

# Real-Time OSPF Route Monitoring

Emmanuel Baccelli<sup>1</sup> and Raju Rajan<sup>2</sup>

## **Abstract:**

*In this paper, we present the design and the implementation of an OSPF route monitoring service hooked up to a managed IP network, enabling the administrator to be aware in real-time of the routes actually used throughout the OSPF domain. Such a service represents an advance over tools available today that offer off-line emulations of routing protocols, or that track network routing behavior in a local, semi-static manner. The real-time route monitoring service has applications to network management functions such as QoS service fulfillment, performance analysis and SLA verification.*

## 1. Introduction

The recent explosion of investment in high-technology is premised on the ubiquity of the internet and commercial success of new services offered over it. Extrapolating from current trends it is reasonable to forecast continued changes in the Internet along several dimensions (a) the number of customers and applications (b) volume and diversity of traffic (c) number and heterogeneity of network devices, (d) speeds and functionalities incorporated in network devices and (e) the continued increase in new services delivered over IP networks.

How do these changes impact network management? The traditional goals of network management -- long-term provisioning and crisis management -- are required to perform at greater speeds and scale. Further, managing new services such as VoIP or VPNs require a paradigm qualitatively different from traditional device management. Operators have to incorporate network-wide and service level abstractions into their worldview. In real-time, ISPs and large enterprise networks need to collect and process network state information in their IP networks for a variety of different purposes – provisioning and dimensioning, fault detection and diagnosis, accounting, SLA reporting and so on. But there are some fundamental obstacles to upgrading network management to face these new challenges. The suite of protocols that constitute IP have been designed to be distributed in their operation and reactive to changes in network topology. While these attributes have contributed to the overwhelming success of the Internet, they also make network management much more difficult due to the absence of centralized information.

In this paper, we address a fundamental component of IP network state – routing state – and describe how it may be monitored in real-time. What is an IP route? In contrast to network technologies such as Frame Relay or ATM, IP is based on datagrams and not on connections. In an IP network, successive similar packets do not necessarily take the same path through the network. An IP *route* is just a convenient way of describing instantaneous forwarding behavior in the network; for now, we treat it as the sequence of successive (router) interfaces visited by an IP packet, starting at some interface at a given time. Needless to say, routes depend on evolving states at different routers in the network, and consequently on specialized *routing protocols* that distribute topology, resource availability and network status.

While the distributed nature of IP routing has contributed to the enormous success of the Internet, it is a serious impediment to obtaining dynamic resource usage and traffic flow information at a central location. The ability to react and report network performance and failure depend on systems that can composite and analyze network behavior in real-time. Routing is obviously an important component of such behavior, and the ready availability of timely routing state information can dramatically improve the processes of provisioning, billing, fault monitoring and SLA reporting/verification.

In this paper, we describe the design and implementation of an IP *route monitoring system* for networks that use Open Shortest Path First (OSPF) routing protocol. This system gathers topology and network state information to track or estimate routes within an OSPF domain. The real time monitoring provided by our system represents

---

<sup>1</sup> Institut Eurécom, Sophia Antipolis. The work described in this paper was done during this author's visit to AT&T Labs Research. Email [baccelli@eurecom.fr](mailto:baccelli@eurecom.fr).

<sup>2</sup> Corresponding author, AT&T Labs Research, 180 Park Avenue, Florham Park, NJ 07932. Email [rajan@research.att.com](mailto:rajan@research.att.com).

a considerable innovation over the state of the art – tools available today offer either off-line simulations/emulations of routing protocols, or work on infrequently retrieved network state to perform coarse predictions. On the other hand, here we propose the design and the implementation of a system which is hooked up to the managed network, enabling the user/administrator to be aware in real-time of the routes actually used throughout the domain. The system we have implemented works with OSPF alone; we speculate that our design and methods have considerable bearing on other routing protocols as well.

The structure of the paper is as follows: we will first of all give a brief overview of OSPF. Then we will describe the different approaches to route monitoring, and the trade-offs that are encountered. Finally, we present details on our implementation of the OSPF route monitoring system.

## 2. OSPF Route Monitoring

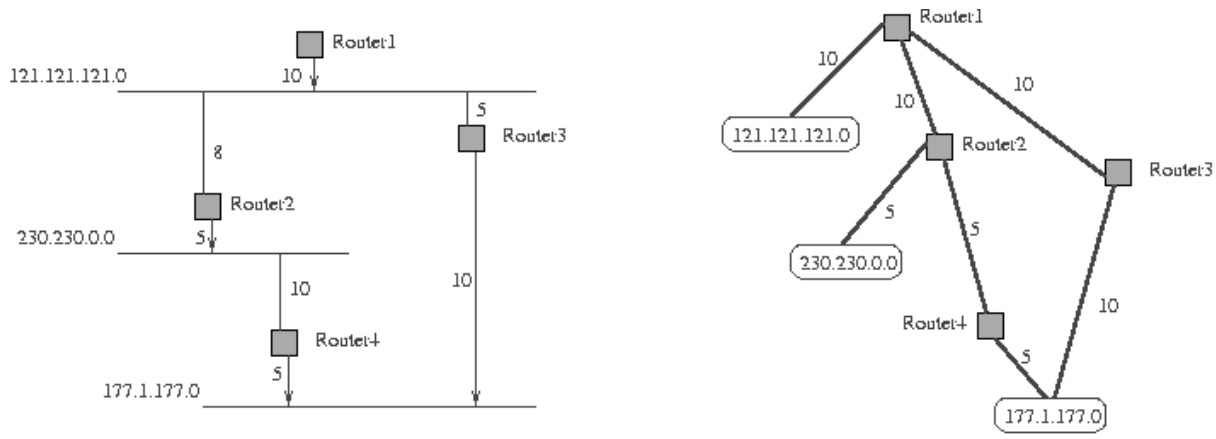
When an IP packet arrives at an interface on a router its next hop (or outgoing interface) is determined by a *forwarding* table at the router. Consequently, the path taken by the packet depends on forwarding tables at various routers in the network. How are forwarding tables updated to reflect the changing state of the network? This is the job of the *routing protocol*, used to disseminate network state information. As it is infeasible for every router to learn the state of every piece of the network, the Internet is divided into different Autonomous Systems (or *domains*) administrated independently from the one another (see Fig. 2). There are two kinds of routing protocols - inter-domain routing protocols or *Exterior Gateway Protocols* (EGPs) such as BGP used for exchange of state information across domains, and intra-domain routing protocols or *Interior Gateway Protocols* (IGPs) such as OSPF, IS-IS, RIP or EIGRP, used for spreading state within a domain. This paper focuses on intra-domain routing, with OSPF. In the following section, we present a simplistic description of OSPF, focussed less on minute protocol details than broad brush summaries of routing information exchange required for later discussion on the design and implementation of our route monitoring system. For further reading on OSPF see [3] and [5].

### 2.1 Overview of OSPF

OSPF (Open Shortest Path First) was first developed as a scalable replacement for another IGP called RIP. A domain using OSPF as its IGP is may be sub-divided into *areas*. A special central area called area 0 or the *backbone area* is connected to all other areas as a hub to spokes. All areas communicate with one another across the backbone (Fig 3). Each router interface belongs to exactly one area, and communicates with a neighboring interface in the same area over a (unidirectional) *link* configured with a *weight* representing the cost of communication over that link. A router with all its interfaces in the same area is called *Internal Router* (IR), while another with interfaces in multiple areas is an *Area Border Router* (ABR). A router that acts as a gateway between OSPF and other routing or other instances of the OSPF routing process is an *Autonomous System Border Router* (ASBR) (see Fig 3.)

Within An Area : Each router generates *Link State Advertisements* (LSAs) describing the state of incident links, roughly speaking, the state of its connectivity with neighboring routers. LSAs are triggered by state changes at the router and flooded reliably throughout the OSPF area. Thus, every router with an interface in an area compiles the same Link State Database (LSD). Using this database, each router computes its forwarding table to destination networks within the area by calculating its shortest path tree (with respect to link weights) using the Dijkstra algorithm with the root of the tree being the calculating router itself (see Fig. 1). In a nutshell, the essential aspects of OSPF within an area are:

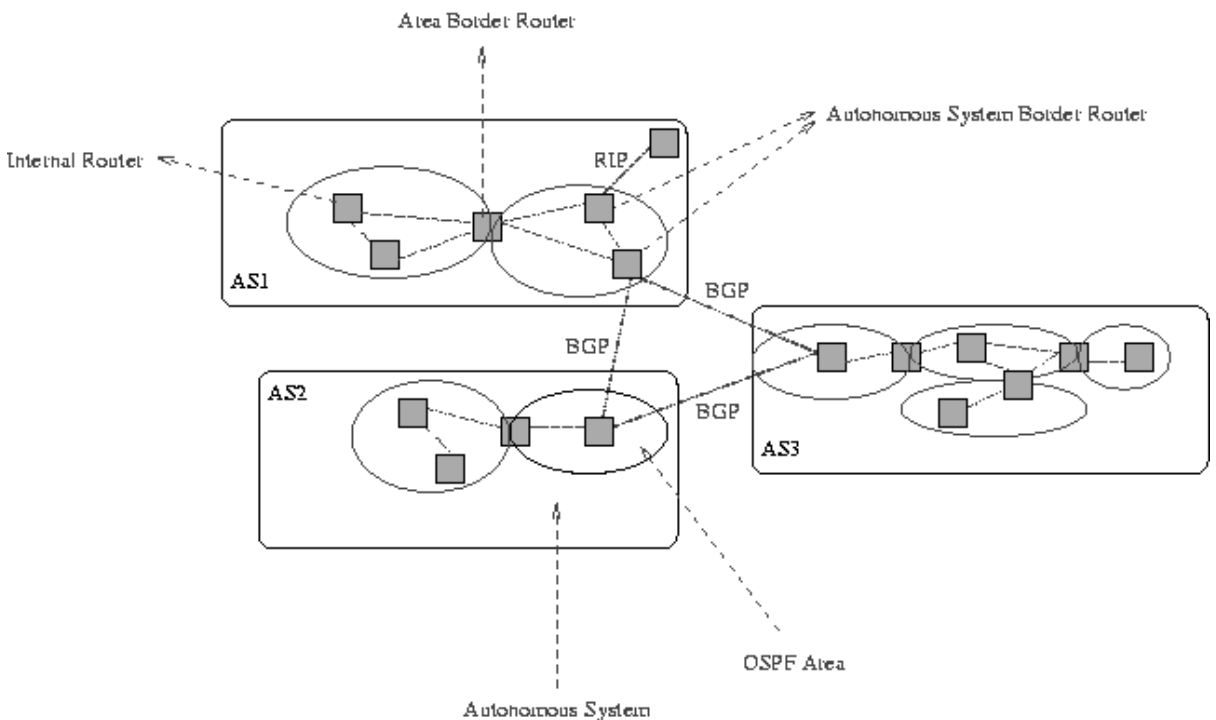
1. Upon initialization, a router generates LSAs (packets describing the state of incident links on that particular router).
2. LSAs are reliably flooded throughout the area.
3. Each router within the area uses its awareness of the network state to update its forwarding table.
4. Changes to the state of a link result in new LSAs being generated by an adjacent router, which are reliably flooded through the area, resulting in re-computations of forwarding tables.



**Figure 1: Example of Shortest Path Tree with Router1 as root node**

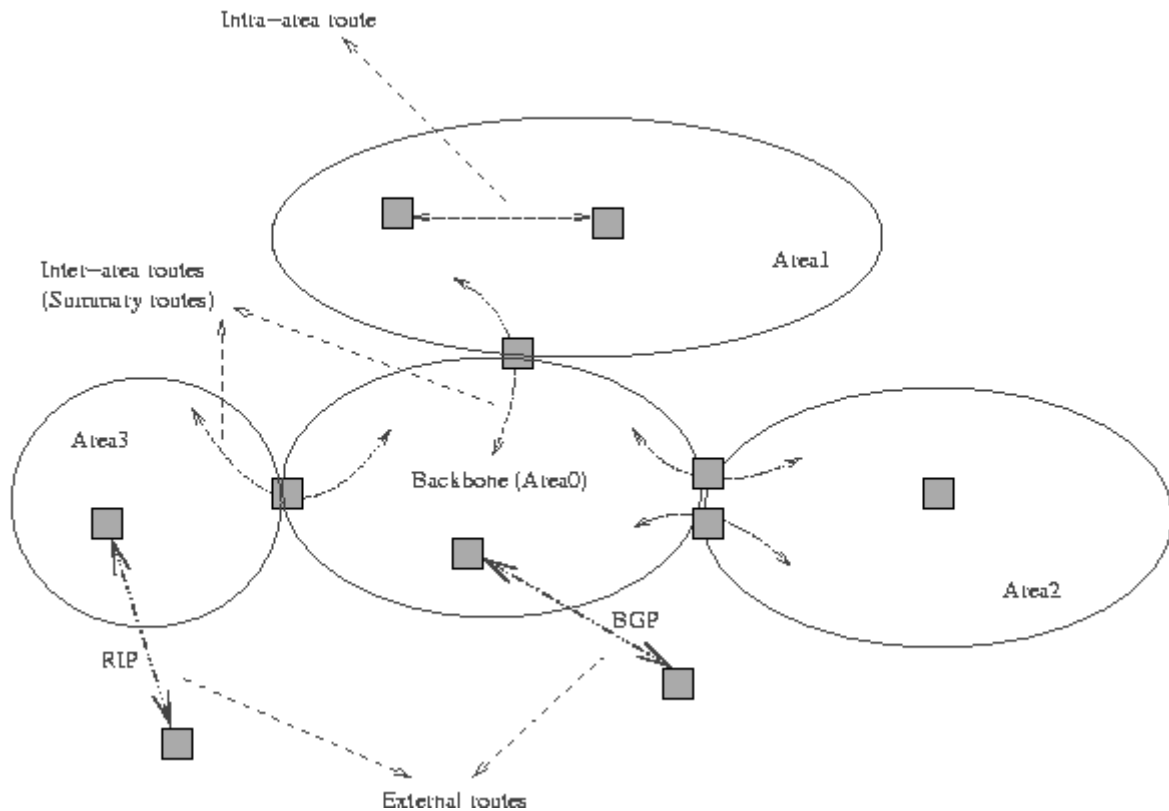
Across Areas: Any change in a single link is propagated throughout its area using flooding as described above. But propagating these changes across area boundaries without some filtering could be disastrous for the network. Hence, state exchanges across areas uses *route summarization* or *route aggregation*. To briefly sketch the operation of OSPF across area boundaries:

1. Each area border router (ABR) aggregates<sup>3</sup> together a number of destinations within the area, and advertise a route to the aggregate with a weight that is set to the maximum weight of the paths from the ABR to each one of the destinations. This aggregated information, known *summary LSAs* are injected into the backbone.
2. The summary LSAs generated by each ABR are reliably flooded throughout the backbone. Therefore all ABRs builds identical summary libraries.
3. Each ABR reliably floods selected LSA summaries from other areas within its own (non-backbone) areas.
4. Each IR builds up its forwarding tables using the external routes obtained from these summaries.



**Figure 2: Internet as Autonomous Systems**

<sup>3</sup> OSPF standards don't describe the way aggregation should be done, so different levels of summarization have been implemented. The first step is to aggregate by subnet, but starting from that, one can aggregate as much as is convenient.



**Figure 3: OSPF Area Splitting and Router Denomination**

Actually, OSPF turns out to be even more complex if one tries to fully understand the broadcast mechanisms, the use of virtual or the adjacency mechanisms and so on. However, the above overview will suffice for this exposition.

### 3. Issues in Designing an OSPF Route Monitoring System

From the discussion in the above section it should be clear that the process of OSPF route distribution and re-computation is highly dynamic and distributed. The notion of an instantaneous route is abstract, and is, at best, an approximate answer to a hypothetical question, i.e., what will be the sequence of interfaces traversed by a packet with a specific destination address, starting from a given interface? Such a question may not have a unique answer even in a real network – for instance, some routers forward packets with the same destination address to different outgoing interfaces in order to balance loads<sup>4</sup> across paths with the same cumulative weight. Given these constraints, what we seek is the ability to track the forwarding behavior of the network “as closely as possible”. In this section, we first put forward a clear set of requirements for the route monitoring system, and then examine different solutions that meet these requirements, and discuss their strengths and drawbacks.

#### 3.1 Requirements

We first require that the route monitoring system support the following two queries.

- a) **Query\_route(src\_if, dest\_if)** Given an origin interface `src_if` and a destination interface `dest_if` in the OSPF domain, the route monitoring system must return a potential set of routes, starting at the origin and culminating at the destination interface. We represent the set of routes as a directed graph of successive interfaces, originating at the source interface and terminating at the destination interface. The uncertainty about the routing state is reflected in the use of a graph rather than a singular *sequence* of interfaces. Naturally we would prefer an implementation with as little uncertainty, i.e., by as sparse a graph as possible.
- b) **Monitor\_route()** This is a persistent version of query (a) and calls for the route monitoring system to return an initial set of routes, and thereafter update the inquiring entity of any changes to this route. This persistent

<sup>4</sup> Load balancing is a vendor specific feature and, in general, even when the full packet header is specified, it is near impossible to predict which of two equal cost routes would be chosen by a router.

query is useful for management applications that wish to be notified when particular route changes could potentially impact an ongoing operation. The notifications are issued through a generic `update_route` event consumed by the management application.

The route monitoring system accepts two inputs – a semi-static *topology*, as well as dynamic network state *updates*. Our definition of topology here refers to relatively unchanging aspects of the OSPF domain -- the set of routers and interfaces, adjacencies, link weights, area assignments, etc. As there are several ways of acquiring such information, we require that the route monitoring system use a generic interface to a *topology module*, and not depend on the mode of acquisition of configuration information.

Further, any route monitoring system is designed to acquire network state information and react to changes dynamically. The nature of state information could vary with different approaches to route monitoring – however, it helps to measure these against common criteria. We require that any solution be evaluated in terms of its communication and processing overhead for point and persistent queries. Further, non-intrusive techniques that do not impact the performance of the routers are more desirable. Different approaches also scale differently as the size of the network increases, or as we demand that routes reflect more timely information. Below, we examine five different route-monitoring methodologies based on distinct information choices, and evaluate them based on the above criteria.

### 3.2 Methodological Choices and Tradeoffs

- a) **Traceroute:** A naive way to answer a `query_route()` is to run the `traceroute` utility from the origin interface/router to directly obtain the route to the destination. Naturally, the persistent `monitor_route()` query will involve periodically re-running `query_route()` to figure out if routes have changed. The response time of a monitoring system based on `traceroute` will be large. Further, this method will only discover one of several possible routes through the network, and will miss alternative routes that ICMP packets themselves never take. At best, this approach can be used as a sanity check to verify the accuracy of other techniques
- b) **Downloading Forwarding Tables:** Another straightforward way is to periodically connect to each router in the domain via SNMP [1] [4] [6] and simply download its forwarding tables. Concatenating these tables, one can get the actual end-to-end routes. While this solution is easily deployable, and has a relatively small computational overhead, it has the large communication overhead of any polling solution. Older routers do not support bulk SNMP requests, and repeated queries generate network traffic while impacting the performance of the router. Further, as we seek to make the tracking more current routers have to be queried at higher frequencies, irrespective of the rate of change of state in the network. Consequently, this solution does not scale well with the size of the network, at a fixed level of link state dynamism.
- c) **Downloading Link State Databases:** We can use SNMP to download instead the link state database of various routers, and use these to emulate the behavior of OSPF. The advantage of this technique over downloading forwarding tables is its smaller impact on router and network performance – we need to query only one router in each area, as all others have identical databases. However, the computational overhead of this solution is much greater, as the route monitoring system has to redo computations to figure out the resulting forwarding behavior at various nodes. Moreover, this too is a polling technique, and results in considerable network overhead, perhaps out of proportion with the dynamism of the network itself, as more timely answers are required.
- d) **LSA Snooping.** Instead of polling each device for information, it might be more sensible to just re-use the overhead already created by OSPF itself, and listen to the LSAs throughout the network (snooping) by interacting with OSPF. A potential way of LSA snooping is to introduce specialized ‘stub routers’ in the OSPF domain that listen to LSAs and create their own LSDs, but neither forward packets nor generate LSAs themselves. LSA snooping scales well in the number of routers/interfaces and doesn’t create any additional overhead. This method also has the advantages of asynchronicity – updates are no more frequent than underlying network changes. Nevertheless, deployment is difficult, requiring the introduction of at least one specialized box per OSPF area in the domain. The “stub routers” incorporate all the complexities of OSPF, and hence their implementation is quite difficult.
- e) **Link State Tracking.** The idea of this approach is to shortcut OSPF by directly tracking interface up/down states through SNMP traps. There is no reliance on OSPF messages. The route monitoring system would then have to emulate OSPF to build an equivalent to the Link State Database, based on which the computation of the routes is done. This database is updated via SNMP traps. Link state tracking is scalable and simply deployable since it requires the use of only one single SNMP manager in the domain. However, it still creates some additional overhead and it isn’t synchronized with OSPF. Namely, the use of SNMP traps to alert the route monitoring system makes it aware of network changes before the LSAs advertising

this change are actually flooded over all the necessary routers. Therefore the system becomes aware of the routes that OSPF is going to compute even before the protocol itself actually converges. In case of route flapping, for example, this solution can prove to be completely out of touch with the routing protocol. Moreover, the aggregation mechanism used in summary LSA computation is non-standard. Hence, the route monitoring system must incorporate detailed understanding of the summarization process, or summary LSAs must be periodically retrieved from a backbone router.

On detailed consideration of the above approaches, we chose to design a system that could work with either of last two approaches, namely LSA snooping as well as link state tracking. In actual deployment, we expect that a combination of these two, for instance, a deployment that snoops on LSAs in certain OSPF areas and using SNMP traps in other areas, would prove beneficial. In the next section, we use to above discussions as a basis to derive the design of an OSPF route monitoring system that would be scalable, asynchronous in its operation, and achieve a good tradeoff between network and computational resources.

### 3.3 System Architecture

Fig 4 depicts the architecture of an OSPF route monitoring system. We first describe the general modules in the architecture, and then go on to their interactions in the three modes of operation: *initialization mode*, *query mode*, and *update mode*.

#### 3.3.1 System Modules

**Route Computation Module** - This is the central module responsible for real-time route calculation. At the API level, the Route Computation Module features the following methods, as explained before:

- `Init()`, that initializes the module
- `query_route()`, that performs the point route query between two interfaces in the OSPF domain
- `monitor_route()` that performs a persistent query on the route query between two interfaces in the OSPF domain

In addition, the route computation module generates `update_route` events to notify the management application of changes to the persistent query. It also processes `update_topology` events generated by the Topology Module and `update_link_state` events generated by the Network State Update Module.

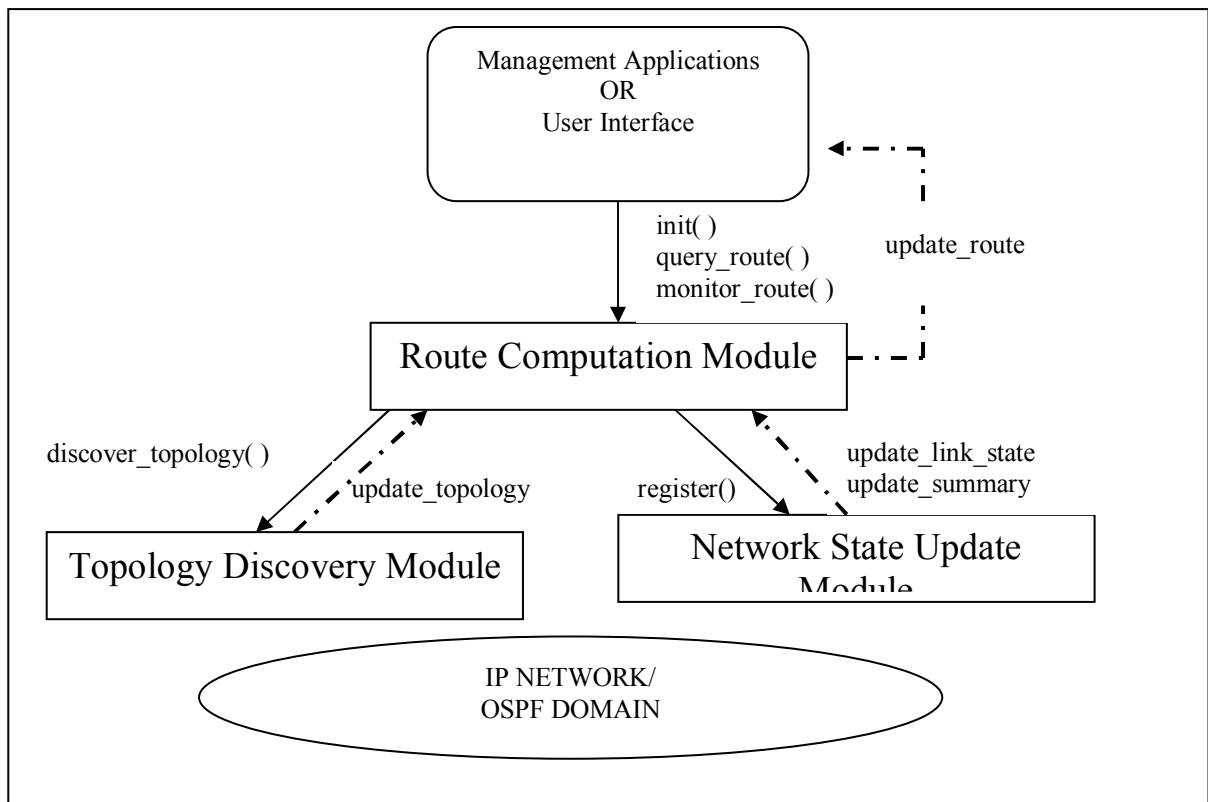


Figure 4 : Modular Architecture

**Network Discovery Module** – This module is responsible for retrieving all the configuration information as well as updates thereto. It supports the following method :

- `discover_topology()`, that returns a list of routers, their interfaces, weights, adjacencies and areas

The Network Discovery Module refreshes the configuration state of the Route Computation Module through `update_topology` events.

**Network State Update Module** - This module monitors the state of the network and maps these changes into `update_link_state` or `update_summary` events. The Route Computation Module must register towards this module using the following method:

- `Register()`, that enables the central module to register for updates

The Network State Update may be implemented as through LSA snooping or Link State Update. In the former case, the snooped LSAs are used to generate the events. In the latter case, interface up/down traps generate `update_link_state` events, while the `update_summary` events are either natively generated by the Network State Update Module itself, or by querying the LSDs of backbone routers.

**Management Application or Graphical User Interface** – These are users of the Route Computation Module and respond to `UpdateRoute` and `Update_link_state()`.

### 3.3.2 The Initialization Mode

In this mode, a management application (the GUI for instance) launches the initialization of the Route Computation Module. The latter launches the discovery of the managed network by the Network Discovery Module. After the discovery is complete, the Network Discovery Module returns the topology to the Route Computation Module. The latter processes this topology and registers with the Network State Update Module for events relating to the discovered areas and routers/interfaces. The system is then initialized and functional, and the Route Computation Module can inform the management application of the topology. At the API level, the flow of method calls is the following :

- The management application calls `init()` in the Route Computation Module
- The Route Computation Module calls `discover_topology()` in the Network Discovery Module
- The Route Computation Module calls `register()` in the Network State Update Module, which returns the current set of link states for the area(s) registered.

### 3.3.3 The Query Mode

When a management application or GUI queries the Route Computation Module by specifying the origin and destination interfaces in the domain the latter responds with a computed set of known routes. If the query is persistent then subsequent events will be generated when routes change.

- The management application calls `query_route()` in the Route Computation Module which immediately returns a list of least cost routes from origin to destination.
- The Route Computation Module generates subsequent `update_route` events in the event of a `monitor_route()` query.

### 3.3.4 The Update Mode

The most important aspect of the Route Monitoring System is its ability to adapt its computations based on network changes. On being notified of changes to topology, summary or link state the Route Computation Module switches to the so-called update mode, re-computes routes, and passes on the updates to the management application(s).

## 4. Implementation of the OSPF Route Monitoring System

In this section, we describe an implementation based on the design described in the previous section. The focus of our discussion is the Route Computation Module, as this raises the most interesting questions at the core of the implementation. We present a brief overview of the system implementation, before we plunge into these questions.

### 4.1 System Implementation Overview

The implementation is written in Java within the context of a larger project *Policy Arena* at AT&T Labs Research. The objective of *Policy Arena* is to enable real-time provisioning of QoS based services over an IP backbone; a project that is integrally dependent on real-time route awareness. The *Policy Arena* project is designed around a standard 4-tier architecture with clean interfaces between events, distributed objects and data (see Fig. 5). The layer of tools interface with web-based GUIs over HTTP or RMI. A layer of distributed objects with a common information model is supported by the Object Gateway. The Data Gateway provides access to both static data stored in directories or databases, as well as dynamic data accessible using SNMP.

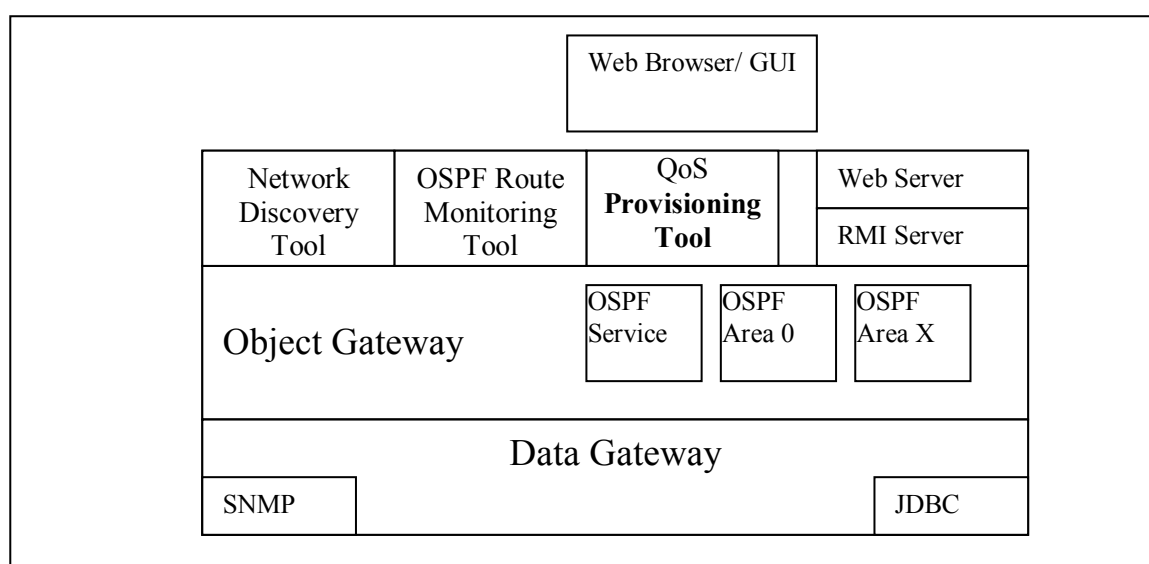


Figure 5: The 4 tier architecture of Policy Arena

The implemented system features a user-friendly GUI that displays the domain topology and the routes on the screen. This way, it is possible for an administrator to visualize in real time what the routes actually taken in the network are. A screen shot is given in Fig 6.

We now briefly map the implementation classes to the modules described before. The **Network Discovery Module** is implemented as a Network Discovery Tool, which allows for topology configuration using the GUI or by interfacing with a commercially available network discovery package such as *HP Openview*. The QoS Provisioning Tool is a **network management application** that uses the route computation service. The **Network State Update Module** is implemented within the data gateway. In the current state of our implementation, we use SNMP traps to directly obtain the up/down state changes of interfaces and use these to drive the Route Computation Module. The Data Gateway provides interfaces whereby the Route Computation Module may register for various interface traps and read SNMP MIBs from network devices. Work on the LSA Snooping approach is ongoing.

On initialization, the OSPF Route Monitoring Tool initializes the Network Discovery Tool. The latter discovers network devices and their OSPF attributes using an external network discovery package and creates various distributed objects corresponding to routers and interfaces in the Object Gateway.

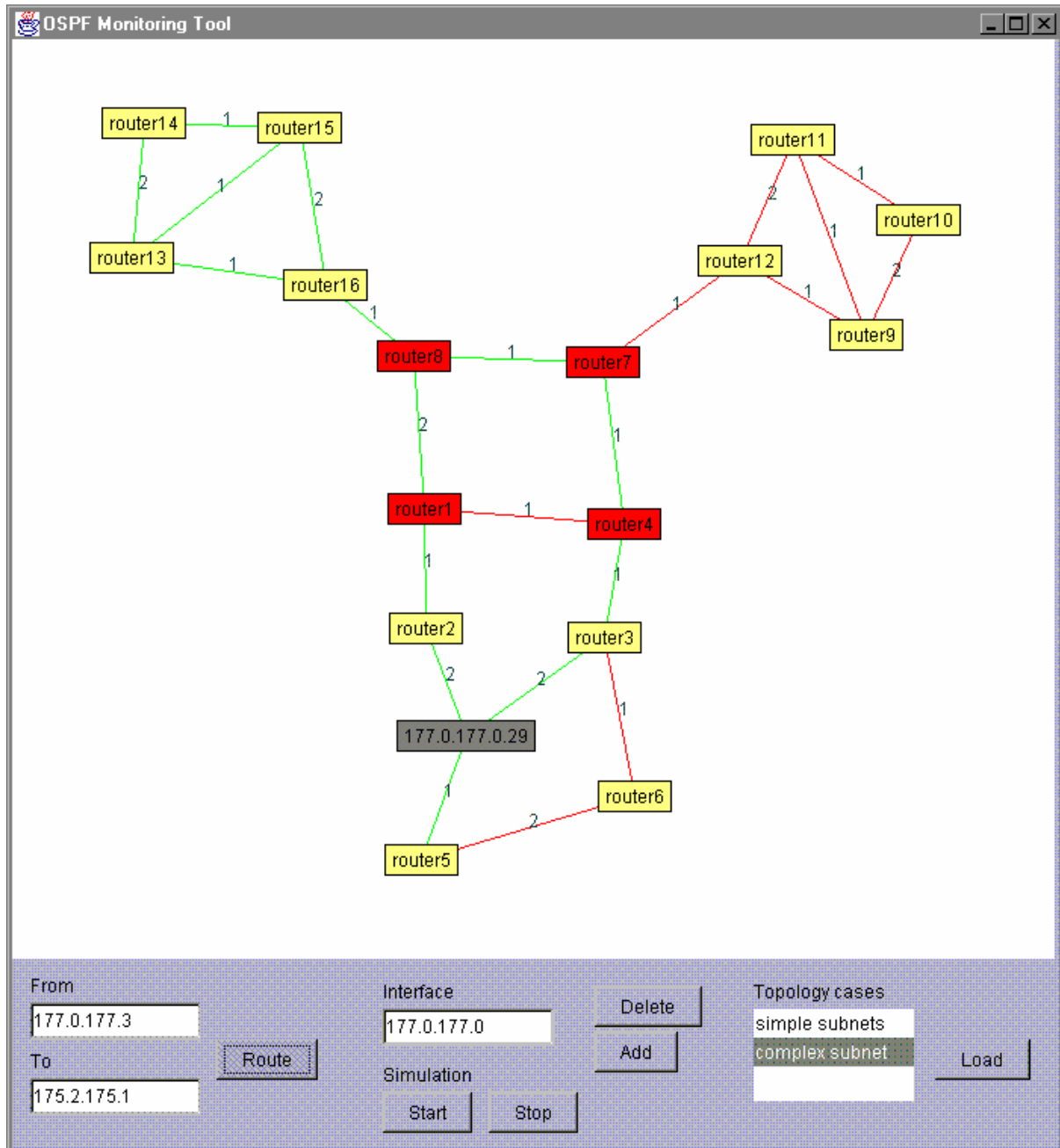


Figure 6: Screen Shot of the Route Monitoring Implementation

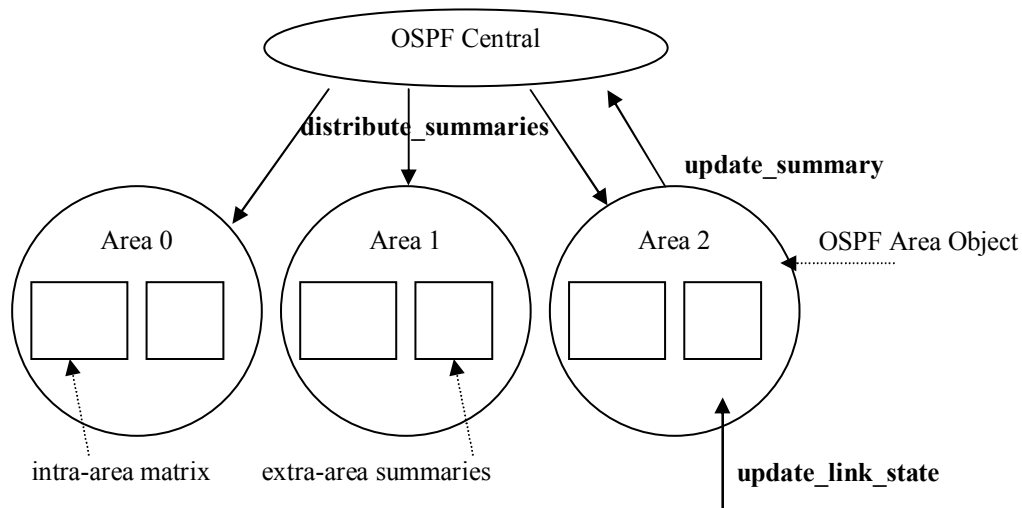
#### 4.2 Implementation of the Route Computation Module

The simplest approach to implementation of the route computation module would involve a single process that computes routes based on state information gathered into a single point in the network. However, such an approach has two drawbacks. A single workstation or process would be unable to keep up with the computational requirements of large networks, and the placement of the management station could end up being geographically distant from most network devices. Therefore it is desirable to implement the route computation module using distributed objects performing different aspects of computation at different management locations. This allows for a versatile tradeoff between distribution of computation and the overhead due to communication between the network devices and various distributed system components.

But then, at what granularity should we design the distributed computation? One possibility is to mimic OSPF and use router objects to form the basis of computation. This approach is seen to be wasteful, as routers within

the same area simply compute shortest path trees based on the same link state database. Using objects that model OSPF areas, we save considerable redundant computation, while simultaneously saving on the overhead of state.

Another implementation issue of interest is the design of summary distribution. Recall that the very design of OSPF area splitting, chiefly the special central role of the backbone, dictates a hub-and-spokes model with a central element, as shown in Fig 7. The alternative is to allow each area to directly distribute its summaries to all other areas. However, this last approach does not avoid a single point of failure. Each area needs information about Area 0 routing in order to use summaries from another areas. As we see no way around this central weakness of OSPF we incorporate the special role using a central element called the OSPF *central object* responsible for the synchronization of summaries across other areas. Note that Area0 is then modeled just like any other area, all its special feature have been abstracted in the central object.



**Figure 7: OSPF Central and Areas**

Then how do the central and area objects co-ordinate and distribute various operational modes of the Route Computation Module?

**The Initialization Mode.** When the system is started at first, initialization is propagated throughout the objects in the following order:

1. The OSPF Central object is constructed by the OSPF tool, and in turn launches the construction of all the area objects.
  2. Each area object constructs an intra-area matrix, representing forwarding tables for internal routes using just the topological attributes of various interfaces in that area, their weights and adjacencies.
  3. When this is done - i.e. when all intra-area matrices areas are constructed the Central object instructs each area object to prepare its summary.
  4. Each area object computes its summary, and hands this over to the Central object, which in turn sorts through and redistributes the summaries. This last phase then involves OSPF Central choosing which summaries are going to be used by which area. Thus, the OSPF Central basically does the job of backbone and the ABRs. After that, the OSPF Central object stores the appropriate summaries in the External Hashtable of each area.
- OSPF Central instantiates the appropriate areas. Each area features a method named `build()` that the central element then calls.
  - Each area features a method named `get_Summaries()` that OSPF Central calls when the areas are built.
  - OSPF Central features a method named `distribute_Summaries()` that it performs when it has received all the summaries from all the areas.

**The Update Mode.** In case of topological change – i.e. if an interface or a router goes up or down – the appropriate OSPF Area is notified of that event as we have seen in the previous section describing the Network

Discovery Tool. The area re-computes its internal routes (its Intra-Area Matrix). If it turns out that the summaries have changed – i.e. that the information that the other areas should have about this area has changed, the latter triggers the recollection of its summaries by OSPF Central and the redistribution of the summaries to all the areas. Therefore all the areas stay synchronized in case of such an event. At the API level, this can be expressed as follows:

- Each area features a method named `update()`, that is triggered when the area is notified of an event.
- OSPF Central features a method named `updateSummaries()` that is called by the updated area if the change needs to be propagated to the other areas.

**The Query Mode.** The system has to answer queries about end-to-end routing throughout the domain. Each area is answerable for routing inside its scope and OSPF Central is able to coordinate the areas in case of inter-area routing. At the API level, this can be expressed as follows:

- An area object features a method named `compute_Route(source,destination)` that returns the part of the route between the specified source and destination that is inside the area.
- The central element features a method also named `compute_Route(source, destination)` that is able to call the `compute_Route` methods in the areas concerned by the routing. In case of inter-area routing it performs the concatenation of the truncated results given by the different areas to construct the end-to-end route.

## 5. Conclusion

We have presented the design and some details of implementation of an OSPF Route monitoring system. This system seeks to track, in real-time, the routing state of a single OSPF domain. While there are multiple methodologies for gathering information from the IP network and figuring the routing state, we chose a design that can flexibly use link state tracking using SNMP and OSPF snooping. Our current implementation relies exclusively on the former method. Our continuing and future work involves implementing OSPF snooping, and extensive performance testing of the OSPF route monitoring system.

## References

- [1] Simple Network Management Protocol (*SNMP*), RFC, <ftp://ftp.isi.edu/in-notes/rfc1157.txt>.
- [2] B. Fortz and M. Thorup. *Internet Traffic Engineering by Optimizing OSPF Weights*, Spring 2000.
- [3] J. Moy. *OSPF Version2*, RFC, <http://www.faqs.org/rfcs/rfc1583.html>.
- [4] D. Joyal. OSPFv3 MIB, Internet Draft – Work In Progress, <http://www.ietf.cnri.reston.va.us/internet-drafts/draft-ietf-ospf-ospfv3-mib-02.txt>.
- [5] J. Moy. OSPF, *Anatomy of an Internet Routing Protocol*, Addison-Wesley, 1998.
- [6] W. Stallings. *SNMP, SNMPv2, SNMPv3 and RMON 1 and 2*, Addison-Wesley, Third Edition, 1999.