network through the port that gives access to the multicast server addressed in the packet until it reaches the switch directly connected to the server. Through this propagation, the path between the server and each client of a multicast group is configured. After this process, the delivery of UDP packets destined to a group is forwarded directly, without intervention from the controller, from the server to each client host that joined the group.

For the leave operation, the IGMP packet must be propagated the same manner as the IGMP Join. In this propagation, the path previously defined is removed from the flow rules of all in that path, interrupting the distribution of packets to the client.

Using the OpenMcast controller, the control packets exchanged in the network can generate considerable high traffic. Moreover, packet loss can occur during the propagation, invalidating an operation, which must be re-executed.

The sequence of operations realized for evaluating was planned based in a real situation, where clients that were participating in a group leave it or clients join one or more multicast groups. In the next section the results achieved in the scenario detailed are shown.

Results

This section presents the results obtained after a series of repetitions of the experiments. All results reflect the scenario described in the section "Prototype Evaluation".

In total one thousand measurements of the time elapsed between the join operation of a client and the receiving of the first UDP data packet addressed to the group. The time taken to configure each switch shows variability. The average time observed between the join operation and the arrival of the first UDP data packet in the OpenMcast was approximately 715 milliseconds, with a standard deviation of 357 milliseconds. The Multiflow approach, the same set of operations resulted in an average time of 427 milliseconds with a standard deviation of 431 milliseconds.

Figure 4 depicts those results in a more detailed view. In the vertical axis is possible to compare the average time of fifty consecutive executions of the interval time from the join operation and the receiving of the first UDP packet. The horizontal axis is shown the sequence number of the test executed. So, in the 200 execution, the y axis showed the average time elapse from the 151º to the 200º execution. This grouping of data makes the chart cleaner and easier to analyse. It's worth noting that the Multiflow controller, which has a priori knowledge of the network topology, performs better (lowest time) than OpenMcast that propagates control packets through switches.

Another interesting result can be observed in Table 1, which presents how the propagation IGMP Query packets escalate over the tree network topology presented in the section "Prototype Evaluation". In the OpenMcast controller, the relation between queries generate in servers and propagated in the network follows the equation $k * 2^n - k$, where k represents the number of queries generated in a server and n the number of levels in the tree topology. In other words, the greater the number of levels in the topology, the greater the number of control packets propagated through the network. In the Multiflow controller, however, the number of queries generated and propagated is always the same, since in this controller does not propagate control packets in the network.

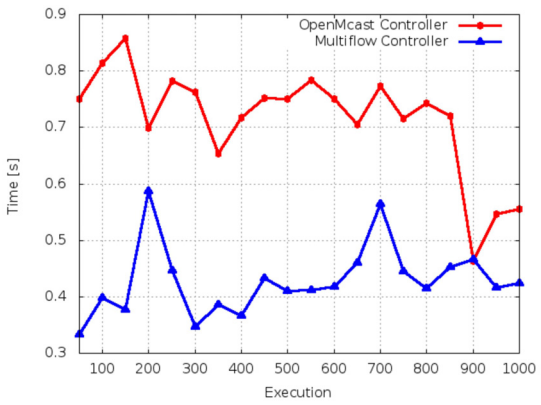

Figure 4. Average time from the client join and the receipt of the first packet.

Table 1. Propagated queries in the two controllers.

Levels (n)	Generated queries (k)	OpenMcast propagated queries	Multiflow propagated queries
2	100	300	100
3	100	700	100
4	100	1500	100

Case study

To justify the use of Multiflow consider the IPTV case study. A typical IPTV service can count on a large subscriber base and provide a number of channels. For each channel you can associate a different multicast group, in which the source would be the station, and the rest of the group would be composed of the multicast subscribers who are watching the channel.

In a TV's scenario, with a variety of channels, it is natural that users constantly alternate between channels. Since a different multicast group represents each channel, the hosts input and output tends to be intense. The work of Kim *et al*. (2009) studies the factors that contribute to increased latency-switching channels in IPTV scenarios using IP multicast. In this work, the authors conclude that the IGMP protocol contributes significantly to delays in the order of seconds.

To illustrate the use of the Multiflow in the context of IPTV, consider the topology of Figure 3, with two hosts transmitting streaming video via two distinct multicast groups. To accomplish the transfer, we used the application VLC Media Player (Videolan, 2012). Through VLC we can set up a video stream to a specific multicast group between the server host and the client host, we can also perform the multicast joining the group in question using VLC, which will display the video in real time.

Once participating in a multicast group receiving information from servers, one can simulate the changing of channels, like occurs in the IPTV.

To perform the channel changing, we simply perform an IGMP leave group operation in the client host and then perform a join operation in another multicast group available on the network. Figure 5 shows a screenshot of the server host and two clients performing a video streaming in the network simulated by Mininet using the Multiflow to control the network.

It is possible to observe from Figure 5 that there are two active multicast groups: 224.0.1.1 and 224.0.1.2, streaming videos viewed by two clients through the application VLC Player. In background of the videos, we can see windows of UNIX terminals displaying the execution log on the client stations, servers and the network controller.

Final remark and further work

Multicast is a packet routing technique for a specific group of network hosts, where the principal benefit is the traffic reduction due to the support of switches in copying and sending a message to several output ports. This paper proposes a clean-slate approach to multicast, logically centralized and based in OpenFlow programmable networks, where during the setup of a multicast group, is realized the calculation of the best existing route between the source and client hosts, with the objective of reducing at maximum the delay with the processing of multicast events.

Defined the approach, a proof of concept was developed, called Multiflow. Experiments were realized to characterize the setup and

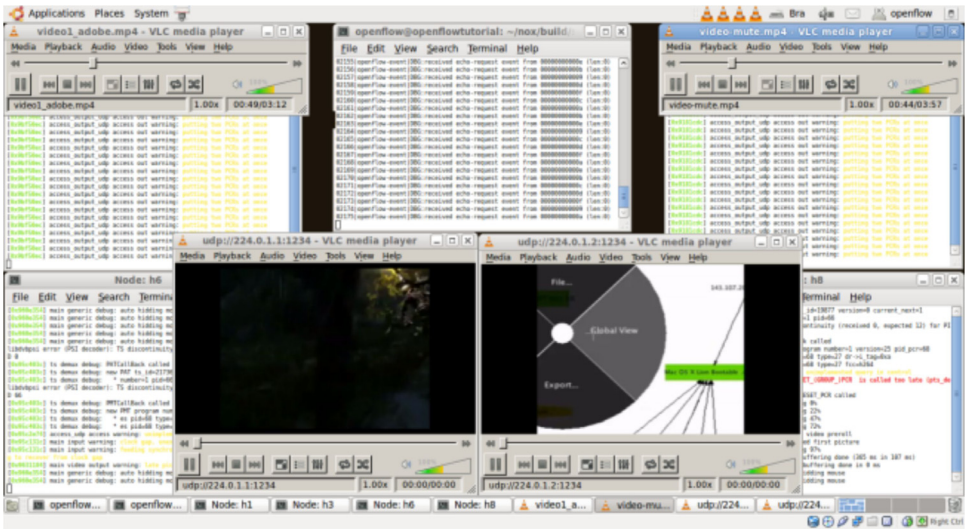

Figure 5. Video-streaming screenshot using Multiflow.

processing time of events. The results have shown that the utilization of a controller, with foreknowledge of the network topology, bring an overall performance improvement in the order of milliseconds, faster than other results published in literature of multicast IP. This improvement is due to the reduction of control information exchanged in the network and the best route calculation based in the topology. Therefore, the proposal of this paper can bring benefits to applications with multipoint communication requirements in which the operations of join and leave of a multicast group are frequent and represent a significant portion of the entire control traffic, as occurs in IPTV.

As future work, the development of experiments with scenarios more proximate of those found on Internet, directed to streaming services, mainly in scale level, is in the roadmap. Security questions, as for example the confidentiality of client and servers of a group and the transmitted messages are also of main concern.

Furthermore, the needs to evaluate heuristics that can be applied to the algorithm of best route, in order to reduce its complexity are a key aspect.

References

DIJKSTRA, E.W. 1959. A note on two problems in connection with graphs. *Numerische Mathematik*, **1**:269-271. http://dx.doi.org/10.1007/BF01386390

GUDE, N.; KOPONEN, T.; PETTIT, J.; PFAFF, B.; CASADO, M.; MCKEOWN, N.; SHENKER, S. 2008. NOX: Towards an Operating System for Networks. *In:* SIGCOMM COMPUTING COMMUNICATIONS REVIEW, 7, Seatle, 2008. *Proceedings…* Seatle, p. 105-110.

HAAS, H. 2012. Mausezahn Fast Traffic Generator. Available at: http://www.perihel.at/sec/mz. Accessed on: May 6, 2012.

KANAUMI, Y.; SAITO, S.; KAWAI, E. 2010. Toward Large-Scale Programmable Networks: Lessons Learned Through the Operation and Management of A Wide-Area OpenFlow-Based Network. *In:* INTERNATIONAL CONFERENCE ON NETWORK AND SERVICE MANAGEMENT, 2010. *Proceedings…* p. 330-333.

KESHAV, S.; PAUL, S. 1999. Centralized Multicast. *In:* INTERNATIONAL CONFERENCE ON NETWORK PROTOCOLS, 7, Washington, DC, 1999. *Proceedings…* Washington, DC, p. 59-68.

KIM, E.; LIU, J.; RHEE, B.; CHO, S.; KIM, H.; HAN, S. 2009. Design and Implementation for Reducing Zapping Time Of IPTV Over Overlay Network. *In:* INTERNATIONAL CONFERENCE ON MOBILE TECHNOLOGY, APPLICATION AND SYSTEMS, 6, New York, 2009. *Proceedings…* New York, p. 1-7.

LANTZ, B.; HELLER, B.; MCKEOWN, N. 2010. A Network in a Laptop: Rapid Prototyping For Software-Defined Networks. *In:* ACM SIGCOMM WORKSHOP ON HOT TOPICS IN NETWORKS, 9, New York, 2010. *Proceedings…* New York, p. 1-6.

MARTINEZ-YELMO, I.; LARRABEITI, D.; SOTO, I.; PACYNA, P. 2007. Multicast Traffic Aggregation in MPLS-Based VPN Networks. *IEEE Communications Magazine*, **45**(10):78-85. http://dx.doi.org/10.1109/MCOM.2007.4342827

MCKEOWN, N.; ANDERSON, T.; BALAKRISHNAN, H.; PARULKAR, G.; PETERSON, L.; REXFORD, J.; SHENKER, S.; TURNER, J. 2008. OpenFlow: Enabling Innovation in Campus Networks. *In:* ACM SIGCOMM COMPUTING COMMUNICATIONS REVIEW, 7, Seatle, 2008. *Proceedings…* Seatle, WA, p. 69-74.

OPEN NETWORK FOUNDATION. 2012. Openflow Switch Specification Version 1.2.0. Available at: http://www.openflow.org/documents/openflow-spec-v1.2.pdf. Accessed on: May 5, 2012.

PAUL, P.; RAGHAVAN, S.V. 2002. Survey of Multicast Routing Algorithms and Protocols. *In*: INTERNATIONAL CONFERENCE ON COMPUTER COMMUNICATION, 15, Washington, DC, 2002. *Proceedings…* Washington, DC, p. 902-926.

RATNASAMY, S.; ERMOLINSKIY, A.; SHENKER, S. 2006. Revisiting IP Multicast. *In:* CONFERENCE ON APPLICATIONS, TECHNOLOGIES, ARCHITECTURES, AND PROTOCOLS FOR COMPUTER COMMUNICATIONS, New York, 2006. *Proceedings…* New York, p. 15-26.

VIDEOLAN. 2012. VLC Media Player. Available at: http://www.videolan.org/vlc. Accessed on: May 8, 2012.

YAP, K.-K.; HUANG, T.-Y.; DODSON, B.; LAM, M.S.; MCKEOWN, N. 2010. Towards Software-Friendly Networks. *In:* ACM ASIA-PACIFIC WORKSHOP ON SYSTEMS, 1, New York, 2010. *Proceedings…* New York, p. 49-54.

Submitted on October 15, 2012
Accepted on February 26, 2013