

Enhancing Simulation Mode Operation of *ospfd*

W. Xie, B. Polepalli, M. Goyal, H. Hosseini
University of Wisconsin - Milwaukee
Milwaukee, WI 53201
Email: wxie,brpj,mukul,hosseini@uwm.edu

Abstract

In this paper, we describe our modifications to *ospfd*, a popular open-source implementation of OSPF routing protocol. The *ospfd* software can also be run in simulation mode, where processes, representing individual routers, communicate with each other over UDP sockets. Even though *ospfd* provides a fine-granularity implementation of OSPF protocol, its simulation mode operation leaves a lot to be desired. The main weakness of *ospfd*'s simulation mode operation is its lack of distinction between the real time and simulated time. The simulated time is represented in terms of ticks and increases at the same rate as the real time, i.e. the duration of an individual tick is fixed (e.g. 50ms). Consequently, the amount of work done by a router process in a given tick depends on how long this process has access to CPU during this tick. In our modifications, we introduced separation between the real time and simulated time by assigning specific (simulated) time durations with individual protocol tasks such as routing calculations and processing of OSPF packets (Hello, LSUpdate etc.). In modified *ospfd*, a tick gets over only when all router processes have completed the work they are supposed to do during the current tick. Additionally, we introduced several other modifications such as event based advancement of simulated time, script-based specification of topology change events and allowing router processes to run on multiple machines.

Index Terms

OSPF, Simulation, *ospfd*

1. Introduction

OSPF [1] is a popular *interior gateway protocol* used for routing with in an *autonomous system* in Internet. Given its wide deployment, OSPF receives considerable attention from networking research community looking to improve various aspects of its operation. Simulations are often the preferred mechanism for analyzing OSPF operation in large scale networks. Several networking simulation tools, such NS2 [2], GTNetS [3], SSFNet [4] and OPNET [5], now provide simulation modules for OSPF. Additionally, several

open-source routing software suites are available [6], [7] that can be used to turn ordinary *PCs* into sophisticated routers. These software provide fine-grained implementation of OSPF protocol but obviously can not be used to study the operation of the protocol in very large networks. In this paper, we focus on one such software, called *ospfd* [8], [9], which has a *dual* mode operation. *ospfd* can be run on a computer as a *daemon* converting the computer into an OSPF router. Additionally, *ospfd* can also be run as a process simulating the OSPF operation in a router. Several such router processes, running on a machine and sending OSPF packets to each other over UDP sockets, can be used to simulate OSPF operation in a large scale network.

The *ospfd* software, with its fine-grained implementation of OSPF protocol and the ability to simulate large scale OSPF networks, can serve as an immensely useful tool for OSPF related research. However, the actual *simulator* mode operation of *ospfd* leaves a lot to be desired. In the following, we identify the main deficiencies in *ospfd*'s simulation mode operation that prevent its use in analyzing OSPF operation in large scale networks:

- The simulation mode operation of *ospfd* does not make any distinction between the *real* time and *simulated* time. The simulated time is represented in terms of *ticks* and increases at the same rate as the real time, i.e. the duration of an individual *tick* is fixed (e.g. 50ms). Consequently, the amount of work done by a router process in a given *tick* depends on how long this process has access to CPU during this *tick*. If the number of simulated routers is large, an individual router may not get a chance to execute at all for several ticks, thereby leading to incorrect results.
- The lack of distinction between the simulated and real time also means that it is not possible to increase the simulated time to the time when the next OSPF event (e.g. firing of a timer or processing of a waiting packet) is scheduled to occur in the simulated routers.
- The topology changes (e.g. a link/router up/down events) can only be enforced through a *graphical user interface*. It is not possible to specify the exact instant in simulated time when a topology change should take place.
- All simulated router processes must run on the same

machine, thereby making it difficult to simulate networks consisting of large number of routers.

In this paper, we describe our modifications to *ospfd* to eliminate the problems listed above. In our modifications, we introduced separation between the real time and simulated time by assigning specific (simulated) time durations with individual OSPF tasks such as *SPF calculations* and processing of OSPF packets (e.g. *Hello, LSUupdate* etc.). In modified *ospfd*, a *tick* gets over only when all router processes have completed the work they are supposed to do during the current *tick*. Additionally, we introduced *event based advancement* of simulated time, script-based specification of topology change events and allowing router processes to run on multiple machines. The modified *ospfd* software is publicly available [10].

The rest of the paper is organized as follows. Section 2 describes *ospfd*'s original simulation mode operation. Section 3 describes various modifications that we have introduced. Finally, Section 4 concludes the paper.

2. Original Simulation Mode Operation of *ospfd*

The simulation mode operation of *ospfd* involves a *controller* process, *ospf_sim*, spawning the *router* processes, *ospfd_sim*, as per a configuration file. The controller process communicates with the router processes over a TCP socket (see the *simulation control* path in Figure 1). Each router process opens a UDP socket (the *data packet port* in Figure 1) to communicate with other routers. The controller process ensures that a router knows about data packet ports of other routers. Additionally, each router process has a *monitoring port*, which is a TCP socket that allows a *monitor* process to access individual routers.

As mentioned earlier, the controller process and all the simulated router processes run on the same machine. All the simulated router processes are spawned at the beginning of the simulation. The options to stop/restart a router and bring an interface down/up can be performed only using a Tcl/Tk based *graphical user interface* (GUI) operated by the controller process. Thus, it is difficult to precisely control the timing of the topology change events and to cause several topology changes to take place simultaneously. The GUI displays a map of the OSPF network being simulated. The state of database synchronization between simulated routers is represented in the GUI via a color-coding scheme.

The controller process advances the simulation time by sending *ticks* to the simulated routers at regular intervals. The work performed by a router process during a *tick* depends on the CPU time allocated to this process during the *tick* duration. There is no provision to ensure that all router processes get equal access to the CPU during a *tick*.

3. Modifications to *ospfd*'s Simulation Mode Operation

As mentioned earlier, we introduced significant modifications to the simulation mode operation of *ospfd*. In the following, we describe these modifications.

3.1. Allowing router processes to run on multiple machines

The need to run all router processes on the same machine makes it difficult to simulate very large OSPF networks using *ospfd*. We modified the simulator to allow the router processes to run on multiple machines. Each machine, used in the simulation, runs a local controller process called *ospf_sim_server*. In order to spawn a router process on a particular machine, the main controller process (*ospf_sim*) sends a request to the *ospf_sim_server* process running on that machine, which then spawns a router process. The termination of a router process is executed in a similar manner. Thus, the *ospf_sim_server* process acts as the interface between the main controller process and the router processes in order to effect topology changes. The router processes still interact with the main controller process via TCP sockets in order to obtain the configuration information and with each other via UDP sockets. The modified architecture is shown in Figure 2.

3.2. Script-based specification of topology change events

The original *ospfd* allows the network topology - the routers, the links and the propagation delays on the links - to be specified in a configuration file. Our modifications allow the topology change events to be specified in the configuration file as well. The event specification in the configuration file includes the time, at milliseconds granularity, at which the event should take place. The main controller process reads the configuration file and stores the topology change events in an AVL tree such that the earliest event is always at the head of the tree. The main controller process fires a topology change event when the simulation time reaches the event time. This feature not only allows exact specification of the time at which a topology change should take place but also enables simulation of *shared risk link groups*, the group of routers and links that fail together at the same time instant or in quick succession. At present, the topology change events can only be specified for routers and *point-to-point* links between them, i.e. it is not yet possible to start/stop *broadcast/NBMA* links.

Various topology change events, that can be specified in the configuration file after our modifications, are as follows:

- *startrouter*: In addition to the time at which this event should occur, the *startrouter* command also specifies the

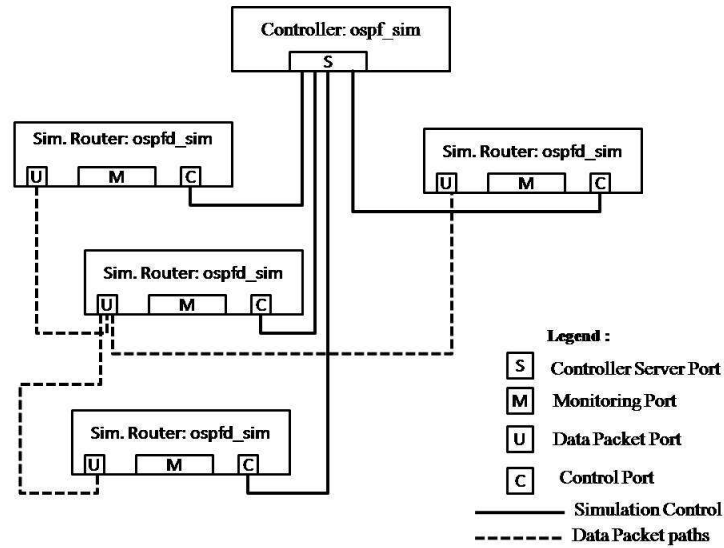


Figure 1. Process Architecture of Original Simulator

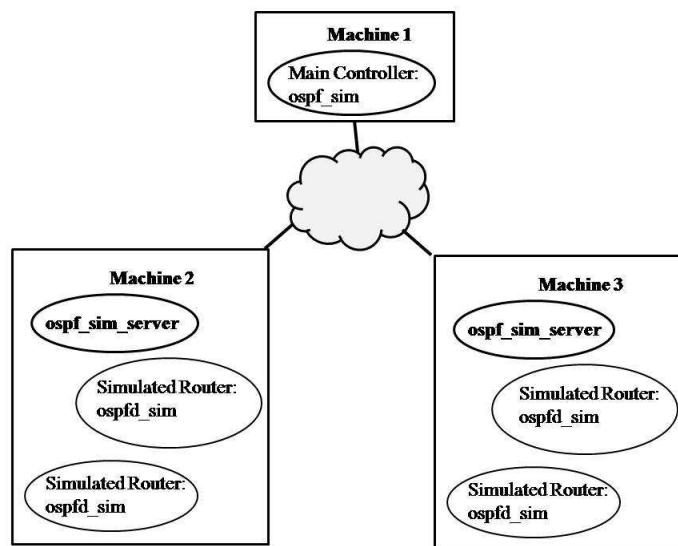


Figure 2. The modified process architecture for simulation mode operation of *ospfd*

id of the simulated router and the IP address of the machine where the router process should be spawned. The firing of this event causes the main controller process to send a request to the local controller process running on the specified machine to spawn the router process. The router process opens a TCP socket to the main controller to download its configuration and to perform all future communication with it. The router process also opens a UDP socket to receive OSPF packets from other router processes that are its neighbors in the simulated network topology.

- *stoprouterabnormal*: The firing of this event causes the main controller process to send a request to the local controller process running on the machine where the specified router process is running to terminate the router process abruptly. Abrupt termination of the router process is achieved by the local controller sending a *SIGKILL* signal to the router process. After killing the router process, the local controller sends a confirmation message to the main controller process.
- *stoproutergraceful*: The firing of this event causes the main controller process to send a request to the local

controller process running on the machine where the specified router process is running to terminate the router process in a graceful manner. This is achieved by the local controller sending a *SIGUSR1* signal to the router process. The router process catches the signal and takes appropriate actions to terminate gracefully. Once the router process has died, the local controller sends a confirmation message to the main controller process.

- *startlink*: The firing of this event causes the main controller process to ask the router processes at the two ends of the link to consider the link as *up*.
- *stoplink*: The firing of this event causes the main controller process to ask the router processes at the two ends of the link to consider the link as *down*. An argument to the command determines if the OSPF operation in the router process immediately considers the link to be down (*hardware failure detection*) or lets the Hello protocol to detect the failure of the link.
- *stopall*: The firing of this event causes the main controller process to ask all local controller processes to terminate abruptly all the router processes they control. The local controllers kill themselves after killing all the router processes they control. The main controller process terminates as well.

3.3. Advancing simulation time

The original *ospfd* has the simulated time advance at the same rate as the real time. Since the amount of work done by a router process during a given time interval depends on the CPU time allocated to the process during this interval, we often encountered errors in simulations resulting from the fact that different router processes got CPU access for different times during the simulation. The errors become bizarre as the number of router processes in the simulation increases and consequently the time duration for which an individual router process has access to the CPU decreases.

We modified the simulation mode operation of *ospfd* to introduce separation between the real time and simulated time. This was achieved by assigning specific (simulated) time requirements with individual OSPF tasks such as *SPF calculations* and processing of OSPF packets (e.g. *Hello*, *LSUpdate* etc.). For example, assigning a time duration 10ms to an SPF calculation means that a router is considered *busy* and hence unable to perform any additional task for 10ms once it initiates an SPF calculation. Rather than increasing the simulated time at the same rate as the real time, the main controller process waits for all the router processes to acknowledge that they have finished all the work they had to do in the current millisecond before advancing the time further.

Suppose a router process gets a new *tick* message from the main controller advancing the simulated time to t ms.

If the router considers itself *busy* until $t + x$ ms completing the work it had initiated earlier (e.g. an SPF calculation), it would immediately send an acknowledgement to the main controller asking it to increase the simulated time to $t + x$ ms. If the router is not busy at the time the new tick arrives or its *busy* state expires before $t + 1$ ms, the router performs as many tasks (e.g. processing pending packets and firing timers) as possible until simulated time $t + 1$ ms. Then the router checks if it still has some pending packets to process. In that case, it sends an acknowledgement to the main controller asking the simulated time to be increased to $t + 1$ ms. If the router does not have any pending packets to process, it checks the earliest time, say y ms, at which a local timer (e.g. the Hello timer) must fire and sends an acknowledgement to the main controller process asking it to increase the simulated time to y ms.

As the main controller receives the acknowledgements to the last tick from all the router processes, it determines how much should the simulated time be increased. This decision not only considers the time increase suggested by individual routers in their acknowledgements to the previous tick but also the time at which the next topology change event should take place. If the simulated time can be increased to the time at which the next topology event takes place, the main controller fires this topology change event. In any case, it generates the new tick message containing the updated simulated time and sends it to all the router processes.

Note that the simulation time advancement, as described above, means that the modified *ospfd* allows the OSPF operation in a network to be simulated with a time granularity of 1 ms.

4. Conclusion

In this paper, we have described the modifications we have made to the simulation mode operation of *ospfd*, a fine-grained and open source implementation of OSPF routing protocol. The original simulation mode operation of *ospfd* has severe deficiencies. There is no distinction between the simulated time and real time. The amount of work done by a router process during a given time depends on the CPU time allocated to this process by the operating system on the machine running the simulation. This often causes incorrect simulation results. Further, it is not possible to specify the exact instant in simulated time when a topology change should take place. Finally, all router processes must run on the same machine, thereby making it difficult to simulate large networks.

We introduced significant modifications to the simulation mode operation of *ospfd* to eliminate the problems listed above. We introduced distinction between the real time and the simulated time, allowed specification of topology change events in a configuration file and also allowed router processes in a simulation to run on multiple machines. The

modified *ospfd* code is publicly available [10] and is being used for OSPF research [11].

References

- [1] J. Moy, "OSPF version 2," Internet Engineering Task Force, Request For Comments (Standards Track) RFC 2328, April 1998.
- [2] "The network simulator - ns2," <http://www.isi.edu/nsnam/ns>.
- [3] "The Georgia Tech network simulator - GTNetS," <http://www.ece.gatech.edu/research/labs/MANIACS/GTNetS/>.
- [4] "Scalable simulation framework," <http://www.ssfnet.org/homePage.html>.
- [5] "Opnet technologies," <http://www.opnet.com>.
- [6] "Quagga software routing suite," <http://www.quagga.net/>.
- [7] "GNU Zebra routing software," <http://www.zebra.org/>.
- [8] J. Moy, "OSPF routing software resources," <http://www.ospf.org>.
- [9] J. T. Moy, *OSPF Complete Implementation*. Addison-Wesley, 2001.
- [10] M. Goyal, S. Bhaasadwaj, S. Venkatesh, and M. Soperi, "A distributed OSPFD simulator," <http://www.cs.uwm.edu/~mukul/newospfd.html>.
- [11] M. Goyal, W. Xie, M. Soperi, H. Hosseini, and K. Vairavan, "Scheduling routing table calculations to achieve fast convergence in ospf protocol," in *Proc. IEEE Broadnets*, 2007.