SWITCHING TIME MEASUREMENT AND OPTIMIZATION ISSUES IN GNU QUAGGA ROUTING SOFTWARE

V. Eramo, M. Listanti, A. Cianfrani
University of Roma “La Sapienza”, INFOCOM Dept.

Abstract- OSPF (Open Shortest Path First) is a widely used intra-domain routing protocol in IP networks. Processing delays in OSPF implementations impact the time needed for both intra-domain and inter-domain routing to reconverge after a topology change. In this paper we introduce a performance index, referred to as the switching time, allowing the router reconverge to be characterized when network topology changes occur. We propose a test methodology in order to measure the introduced performance index in router realized with the Personal Computer hardware and equipped with Quagga, the most used open routing software. An optimization of the Quagga routing software has been performed and better performance has been obtained in terms of switching time.

Index Terms: Open Source Code, Routing Software Quagga, Open Shortest Path First, Dijkstra’s algorithm, Binary Heap

1 INTRODUCTION

The call for high performance switching and transmission equipment keeps growing, due to the continuous increase in the diffusion of information and communication technologies and of new applications and services based on audio video streaming and requiring high bandwidth. Packet routers have been able to support the performance growth by offering an ever increasing switching speed, mostly thanks to the technological advances of microelectronics.

Contrary to what happened for Personal Computers (PC) architectures, where standards were defined, that allowed the development of an open multivendor market, at least for the hardware component, the field of networking equipment in general and of packet switches in particular, has always seen the development of proprietary architectures. This means incompatible equipment and architectures, specially in configuration and management procedures. This situation has in practice given rise to commercial practices which are not based on free competition, and often the final cost of the equipment is high with respect to the offered performance and the equipment complexity.

Software implementations of routers based on standard PC hardware have been recently made available in the "open software" and "free software" world [1]. Those implementations are quite interesting, even if they often aim more to low-end alternatives of proprietary routers rather than top performance. In particular, the following projects are particularly interesting:

-Click Modular Router [1]: a software architecture based on Linux, and developed at the MIT, well documented, and freely distributed.

-Xorp [1]: an open router software platform under development at UC Berkeley, aiming at easy extensions to support future services. Xorp will support different hardware platforms, from simple PC, to specialized network processors [1], to dedicated hardware architectures. Xorp will support a variety of routing protocols and control interfaces. Scheduling and buffer management algorithms for QoS support are available.

-LRP [1]: a free and open distribution of Linux, that supports the main routing protocols. It allows a flexible and dynamic configuration of the router functionalities.

-Freesco [1]: a free and open distribution of Linux, similar to LRP, with reduced functionalities and simpler configuration. The maximum number of line interfaces is limited to 3 Ethernet cards and 2 modems. The router configurations allow only static assignments, and dynamic algorithms and routing protocols are not supported.

-MikroTik [1], a commercial solution that offers a software router (free but not open).

Some of the main companies that build switching equipment are considering the adoption of general purpose software platforms, if not standard hardware platforms. For example, several internal projects based on Linux are under way in Cisco, that is also considering with interest some initiatives for the open implementation of IOS [1].

Last two years we have participated to EURO (University Experiment of an Open Router), an Italian project whose main research goal has been to study a versatile, high-performance router, based on a standard personal computer (PC) architecture, and implemented following the open software/hardware philosophy. The choice of considering a router architecture based on a PC architecture is a consequence of the following facts: i) multivendor PC hardware is available at low cost (because of large scale production), ii) PC hardware architectures are well documented, and iii) their performance evolution is guaranteed by that of commercial PCs.

In particular we have developed a set of tests to analyze the routing performances of a router running the OSPF protocol [2], according to the IETF specifications [3-5]. In particular we have taken Black Box measures [6] of the time needed to perform the Switching Time computation on a Personal Computer (PC) equipped with Operating System Linux and Routing Software Quagga 0.98 [7]. The Switching Time, defined in [8,9], determines the time for an OSPF router to reconverge the routing tables and redirect data traffic when a best route to a destination is available. In particular it depends on the Shortest Path First (SPF) computation time, that is the time needed to execute the Dijkstra’s algorithm when a topology change is received by the router. The
Switching Time measures have been compared to the ones performed on the CISCO 2621 commercial router. The Switching Time evaluation raises the evidence of a lack of optimization in the Dijkstra’s algorithm implemented in Quagga 0.98. A deep analysis of the code evidenced that the data structures used to implement the Candidate List [10] during the SPF calculation were not optimized. So we have modified the code and have implemented a binary heap data structure [10]. The new implementation allows a router based on the PC hardware and Quagga 0.98 routing software to have performance better than commercial OSPF routers.

The paper is organized into five sections. Both the router model based on PC-hardware and Quagga Routing Software are described in Section 2. The test methodology and the software needed in order to evaluate the Switching Time are illustrated in Section 3. In Section 4 the experimental results obtained on GNU Quagga are shown and compared to the ones taken on the CISCO 2621. The results reported in Section 4 will show that the source code implementing the SPF computation is not optimized in Quagga because does not allow the expected trend of the Dijkstra’s algorithm to be obtained. Both the code optimization we have carried out and the Switching Time results on the router equipped with the optimized Quagga Routing Software are described in Section 5. Finally the main conclusions and further research items are illustrated in Section 6.

2 ROUTER MODEL BASED ON PC HARDWARE AND QUAGGA ROUTING SOFTWARE

The hardware available on a PC allows a shared bus, shared memory router to be implemented. The PC-based router architecture is shown in Figure 1. The Network Interface Cards (NIC) receive and store packets in the main RAM, the CPU routes them to the correct output interface, and NIC fetch packets from the RAM and transmit them on the wire. Today some open source operating systems are available that implement IP functionalities. In particular the networking code in the Linux kernel is considered to be the most modular: the hardware-independent IP stack has a well defined application programming interface (API) toward the hardware-dependent device driver, which is the glue making the IP layer to operate with most networking hardware. The Linux kernel networking code implements a standard RFC 1812 IP router. After a few sanity checks such as IP header checksum verification, packets that are not addressed to the router are processed by the routing function which determines the IP address of the next router to which they must be forwarded, and the output interface on which they must be enqueued for transmission. The Kernel implements an efficient routing cache based on a hash table with collision lists; the number of hash entries is determined as a function of the RAM available when the networking code is initialized at boot time. The route for outgoing packets is first looked up in the routing cache by a fast hash algorithm, and, in case of miss, the whole routing table stored in the forwarding information base (FIB) is searched by a slower prefix matching algorithm. Dynamic FIB are built using routing protocols. These protocols are ways in which routers communicate among them, giving each other information about the most efficient way of routing data given the current state of the network. A router with a dynamic routing table can automatically switch data to a backup route if the primary route is down. It can also always determine the most efficient way of routing data toward its final destination.

Linux operating system does not implement code for routing protocols but some open source routing software are available today. One of the most popular is Quagga [7], a routing software that provides TCP/IP based routing protocols. It is a free routing system designed for Unix operating systems, including Linux, BSD and Solaris. One of the attractive of Quagga is that it does not require dedicated hardware, such as a router, to run on. Quagga runs on several platform, mainly PC-based platform, such as Linux, FreeBSD, NetBSD and OpenBSD as well as SUN Solaris. With Quagga a multihome computer, that is a host with multiple interfaces, can easily be configured as a router that runs multiple routing protocols. Quagga is made from a collection of several daemons that work together to build a routing table. In particular the ripd daemon handles the RIP protocol [2], while ospfd is a daemon which supports OSPF protocol [2], bgpd supports the BGP-4 protocol [2]. The zebra daemon is responsible for Linux kernel routing table update changes and it redistributes the routes between different routing protocols.

3 DESCRIPTION OF THE TEST-BED FOR THE EVALUATION OF THE SWITCHING TIME

This test determines the time for an OSPF router to reconverge the routing table and redirect data traffic when a best route to a destination is available. To determine the route reconvergence performance of a Device Under Test (DUT) we use the test configuration reported in Figure 2.

The DUT is connected to two testing PCs, called PC-A and PC-B respectively. The PC-A is connected to the DUT with an only Fast Ethernet Network Interface. Its function is to generate the data traffic that the DUT will...
switch when a best route will be available. It generates UDP traffic by means of the RUDE traffic generator [11].

Fig. 2 In order to perform the Switching Time, the Device Under Test is connected to testing PC-A and PC-B. PC-A sends data packets by means of the traffic generator RUDE. PC-B emulates a network topology and decides which path will be followed by the data packets.

The PC-B is connected to the DUT with two Fast Ethernet Network Interfaces. Its function is to emulate a complex network topology and to generate some particular Link State Advertisements (LSA) notifying to the DUT the availability of a best route toward a destination network of the emulated topology. In particular the PC-B allows the topology reported in Figure 3 to be emulated.

This topology is made up of the two routers B1, B2 and a variable number of fictitious routers and networks, so that the DUT will have to find the shortest path to all the vertexes of the emulated network, a vertex being either a network or a router. In particular the emulated network topology is generated by means of the software BRITE illustrated and available in [12]. So that the DUT “sees” the emulated network topology it is needed that the testing PC-B sends to the DUT some appropriate Link State Advertisements (LSAs) describing the emulated network topology. For this reason the PC-B has been equipped with the LSA generator software SPOOF illustrated and available in [13].

Accoring to the test configuration reported in Figure 3, the data traffic sent from the PC-A can reach any destination network \( N_d \) of the emulated topology through two different paths each involving one of the two network interfaces between the DUT and the PC-B. In particular if the costs \( c_j \) of the networks \( N_j \) and \( N_d \) are equal, either of the paths will be chosen by the DUT according to the cost values \( c_j \) of the networks \( N_j \) and \( N_d \). At the beginning of the test we set \( c_j = c_{j_0} \) and all data traffic is directed through the path including networks \( N_1 \), \( N_2 \) and router \( B_1 \). Then the PC-B will generates an update LSA with new cost \( c_j \) and such that \( c_j > c_{j_0} \). After that the DUT will have processed the update LSA, computed the new best paths by means of the Dijkstra’s algorithm and updated its routing table, the data traffic will be switched on the best paths including networks \( N_3 \), \( N_4 \) and router \( B_2 \). The time this switching operation takes will be called Switching Time. The steps description in the test aiming at evaluating the Switching Time is illustrated in Figure 4.

**Fig. 3** A network topology is emulated in PC-B by sending appropriated LSAs to the DUT. Initially \( c_1 = c_2 \), \( c_3 = c_4 \) and the data traffic is sent through the path involving Networks \( N_1 \), \( N_2 \) and router \( B_1 \). After that PC-B sent an update LSA, \( c_1 = c_2 \), \( c_3 > c_4 \) and the data traffic is switched on the path involving Networks \( N_3 \), \( N_4 \) and router \( B_2 \).

**Fig. 4** Steps description in the test aiming at evaluating the Switching Time

It is composed by the following steps:

1) The PC-B emulates a network topology and loads it on the DUT by sending the LSAs describing the topology. The cost of the networks \( N_1 \) and \( N_2 \) are chosen such that \( c_1 < c_2 \).

2) The PC-A sends data traffic to a destination network \( N_d \) of the emulated network. According to the costs chosen in step 1, the DUT will route the data traffic on the best path passing through Network \( N_1 \).

3) PC-B sends an update LSA with new cost \( c_3 \) and such that \( c_3 > c_4 \). The PC-B also measures the time instant \( T_1 \) in which the update LSA is sent. After that the DUT has received and processed the update LSA, computed the Shortest Path First (SPF) and updated the routing table, the data traffic will be routed on the new path including network \( N_2 \).

4) PC-B measures the time instant \( T_2 \) in which the first data packet is received from the interface connected to Network \( N_2 \).

5) PC-B computes the Switching Time \( T_s = T_2 - T_1 \).

In particular the test will be performed varying the destination network \( N_d \).

4 PERFORMANCE EVALUATION

The router tested is based on a high-end PC equipped with one 2.8Ghz Intel Xeon processor and 512Mbyte RAM. All experiments were performed running Linux 2.6 and Quagga 0.98 Routing Software.

The test input parameters are the following: i) \( F_p \), the constant rate at which the packets are transmitted by PC-A; ii) \( L_p \), length of the packets sent from the PC-A; iii) \( N \)
and \( M \), the number of vertexes and edges of the directed graph representing the emulated network topology respectively.

In particular notice that the \( N \) and \( M \) parameters depends on the number of the routers and networks in the emulated network topology.

In all performed tests the router B1, B2 are connected to a fully meshed network topology having each router connected to each other through a different transit network. Figure 3 shows an example of emulated fully meshed topology with 4 routers. It is important to remark that in representing the emulated network as a directed weighted graph [2], each router and each transit network of the emulated network topology becomes a vertex of the graph, and each network-router link becomes an edge. Each edge is labelled with a cost representing the interface cost of the link connecting a router to a network [2].

The Switching Time will depend on the number of vertexes \( N \) and edges \( M \) in the graph representing the emulated network. This is due to the fact that one of the components of the Switching Time is the SPF computation time whose evaluation depends on both \( N \) and \( M \). In particular the SPF computation is primed when the DUT receives the update LSA.

If we consider an emulated network topology composed by \( R \) routers, we have that:

\[
N = \frac{R(R-1)}{2} + R + 4 \quad (3.1)
\]

\[
M = 2R(R-1) + 8 \quad (3.2)
\]

It is important to notice that the number of edges \( M \) is almost proportional to the number of vertexes \( N \). In particular from (3.1) and (3.2) we can assume that \( M = O(N) \).

We have performed the measurements in two different cases. In the first case, called “near vertex case” the chosen destination network \( N_d \) is near the DUT, that is the vertex representing \( N_d \) in the directed graph is among the first to be inserted in the Spanning Tree when the DUT executes the Dijkstra’s algorithm. In the second case, called “far vertex case” the chosen destination network \( N_d \) is among the latest vertexes to be inserted in the Spanning Tree. In the performed measure we can choose either cases by modifying the cost of the edges connected to Network \( N_d \).

We report in Figure 5 the Switching Time as a function of \( N \) for packet length and rate \( L_p=100 \) bytes and \( F_p=2000 \) pack/s respectively. The values obtained for the PC-based router are compared to the ones of a commercial access router, the CISCO 2621. We consider both near and far vertex cases mentioned in Section 3. In particular for the PC-based router an only curve is shown because we have obtained the same Switching Time values for the near and far vertex cases. This is due to the fact that in Quagga Routing Software when the router receives an update LSA, it first executes the Dijkstra’s algorithm and after it updates the FIB. That is the router first evaluates the best paths to reach all of the destination networks and after it updates the FIB by inserting in it the IP interface addresses of the next hop router allowing the networks to be reached through the previously evaluated paths. On the contrary in CISCO 2621 the Dijkstra’s algorithm and the FIB updating procedure are probably executed simultaneously. Hence when the best path of a destination network has been evaluated, the router updates immediately the IP interface address of the next hop router allowing the network to be reached through the previously evaluated paths. This is the reason for which in CISCO 2621 the Switching Time in the near vertex case is smaller than in the far vertex case.

It is possible to notice from Figure 5 that the experimental values taken on Cisco 2621 router fit the curves \( 1.4 \times 10^{-4}(N + N \log N) \) very well for far vertex case. This result was expected because the obtained trend represents the complexity of the Dijkstra’s algorithm [10] executed by the DUT in order to evaluate the Short Path First whose computation time represents the main component of the Switching Time. On the contrary the results obtained on a router based on the PC hardware and equipped with Quagga 0.98 routing software are quite different. The computation time quickly increases up to 5 sec. as the number of vertexes reaches 2500 and the measured values fit on the \( 7.6 \times 10^{-7} N^2 \) interpolating curve. Comparing the two curves, it appears that the GNU Quagga performance is better than the one of the Cisco 2621 only when the number of routers \( R \) in the emulated topology is smaller than 45 and 60 for the near and far vertex cases respectively and corresponding to 1150 and 1800 vertexes respectively. With more than 48 and 60 routers for the near and far vertex cases, instead, the Cisco 2621 performance becomes better, because GNU Quagga pays the fees of a sub-optimal implementation of the Dijkstra’s algorithm. On the basis of these results we retain that some changes are needed inside the GNU Quagga code, to obtain performances comparable to top level commercial routers. Section 5 therefore will be dedicated to the analysis of the Dijkstra’s algorithm complexity and to its optimisation.
5 OPTIMIZATION OF THE SPF COMPUTATION IN GNU QUAGGA

The SPF computation is based on the Dijkstra’s algorithm, as described in [2]. The algorithm examines the directed weighted graph already described in Section 4, in order to find the shortest paths from a root vertex to each other vertex in the graph. All these paths give raise to a Spanning Tree of the graph. In Quagga 0.98 the directed graph is itself represented by the LSA set, stored in the LSA Database. During the iterations of the algorithm all the vertexes must be extracted, one by one, from the graph and inserted into the Spanning Tree. Moreover Quagga 0.98 also uses the Candidate List, a structure that contains all the reachable vertexes that have not yet been inserted but can be reached from vertexes already inserted into Spanning Tree. The Candidate List is used as a step in the middle during the migration of the vertexes from the graph to the Spanning Tree. Each of the reached vertexes is extracted from the graph, inserted into the Candidate List and provided with a key that represents the total cost needed to reach it starting from the root and crossing the minimum cost path composed by only the vertexes that have already been inserted into the Spanning Tree. According to the Dijkstra’s algorithm a vertex will be extracted from the Candidate List and inserted into the Spanning Tree only when it becomes the node with the lowest key in the Candidate List. The algorithm finishes when all of the vertexes have been inserted in the Spanning Tree and that occurs when the Candidate List becomes empty. During this procedure the Candidate List is the most stressed structure. Its management is the key point of the resulting global performances, and it is performed by four different functions: the Extract-Min function, that finds and extracts the node with the minimum key from the Candidate List; the Insert function that inserts a node into the Candidate List, the Decrease-Key function, needed to update and to reduce the total cost associated with a particular node; the Lookup function, needed to check if a node is already present inside the Candidate List. In Section 5.1 we will evaluate the complexity of the operations for the Candidate List management in Quagga 0.98. In Section 5.2 a Binary Heap data structure implementing the Candidate List will be proposed and its complexity will be evaluated; because a Binary Heap data structure does not support the Lookup function efficiently, we will illustrate how this operation can be eliminated by modifying the LSA database data structure. Finally the results concerning the modified Quagga 0.98 routing software will be evaluated in Section 5.4.

5.1 Complexity Evaluation of Quagga 0.98

Next we evaluate the computational time complexity of the functions for the management of the Candidate List in Quagga 0.98. Let us consider a graph with N vertexes and M edges. Each vertex will be inserted and extracted from the Candidate List before it becomes part of the Spanning Tree, so both the Insert function and the Extract-Min function will be performed exactly N times. The Lookup function is performed for each one of the M edges of the graph, so exactly M times. The number of times in which the Decrease-Key function is performed cannot a priori be evaluated because depends on the values of the costs of the links. However it will be always less than the number of edges, so we can assume that the Decrease-Key function will be performed O(M) times.

The average cost of each of these functions, will depend on the data structure used to store the Candidate List. In Quagga 0.98 that structure is a linked list, sorted in key increasing order. As the first element of the list is the one with the minimum key, the Extract-Min function is almost costless and its cost is O(1). Regarding the Insert function, as the list is key-ordered, we need to go down along the list, element by element, in order to find the right place where to insert the new element. Because in the worst case, the list contains N element, the cost of this operation is O(N). Both the Lookup and the Decrease-Key functions work the same way, and have again a O(N) cost.

According to the previous considerations we are able to calculate the amortized cost of the functions above mentioned. The Insert function has O(N2), the Extract-Min function has O(N), the Decrease-key function has O(M*N) and the Lookup function has O(M+N) amortized cost.

The total amortized cost of the implementation of the Dijkstra’s algorithm in Quagga 0.98 will not be less than the sum of these four costs, i.e. not less then O(N^2 + M*N). If we want to obtain an O((N+M)*log N) total cost, we need to reduce the cost of the Insert, Decrease-Key and Lookup functions down to O(log N). This result can be achieved only changing the data structure adopted to implement the Candidate List.

5.2 A Binary Heap data structure to implement the Candidate List in modified Quagga 0.98

We have modified the Quagga 0.98 original version and we have written a patch available in [14]. In particular in the new Quagga version we have chosen the binary heap data structure to replace the poor sorted list used in the original Quagga version.

A binary heap is a complete and balanced binary tree with a local sorting [10]. Leaves are always inserted starting from the left, and a new level is actually created only when the previous one is complete. Thus the heap depth is always less than logN, where N is the number of nodes. Each node of the heap has a key, and the whole heap is locally ordered on these keys, so that each node has a key lower than the ones of both its children. This particular sorting ensures that the node with the minimum key is the root of the heap.

In [10] it is shown that the binary heap can perform the Extract-Min, Insert and Decrease-Key functions of a node in a O((N+M)*log N) time complexity. The amortized costs consequentially change to O(Nlog N) for the Insert function, O(Nlog N) for the Extract-Min and O(Mlog N) for the Decrease-Key function.

Unfortunately the changes made to implement the Candidate List rise a new problem: the binary heap does not support the Lookup function, as the structure is only locally ordered, and finding a particular node would require to scan one by one all the nodes, thus obtaining again a O(N) cost. Instead of finding a way to implement the Lookup function, with a O(log N) cost, we have...
modified the LSA database data structure so that the Lookup function becomes no longer needed at all. In particular for each LSA, stored in the database, we have added an information denoting if or not the LSA is in the Candidate List. In positive case the information also denotes the position in the Candidate List where the vertex associated to the LSA is stored. That allows a vertex associated to an LSA to be immediately accessed during the execution of the Dijkstra’s algorithm. Further, because the Extract-Min, Insert and Decrease-Key operations may change the position of a vertex in the Candidate List, a pointer to the information of the associated LSA is added for each vertex so that the position updating can be accomplished when these operations are performed.

Finally, because before we will show that by modifying the LSA database data structure the Lookup function is no more needed, the total amortized cost of the new implementation of the Dijkstra’s algorithm in modified Quagga 0.98 routing software becomes as expected $O((M+N)\log N)$.

5.3 Numerical Results for Modified Quagga 0.98 Routing Software

The test evaluating the Switching Time on the modified Quagga 0.98 version produced experimental results that perfectly reflect the $(M+N)\log N$ trend. The measured values, varying the number of vertexes in the graph, are presented in Figure 6, and compared with the same measure taken on the Cisco 2621 and on the original Quagga 0.98 version. For the CISCO 2621 we report only the curve relative to the near vertex case. In Figure 6, we show the results for a packet length $L_p=100$ bytes and when two different packet rates are considered, $F_p=2000$ pack/s and $F_p=5400$ pack/s. From the results shown we notice that the Switching Time on the modified Quagga version is always less than the time needed on the original version, proving that the optimization process have been successfully completed. Moreover the performance of the PC based router equipped with the routing software modified Quagga 0.98 becomes in term of Switching Time about eight times faster than the one of the Cisco 2621 when $F_p=2000$ pack/s and the emulated network topology is composed by a number of vertexes equal to 1830. In particular the Switching Time equals 1.7 sec. and 0.22 sec. in the Cisco 2621 and modified Quagga 0.98 case respectively. From the results reported in Figure 6, we can also notice that in CISCO 2621 router the performance depends much on the packet rate considered, in fact when $N=1830$ the Switching Time passes from 0.8 sec. to 1.7 sec. when the packet rate increases from 2000 pack/s to 5400 pack/s. This is due to the fact that when the packet rate increases the resources available to process the update LSA, to execute the Dijkstra’s algorithm and to update the FIB are reduced because a part of them are engaged to process and to route the IP data packets. The Switching Time increase is negligible in the PC-based router case, because the PC processing capacities are higher than in the CISCO 2621 router case so that the IP data packet processing operation does not influence much the routing protocol performance.

6 CONCLUSIONS

The aim of our work was to evaluate the routing performance an OSPF router built by using a standard PC equipped with open source Operating System and Routing Software. The first experimental results for the Switching Time obtained on a PC running GNU Quagga were not good. We have performed an optimization of the Quagga code and we have shown that it is possible to reach performance better than the ones of a commercial router.

REFERENCES