International Journal of Computer Science & Applications

Vol. 4, No. 1, pp. 53 - 68

OSPF Performance and Optimization of Open Source Routing Software

Vincenzo Eramo, Marco Listanti and Antonio Cianfrani
INFOCOM Dpt, University of Rome "La Sapienza",
Via Eudossiana 18, 00184 Roma-Italy
e-mail: Vincenzo.Eramo@uniroma1.it

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 re-convergence after a topology change. The re-convergence capability of a router can be characterized by a performance index, referred to as switching time. We have built a test-bed and we have measured the switching time in routers based on the Personal Computer (PC) hardware and open source routing software. The obtained results show that, if the open source routing software is optimized, the PC-based router performs, in terms of switching time, better than a commercial router as Cisco 2801.

Index Terms: Open Source Code, Xorp and Quagga Routing Software, Open Shortest Path First

1. Introduction

The demand for high-performance switching and transmission equipment keeps growing, due to the increasing popularity of information and communication technologies and the emergence of new high bandwidth applications and services based on audio/video streaming. Thanks to the technological advances of microelectronics, packet routers have been able to offer ever increasing switching speed.

Unlike 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 [1-7]. Those implementations are quite interesting, even if they often aim more to be low-end alternatives of proprietary routers rather than top performance. In particular, the following projects are particularly interesting:

- Click Modular Router [8]: a software architecture based on Linux, and developed at the MIT, well documented, and freely distributed.

- Xorp [9]: 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 [10], 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 [11]: 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 [12]: 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.

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 [12].

Criticism to software routers are focused on limited performance, instability of software, lack of system support, scalability problem, lack of functionalities. Performance limitations can be compensated by the natural evolution of the performance of the PC architecture. Current PC-based routers and switches have the potentiality for switching up to a few Gbit/s traffic, which is more than enough for a large number of applications. Today, the maturity of open source software overcomes most problems related to stability and availability of software functionalities. It is therefore important to explore the intrinsic limitations of PC-based router [13].

We are participating to BORA-BORA (Building Open Router Architecture Based on Router Aggregation) [14], an Italian project whose main research goal has been to develop a versatile, high-performance router, based on a standard PC architecture, and implemented following the open software/hardware philosophy. This is motivated by the following observations: 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.

Some partners of BORA-BORA project focused their attention on data plane [13,15]. It has shown that when looking for high-end performance, commercially available Network Interface Cards (NICs) present some limitations. When both the classical Linux stack and Click [8] were used to build a router with commercial NICs, is not possible to route a single 1 Gbit traffic flow of minimum size packets, even if the internal bus bandwidth is 8 Gbit/s. The main limitation come from CPU overload and from a long latency during host memory read operations, that represents up to 90% of the transfer time for minimum size packets. One possible solution to overcome performance limitations is building a custom NIC based on the well know idea of direct line-card communication, which is not possible today with commercial NICs that lack programmability. This approach presents several advantages: latency during read operations is avoided, a more efficient use of the bus is achieved since packets traverse only once the bus and CPU resources are freed up. The authors describe in [16] the development of a specific NIC allowing them to obtain a wire speed transfer in all scenarios, reaching up to 6 Gbit/s of aggregate throughput considering 64-byte packets.

Within BORA-BORA project we focus on limited performance of the control and management plane of a PC-based router. In the work carried out in [17,18], we have developed a set of tests [19-21] to analyze the routing performances of a router running the OSPF protocol [22-24].

In this paper, we have conducted Black Box measurements [25] of *switching time* [26-28] in a PC running Linux [29] and Open Source [30,31] routing software. *Switching time* is defined as the time needed by a router to reconverge its routing tables and redirect data traffic to the best route. *Switching time* depends on the *Shortest Path First* (*SPF*) computation time, that is, the time needed to execute Dijskstra's algorithm when a topology change is received by the router. Our measurement results compare *switching time* in a PC router and in a Cisco 2801 commercial router [32].

From our evaluation of *switching time* we uncover an inefficiency in the implementation of Dijsktra's algorithm in Quagga [31] routing software. Our analysis of the code found that the data structures used to implement the Candidate List [33] during the *SPF* calculation were suboptimal. We modified the code to use a binary heap data structure [34] and have evaluated the *switching time* of the new version of Quagga.

The paper is organized into six sections. In Section 2 we describe the router model based on PC-hardware and Open Source routing software. In Section 3 we discuss the test methodology and the software needed for evaluating *switching time*. In Section 4 the experimental results for a PC router are shown and compared to the ones for a Cisco 2801. Both the code optimization we have carried out and *switching time* results on the router equipped with the optimized Quagga routing software are described in Section 5. Finally, Section 6 presented the main conclusions and further research items.

2. Router Model based on PC hardware and Open Source Routing Software

The hardware available on a PC allows a router with shared bus and memory to be implemented. The PC-based router architecture is shown in Fig. 1. The Network Interface Cards (NICs) receive and store packets in the main RAM, the CPU routes them to the correct output interface, and NICs fetch packets from the RAM and transmit them on the wire.

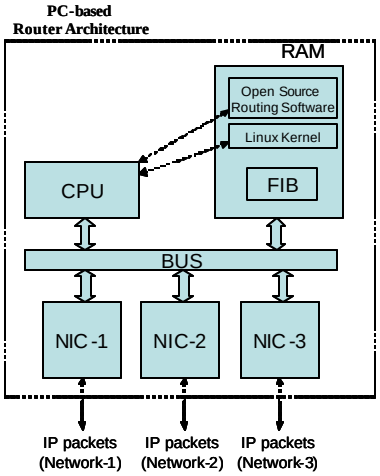

Fig.1. IP router based on the Personal Computer (PC) Architecture and the Open Source

Routing Software

Today some open source operating systems are available that implement IP routing functionalities. In particular, the networking code in the Linux kernel is 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 [35] 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 available RAM 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 the case of a miss, the whole routing table stored in the forwarding information base (FIB) is searched using a longest prefix matching algorithm. The FIBs are built using routing protocols. These protocols are used by routers to give each other information about the most efficient way of routing data given the current state of the network. In general, 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.

The Linux operating system does not implement any routing protocols but some open source routing software is available today. The most popular are Xorp [30] and Quagga [31], two open source routing software for IP networks. They are routing systems designed for Unix operating systems, including Linux, BSD and Solaris. One of the strengths of Xorp and Quagga is that they do not require special hardware, just a PC. With Xorp and Quagga, a multihome computer, that is a host with multiple interfaces, can easily be configured as a router that runs multiple routing protocols.

3. Test-bed for the evaluation of the *switching time*

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

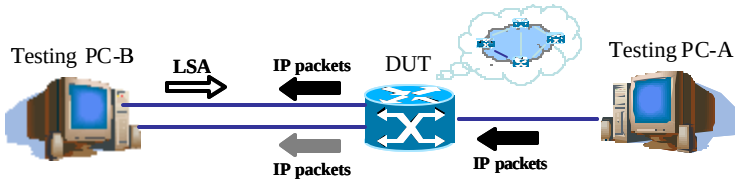

Fig. 2. To measure the *switching time*, the Device Under Test is connected to PC-A and PC-B. PC-A using RUDE traffic generator, sends data packets. PC-B emulates a network topology and decides which path will be followed by the data packets.

The DUT is connected to two testing PCs, called PC-A and PC-B respectively. PC-A is connected to the DUT with one Fast Ethernet network interface. Its function is to generate the data traffic that the DUT will switch using the best available route. PC-A

generates UDP traffic by means of the RUDE traffic generator [36]. 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 (LSAs) notifying the DUT of new best routes toward a destination network in the emulated topology. In particular, PC-B emulates the topology reported in Fig. 3. This topology is composed of two edge routers B1 and B2, and a variable number of simulated routers and networks. The DUT has to find the shortest paths to every network and router. The emulated network topology is generated by using the software BRITE [37-39]. PC-B uses the LSA generator software SPOOF [40] to send to the DUT appropriate LSAs describing the emulated network topology.

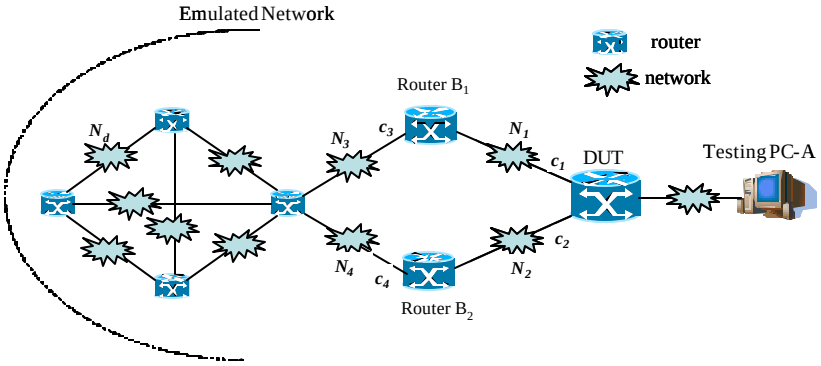

Fig.3. A network topology is emulated in PC-B by sending appropriated LSAs to the DUT. Initially link costs satisfy $c_1=c_2$, $c_3<c_4$ and the data traffic is sent through Networks N_1 and N_3 and router B_1. After that, PC-B sends an update LSA, with $c_3>c_4$ and the data traffic is switched on the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

As illustrated in Fig. 3, the data traffic sent from 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 PC-B. In particular, if the costs c_1 and c_2 of the networks N_1 and N_2 are equal, either of these paths will be chosen by the DUT according to the cost values c_3 and c_4 of the networks N_3 and N_4. At the beginning of our tests we set $c_3<c_4$, which means that all data traffic is directed through the path including networks N_1 and N_3 and router B_1. At instant t_{u_LSA}, PC-B generates an update LSA with new cost c_3 such that $c_3>c_4$. Then the DUT processes the update LSA, computes new best paths by means of Dijkstra's algorithm and updates its routing table. As a result, data traffic is switched to the path including networks N_2 and N_4 and router B_2; we indicate with t_{UDP} the time in which the first UDP packet is received on B2. The time this switching operation takes will be called *switching time*. The steps performed to evaluate the *switching time* measurement is illustrated in Fig. 4. The test can be performed varying the position of the destination network N_d. In particular we consider the case of highest switching time and this happens when N_d is the last inserted in DUT Routing Table. In this case the components of the swit ching time, reported in Fig. 5, are:

- RTT, the Round Trip Time between the PC-B and the DUT;
- T_{proc_LSA}, the update LSA processing time;
- T_{SPF}, the Shortest Path First computation time, the time needed to perform Dijkstra algorithm finding shortest paths from the DUT to all destinations;
- T_{upd_RT}, the time needed to update the routing table of the DUT

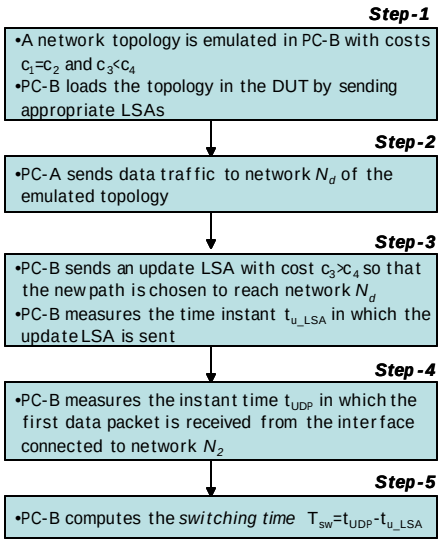

Fig.4. Steps for the evaluation of *switching time*

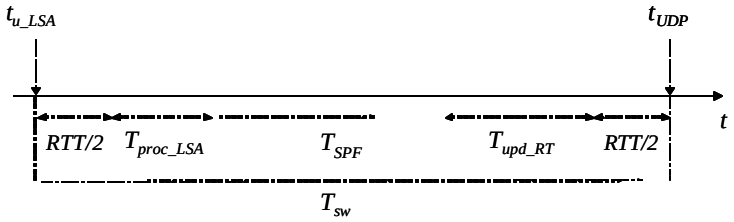

Fig.5. Components of the switching time T_{sw}.

The switching time can be calculated as the time difference between the sending of the update LSA and the receiving of the first UDP packet on the new path including networks N_2 and N_4 and router B_2:

$$T_{sw} = t_{UDP} - t_{u\_LSA} \tag{3.1}$$

and can be expressed as follows:

$$T_{sw} = RTT + T_{proc\_LSA} + T_{SPF} + T_{upd\_RT} \tag{3.2}$$

Let us analyze how each component influences the switching time. *RTT* can be ignored because PC-B and DUT are connected through a Fast Ethernet directed link so its value is smaller than 1 μs. T_{proc_LSA} and T_{SPF} can be measured according to the test methodologies described in [19-21]. From results reported in [17,18] it is possible to verify that T_{proc_LSA} is much smaller than both T_{SPF} and T_{upd_RT}. In conclusion switching time mainly depends on two operation: SPF computation time T_{SPF} and routing table updating time T_{upd_RT}.

4. Performance Evaluation

The tested router runs on a high-end PC equipped with one 2.8 Ghz Intel processor and 512Mbyte of RAM. All experiments were performed using Linux 2.6 and Xorp and Quagga routing software. The obtained results are compared to the ones taken on the Cisco 2801 commercial router [32] with IOS 12.3(14) Operating System and equipped with 128Mbyte of RAM.

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, the length of the packets sent from PC-A; iii) N and M, the number of vertexes and edges of the directed graph representing the emulated network topology respectively. Notice that the N and M parameters depends on the number of routers and networks in the emulated network topology.

In all of the performed tests, the routers B1 and B2 are connected to a fully meshed network topology, where each router is connected to each other through a different transit network. Fig. 3 shows a sample topology with 4 routers. It is important to remark that in representing the emulated network as a directed weighted graph [41], 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 labeled with a cost representing the interface cost of the link connecting a router to a network [41].

Switching time depends on the number of vertexes N and edges M in the graph representing the emulated network. This is due to the fact that, as mentioned at the end of Section 3, one of the components of *switching time* is the *SPF* computation time whose evaluation depends on both N and M. In particular, the *SPF* computation starts when the DUT receives the update LSA.

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

$$N = \frac{R(R-1)}{2} + R + 4 \tag{4.1}$$

$$M = 2R(R-1) + 8 \tag{4.2}$$

It is important to notice that, in the considered study case, the number of edges M is proportional to the number of vertexes N, that is $M=O(N)$.

We report in Fig. 6 *switching time* as a function of N for packet length L_p=100 bytes and rate F_p=500 pps. The values obtained for the PC-based router with both Xorp and Quagga routing software are compared to the ones of a commercial access router, the Cisco 2801 [32]. Because *switching time* depends on considered destination network N_d, we report in Fig. 6 the measurement obtained in correspondence of the network determining the highest *switching time*. That occurs when N_d is chosen to be the last network inserted in the routing table.

From Fig. 6 we can notice that the worst results are obtained in the case of PC-based router equipped with Xorp routing software. For example for N=5050, *switching time* is 1 second, 11,18 seconds and 13,60 seconds in Cisco 2801, Quagga and Xorp respectively. By a more accurate analysis we argued that the high *switching time* in Xorp is due to the high time needed to update the routing table once the Dijkstra's algorithm has been executed and the best paths have been evaluated. This can be revealed by Fig. 7 where we report the *SPF* computation time as a function of N for Cisco 2801, Quagga and Xorp. Measurements have been performed according to the test methodologies described in [19-21]. Because for N=5050 the *SPF* computation time is 5,37 seconds, the PC-based router equipped with Xorp takes about 7 seconds to update the routing table.

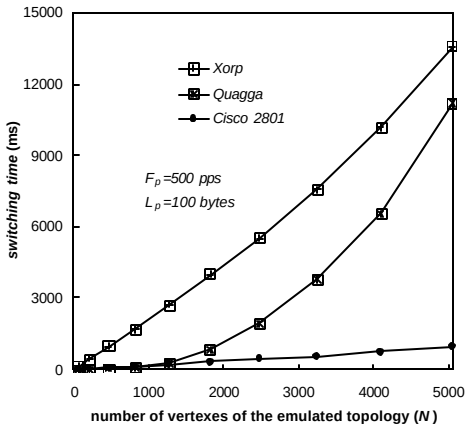

Fig. 6. *Switching time* as a function of the number of vertexes (N) in the emulated network topology for a Cisco 2801 router and a PC-based router. The test is performed choosing packet rate of F_p=500 pps and packet length L_p=100 bytes. The performance of Quagga and Xorp routing software have been evaluated in the PC-based router.

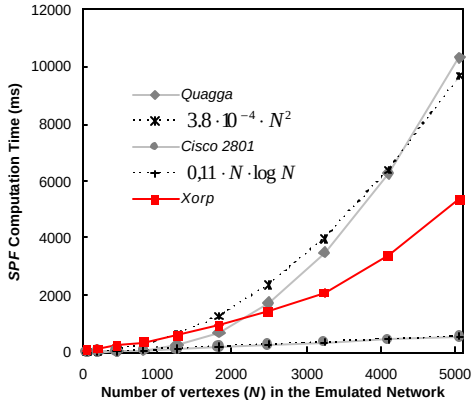

Fig. 7. *SPF* computation time as a function of the number of vertexes (N) in the emulated network topology for a Cisco 2801 router and a PC-based router. The performance of Quagga and Xorp routing software have been evaluated in the PC-based router.

The results obtained on a PC-based router equipped with Quagga routing software and the Cisco 2801 are quite different. Observing Figs 6 and 7, we can notice that, in this case, the main component of *switching time* is the *SPF* computation time. Comparing the two curves, it appears that Quagga performs better than the Cisco 2801 only when the number of routers R in the emulated topology is smaller than 45 corresponding to 1150 vertexes. The Cisco 2801 performed better for a larger number of routers. That is due to a sub-optimal implementation of Dijkstra's algorithm in Quagga. In fact it is possible to notice from Fig. 7 that the experimental measurements obtained for the Cisco 2801 router

closely fit the curves $0{,}11 \cdot N \log N$. This result is expected from the complexity of Dijkstra's algorithm [33,34]. On the contrary the results obtained for the PC-based router running Quagga are quite different and the measured values fit on the $3{,}8\ 10^{-4}\ N^2$ interpolating curve.

On the basis of these considerations we have optimized Dijkstra's algorithm to obtain performances comparable to commercial routers. Section 5 therefore will be dedicated to the analysis of the Dijkstra's algorithm complexity and to its optimization in Quagga.

5. Optimization of the *SPF* computation in Quagga

The *SPF* computation is based on the Dijkstra's algorithm, as described in [41]. 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 vertex in the graph. Each of these paths give raise to a spanning tree of the graph. In Quagga, the directed graph is represented by the LSA set, stored in the LSA database. During each iteration of the algorithm, every vertex is extracted from the graph and inserted into the spanning tree. Moreover Quagga also uses a Candidate List, which contains all the reachable vertexes that have not been inserted yet but that can be reached from vertexes already in the spanning tree. The Candidate List is used as a middle step 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 of the minimum cost path between the root and the vertex considered and passing through only vertexes already inserted in the spanning tree. According to 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, and its implementation is critical for the resulting global performances. The Candidate List performs four different functions: *Extract-Min*, which extracts the node with the minimum key from the Candidate List; *Insert* which inserts a node into the Candidate List; *Decrease-Key*, which updates the total cost associated with a particular node; *Lookup*, which finds a node whether it is stored in the Candidate List. In Section 5.1, we will evaluate the complexity of the Candidate List functions in Quagga. In Section 5.2, a binary heap data structure implementing the Candidate List will be proposed and its complexity will be evaluated. One difficulty with this proposal is that binary heaps do not provide efficient *Lookup* function. In Section 5.3 we will illustrate how this operation can be eliminated by modifying the LSA database data structure. In Section 5.4, the modified Quagga routing software will be evaluated.

5.1 Complexity Evaluation of Quagga

Let us consider the *SPF* computation for a graph with N vertexes and M edges. Each vertex is inserted and extracted from the Candidate List before it becomes part of the spanning tree, so both the *Insert* and the *Extract-Min* functions are performed exactly N times. The *Lookup* function is performed for each 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 evaluated because it depends on the costs of the links. However, it is always less than the number of edges, so we can assume that the *Decrease-Key* function is performed $O(M)$ times.

The average cost of each of these functions, depends on the data structure used to store the Candidate List. In Quagga, 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 through the list, to find the right place where to insert the new element. The cost of this operation is *O(N)* because the list contains *N* elements in the worst case. Both the *Lookup* and the *Decrease-Key* functions work in the same way, and have *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 $O(N^2)$ cost, the *Extract-Min* function has *O(N)* cost, the *Decrease-key* function has *O(M*N)* cost and the *Lookup* function has *O(M*N)* cost.

The total amortized cost of the implementation of Dijkstra's algorithm in Quagga is therefore no less than the sum of these four costs, *i.e.* no less than $O(N^2 + M*N)$. 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

We have modified Quagga original version and we have written a *patch* available in [42]. In particular in the new Quagga version we have chosen a binary heap to implement the Candidate List.

A binary heap is a complete and balanced binary tree with a local sorting [33,34]. 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 its two children. This particular sorting ensures that the node with the minimum key is the root of the heap.

The management of the tree is based on two internal functions: the *sift-up* and the *sift-down* functions. The *sift-up* function brings up a node with a low key toward the root node. It checks if the key of the node is lower than the one of its parent, and if this occurs it exchanges the node with its parent. The *sift-down* function, on the other side, pushes down a node with an high key toward the leaves of the heap. If the node has a key greater than at least one of the keys of its children, then it exchanges the node with the child with the lower key. These two functions perform the three main functions supported by the heap structure: *Insert*, *Extract-Min* and *Decrease-Key*.

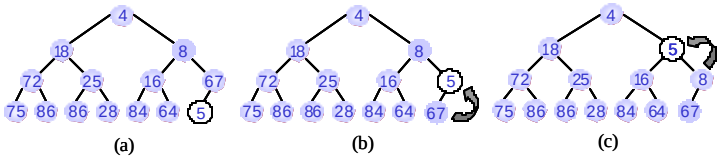

Fig.8. An example illustrating *Insert* function for a node with key 5. The node is inserted as leaf node (a) and two *sift-up* operations (b,c) complete the insertion.

The *Insert* function takes the new node and inserts it at the bottom of the heap, *i.e.*, makes it a leaf. Then it executes a *sift-up* procedure on the inserted node, and brings it up to its correct position. This is shown in the example reported in Fig. 8 where the

node with key 5 is inserted in the heap and two *sift-up* operations have been performed. Notice that because the *Insert* function needs at most *log(N) sift-up* operations, *i.e.* the maximum depth of the heap, its cost is *O(logN)*.

The *Extract-Min* function removes from the heap the node with the lowest key. This node is always at the root of the heap, and its extraction is almost costless. After this extraction, we have two sub-trees, which are themselves heaps, and need be merged into one single heap. To achieve this result, the *Extract-Min* function takes the last leaf of the heap and puts it at the root position, then executes the *sift-down* procedure on it and pushes it down to its correct position. This procedure is exemplified in the example reported in Fig. 9 where the node with lowest key 4 is extracted by the heap. The *Extract-Min* function executes at most *logN sift-down* operations, so its complexity is *O(logN)*.

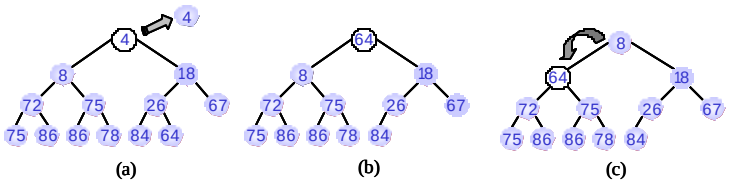

Fig.9. An example illustrating *Extract-Min* function for the node with lowest key 4. The node is extract (a), the last leaf node is inserted as root node (b) and a *sift-down* operation is accomplished (c).

The *Decrease-Key* function changes the key of a particular node to a lower value. Once the key value has been decreased, it executes the *sift-up* procedure on the node, and takes it to its new position. An example is shown in Fig. 10 where the key of a node is decreased from 16 to 5. Notice that to implement the *Decrease-Key* function, a number of *sift-up* operations at most equal to the maximum depth of the heap is required. For this reason the *Deacrese-Key* function cost is *O(logN)*.

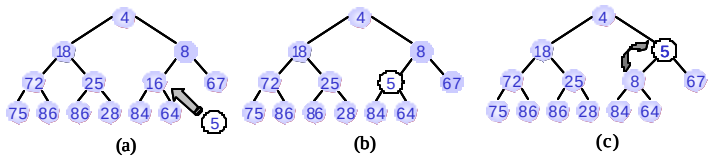

Fig.10. An example illustrating *Decrease-Key* function for the node decreasing its key from 16 to 5. The node is searched (a), the key is changed from 16 to 5 (b) and a *sift-up* operation is accomplished (c).

As described before, the heap can perform *Extract-Min*, *Insert* and *Decrease-Key* in *O(logN)* time. The amortized costs consequentially change to *O(NlogN)* for *Insert*, *O(NlogN)* for *Extract-Min* and *O(MlogN)* for *Decrease-Key*. In the following section we will show that the LSA database data structure can be modified to eliminate the need of the Lookup function making the tot al amortized cost of the new implementation of the Dijkstra's algorithm *O((M+N)logN)* in our modified Quagga routing software.

5.3 A New Solution to Optimize the Lookup Operation in the Candidate List

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 an *O(logN)* cost, we have modified the LSA database data structure so that the Lookup function becomes no longer needed. In particular for each LSA, stored in the database, we have added information denoting whether the LSA is in the Candidate List. In the 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 *sift-up* and the *shift-down* operations may change the position of a vertex in the Candidate List, a pointer to the information of the associated LSA is added to each vertex, so the LSA can be updated simultaneously. Further clarifications about the changes carried out in the LSA database data structure can be found in [42].

5.4 Numerical Results for Modified Quagga Routing Software

The measured values of the *switching time* on the modified Quagga version is presented in Fig. 11 varying the number *N* of vertexes in the graph. They are compared with the same measure taken on the Cisco 2801 and on the original Quagga version. We show the results for a packet length L_p=100 bytes and when the packet rate is F_p=500 pps.

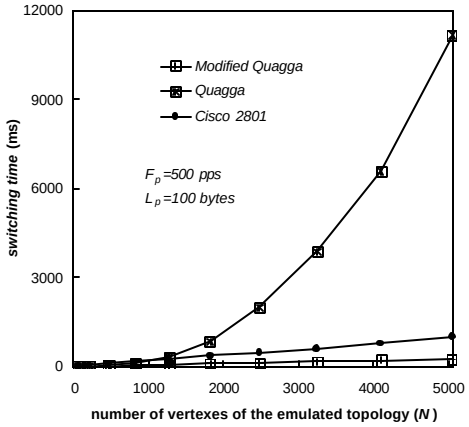

Fig.11. *Switching time* as a function of the number of vertexes (*N*) in the emulated network topology for a Cisco 2801 router and a PC-based router with Quagga and modified Quagga. The test is performed choosing packet rate of F_p=500 pps and packet length L_p=100 bytes.

From the results shown in Fig. 11 we notice that *switching time* on the modified Quagga version is always less than the time needed on the original version, proving the success of our optimization process. In particular the test evaluating the *SPF*

computation time [17,18] on the modified Quagga version, produced experimental results that perfectly reflect the *NlogN* trend, as shown in Fig. 12. Modified Quagga version allows the PC-based router to obtain performance better than Cisco 2801. In particular *switching time* in Cisco 2801 is four times higher than in modified Quagga when the number *N* of vertexes of the emulated network is 5050. In this cases *switching time* equals 1 second and 0,23 seconds for Cisco 2801 and modified Quagga respectively.

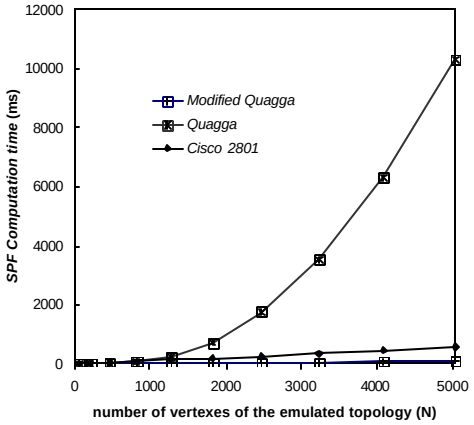

Fig. 12. *SPF* computation time as a function of the number of vertexes (*N*) in the emulated network topology for a Cisco 2801 router and a PC-based router with Quagga and modified Quagga.

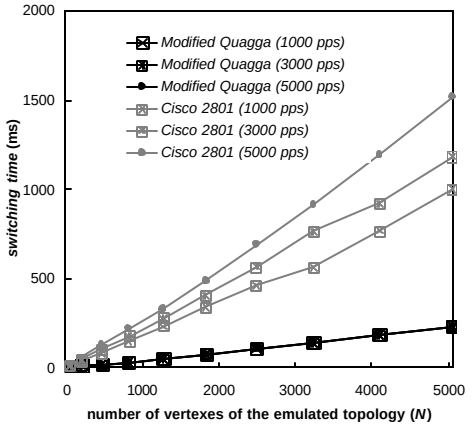

Fig. 13. *Switching time* as a function of the number of vertexes (*N*) in the emulated network topology for a Cisco 2801 router and a PC-based router with modified Quagga. The test is performed for packet length L_p=100 bytes and when packet rate F_p equals 1000 pps, 3000 pps and 5000 pps.

To study the influence of packet rate on *switching time*, we report in Fig. 13 *switching time* as a function of the number N of the vertexes when F_p equals 1000 pps, 3000 pps and 5000 pps. From the results reported in Fig. 13, we can also notice the performance of the Cisco 2801 depends much on the packet rate considered. For example when N=1830 *switching time* passes from 340 ms to 487 ms as packet rate increases from 1000 pps to 5000 pps. 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 routing table are reduced because a part of them are engaged to process and to route the IP data packets. In contrast, *switching time* increase is negligible for the PC-based router, because the processing capacities of the PC are higher than the ones of the Cisco 2801 router. As a result, packet processing does not influence the performance of the routing protocol significantly.

6. Conclusions

The aim of our work was to evaluate the routing performance of an OSPF router built by using a standard PC equipped with open source operating system and routing software.

The first experimental results for *switching time* obtained on a PC running GNU Quagga were not good, and therefore unsuitable for complex networks and high-level solutions. However our changes to Quagga code radically improved performance. The results obtained for *switching time* with our modified version of Quagga are better than the ones obtained with a high-end commercial router. This raises great expectations for building a competitive router using standard PC hardware.

In our future work, we plan to propose and implement dynamic Shortest Path Tree (SPT) algorithms [43] in Quagga Software. These algorithms do not compute a new SPT from scratch after changes in the link state of the network. On the contrary, they make use of previously evaluated structures to recompute the SPT. This results in considerable saving in CPU time, as long as the number of changes in state of the links is small.

7. References

[1] Bux W., Denzel W.E., Engbersen T., Herkersdorf A., Luijten R.P., "Technologies and building blocks for fast packet forwarding," IEEE Communication Magazine, Jan.2001, pp.70-77

[2] Keshav S., Sharma R., "Issues and trends in router design," IEEE Communication Magazine, vol.36, n.5, May 1998, pp.144-151

[3] Afek Y., Bremler-Barr A., Har-Peled S., "Routing with a clue," IEEE Trans. On Networking, Vol.9, n.6, 2001, pp.693-705

[4] B., Srinivasan V., Varghese G., "IP lookups using multiway and multicolumn search," IEEE Trans. On Networking, Vol.7, n.3, June 1999, pp.324-334

[5] K. Sklower, "A tree-based routing table for Berkeley Unix," presented at the 1991 Winter Usenix Conf., Dallas, TX

[6] C. DiBona, S. Ockman, M. Stone, "Open Sources Voices from the Open Source Revolution," O' Reilly Ed., Jan.1999

[7] The Open Source Initiative, www.opensource.org

[8] Click Modular Router, www.pdos.lcs.mit.edu/click/, MIT, Cambridge, MA

[9] Xorp, www.xorp.org, Berkeley Univ., CA

[10] Intel network processors, www.intel.com/design/network/products/npfamily/index.htm
[11] Linux Router Project, LRP, www.linuxrouter.org
[12] FREE ciSCO, www.freesco.org
[13] A. Bianco, J. M. Finocchietto, G. Galante, M. Mellia and F. Neri, "Open Source PC-Based Software Routers: A Viable Approach to High Performance Packet Switching," in Proc. of the third International Workshop, QoS-IP 2005, Catania, Italy February 2005
[14] BORA-BORA project, http://www.telematica.polito.it/projects/borabora/
[15] A. Bianco et al., "Click vs Linux: Two efficient open-source IP network stacks for software routers," in Proc. of the IEEE Workshop on High Performance Switching and Routing, Hong Kong, Italy, May 2005
[16] A. Bianco et. al., "Boosting the Performance of PC-based Software Routers with FPGA-enhanced Network Intaerface Cards," in Proc. of the IEEE Workshop on High Performance Switching and Routing, Poznan, Poland, June 2006
[17] V. Eramo, M. Listanti, N. Caione, I. Russo, G. Gasparro, "Router Performance of a Router Based on PC Hardware and Open Source Software," in Proc. of the IASTED International Conference on INTERNET AND MULTIMEDIA SYSTEMS AND APPLICATIONS, Grindelwald (Switzerland), 21-23 February 2005
[18] V. Eramo, M. Listanti, N. Caione, I. Russo, G. Gasparro, "Optimization in the Shortest Path First Computation for the Software Routing GNU Zebra," IEICE Transactions Communications, vol. E88-B, no. 6, June 2005, pp. 2644-2649.
[19] V. Manral, R. White, A. Shaikh, "Benchmarking Basic OSPF Single Router Control Plane Convergence," RFC 4061 , April 2005
[20] V. Manral, , R. White, A. Shaikh, "OSPF Benchmarking Terminology and Concepts," RFC 4062, April 2005
[21] V. Manral, , R. White, A. Shaikh, "Consideration When Using Basic OSPF Convergence Benchmarks," RFC 4063, April 2005
[22] J. T. Moy, "OSPF: Anatomy of an Internet Routing Protocol," Addison Wesley, January 1998
[23] C. Huitema, "Routing in the Internet," Prentice Hall, 1st Edition, 1995
[24] A. Basu and J. G. Riecke, "Stability Issue in OSPF Routing," in Proc. ACM SIGCOMM Computer Communication Review, Vol. 31, no. 4, October 2001
[25] A. Shaikh and A. Greenberg, "Experience in Black-box OSPF Measurements," in Proc. ACM SIGCOMM Internet Measurement Workshop (IMW) 2001, pp. 113-125, Nov 2001
[26] Next Generation Network Test Methodologies, Spirent Communication, http://www.spirent.com/ documents/775.pdf
[27] OSPF Conformance and Performance Testing, Ixia Communication, http://www.ixiacom.com/ test_plans/pdfs/ospf.pdf
[28] V. Eramo, M. Listanti and A. Cianfrani, "Switching Time Measurement and Optimisation Issues in GNU Quagga Routing Software," IEEE Globecom 2005, St. Louis (USA), December 2005
[29] L. Torwalds, Linux, http://linux.org
[30] Xorp Routing Software, http://www.xorp.org
[31] GNU Quagga Routing Software, http://www.quagga.net/
[32] www.cisco.com

[33] S. Saunders, "A Comparison of Data Structures for Dijkstra's Single Source Shortest Path Algorithm," http://www.cosc.canterbury.ac.nz/research/reports/ HonsReps/ 1999/ hons_9907.pdf
[34] A. V. Goldberg and R. E. Tarjan, "Expected performance of Dijkstra's Shortest Path algorithm," Technical Report 96-062, NEC Research Institute, Princeton, NJ, June 1996
[35] Baker F., "Requirements for IP version 4 routers," RFC 1812
[36] RUDE/CRUDE Traffic Generator, http://rude.sourceforge.net/
[37] Network Emulation Software Brite, http://www.cs.bu.edu/brite/download.html
[38] A. Medina, A. Lakhina, I. Matta, and J. Byers, "BRITE: An Approach to Universal Topology Generation," in Proc. Ninth International Symposium in Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'01), IEEE Computer Society, Cincinnati, August 15-18 2001
[39] A. Medina, A. Lakhina, I. Matta, and J. Byers, "On the Origin of Power Laws in Internet Topologies," ACM Computer Communication Review, April 2000, vol. 5
[40] LSA Generator Software SPOOF, http://www.cs.ucsb.edu/~rsg/Routing/download.html
[41] J. Moy, "OSPF Version 2," RFC 2328, April 1998
[42] Routing Software *Patch* GNU Quagga 0.97, http://net.infocom.uniroma1.it/projects/progetti_dip/zebra/
[43] P. Narvàez, K.Y. Siu and H.Y. Tzeng, "New Dynamic Algorithms for Shortest Path Tree Computation," IEEE/ACM Transactions on Networking, Vol. 8, n. 6, December 2000