Virtual Routing and Tunneling Protocol: 
a simple heterogeneous network emulator 
providing support for QoS and Multicast

Andrea Biasco
Computer Networks exam project
Bologna, Italy

March 30, 2005
## 1 Introduction

One of the most important challenges in the current research on the Internet infrastructure evolution is to provide support for multimedia applications, such as audio/video streaming, phoning and videoconferencing, real-time interaction. This kind of applications require a strong development on two main research topics:

- **Quality of Service (QoS)** is the ability of a network to provide warranties on communication delays, jitter, bandwidth and data loss that can be associated to a single data flow depending on the network traffic changes. The well-known ‘best-effort’ Internet Protocol (IP) behaviour provide no warranties at all. In its broader meaning, QoS can also include security support (such as data flow encryption and endpoint authentication), fault tolerance and accounting capabilities; in this article a narrower definition of QoS will be adopted.

- **Multicast** is the ability of a network to provide group communications in which multiple subcribers can receive the same data flow from a single producer (one-to-many). The current Internet provide support just for unicast (end-to-end) communications. In its broader meaning, multicast can also include communication between multiple end-entities (many-to-many), each of them being producer and consumer at the same tame (bi-directional data flows); again, in this article a narrower definition of Multicast will be adopted.

Both of these topics have been explored in a dual perspective, wich is orthogonal to them.

- Application-level support for QoS or multicast is a way to introduce these features in the Internet without changing the network layer architecture:
  - QoS can be improved by adopting some design issues at the application level (such as receiver-side buffering, data loss recovery algorithms, adaptive coding techniques); even if this approach can’t provide warranties on maximum delays, jitter and data loss thresholds (nor minimum values for bandwidth), it is widely adopted because of its simple implementation over the existing network infrastructure. Moreover, Internet routers and links are more and more fast and can provide a better (even though ‘best-effort’) service.
  - Multicast communications can be implemented by a set of unicast communications managed at the application level. This approach mainly causes an overhead in network traffic comparing to a network-layer implementation.

- Opposing to the application-level approach, the network-layer upgrade is the only way to get over ‘best-effort’ and unicast IP communications, with a strong impact on protocols and routing technologies:
  - QoS can be obtained through advanced routers that can distinguish between different traffic classes, giving a different service priority to each of them; ad-hoc protocols for network resources reservation are used to establish virtual-circuit connections and to negotiate different QoS levels; proactive queue management algorithms can avoid network congestion and data loss.
  - Multicast communication can be established through specific routing algorithms that can quickly produce a spanning tree over the network graph, connecting all participants of a group; protocols to join or leave a group are defined and corresponding techniques to update the tree (named pruning and grafting respectively) are defined too.

This article will focus on the network-layer approach above and it will describe a Java project which is composed by two main packages:
• The first package, Virtual Routing, will be shown in the first section of this article and it is a simple framework for basic and advanced routing technologies emulation in a real Local Area Network (LAN).

• The second package, Tunneling Protocol, will be shown in the second section of this article and it is a simple protocol to improve QoS and to provide Multicast support in an heterogenous network, running on the Virtual Routing emulator.

Finally, results from the Test package (which is a third package for a simple testing of the two packages above) will be shown.

2 Virtual Routing

The Virtual Routing package contains a Java application that emulate an Internet router, referred to as a 'virtual router'. Multiple virtual routers can interoperate by installing the same application on multiple client of a same IP network (such as a LAN), giving each of them knowledge of a 'virtual network' topology mapped over the underlying real LAN. Every virtual router has the IP address of the client machine it is installed on.

A virtual router can be an instance of one of two main classes (both inheriting from the AbstractRouter parent class):

• The BasicRouter class defines a simple Internet router, with a 'best-effort' and unicast design.
• The AdvancedRouter class defines a more complex router architecture, with some support for QoS and Multicast.

This basic/advanced dichotomy is remarkable all over the class taxonomy of the Virtual Routing package, as shown in figure 1.

Therefore, the virtual network obtained from the distributed deployment of the Virtual Routing application is an heterogeneous environment in which routers that have QoS and Multicast support have to deal with routers that have not. This is a very intriguing scenario in which the coexistence between the legacy technology and the new one well represents the slow transition between the old Internet and a new Internet.

In this section the Virtual Routing software architecture will be shown in detail.

2.1 The router abstract model

The AbstractRouter class is an aggregation of some functional components, according to the simple router model below: a set of input ports, a switching fabric that transports each datagram from the input port it came from to the appropriate output port, a set of output ports. Every input/output port is connected via an unidirectional link to another router (referred to as the peer virtual router) or to an end-entity.

In a virtual router that extends the AbstractRouter class, input/output ports are thread pools of InputPort and OutputPort instances respectively; they perform the receiving/sending of the virtual IP datagrams, Datagram class instances deserialized/serialized by the ports in order to receive/send them through UDP packets.

An InputPort thread that receives a Datagram object, simply moves it to the AbstractSwitchingFabric inherited class instance (which is unique to each virtual router), that forward it to the chosen output port according to the RoutingTable instance interrogation. Every OutputPort thread has an AbstractQueueManager inherited class instance, which can manage single or multiple FIFO queues for sending virtual datagrams.

Queues in the output ports are responsible for the most important delay factor; furthermore, variability of the queue length due to changes in network traffic is responsible for the jitter and, if the queue is full, for data loss due to datagram rejection.

The AbstractSwitchingFabric also provide support for broadcast datagrams and for TTL (Time To Live) control.

![Figure 1: UML class diagram showing the basic/advanced dichotomy characterizing the whole Virtual Routing package class taxonomy.](image-url)
The Virtual Routing application does not implement any routing algorithm to update the RoutingTable object, which is manually initialized through the virtual network topology specifications. Indeed, dynamic updating the routing table is not necessary for the Tunneling Protocol application test to work properly; however, the Virtual Routing application can be extended in order to support some link state or distance vector routing algorithm. The metric used for choosing the best route to a destination IP address is the hop number, as frequently is set up by network administrators in LANs.

Furthermore, the Virtual Routing application cannot emulate a high complexity network with hierarchical routing distinguishing between intra-AS (Autonomous System) and inter-AS datagram forwarding. Again, a LAN model is assumed, performing routing over a single domain (intra-AS routing).

### 2.2 The basic router

The BasicRouter class extends the AbstractRouter class without adding it further functionalities. It instantiates a single BasicSwitchingFabric and a BasicQueueManager for each OutputPort thread. Datagram management is therefore the same as described above: interactions in a basic router are shown in figure 2.

A single FIFO queue is used by the BasicQueueManager of an OutputPort, implementing a 'best-effort' policy:

- Every datagram that is forwarded by the switching fabric to an output port is queued on the same output port FIFO queue, without managing any priority between datagrams belonging to different traffic classes.

- If the queue is full, datagrams are rejected in a totally reactive way.

![Figure 2: UML sequence diagram showing the interaction for datagram forwarding in a basic router.](image)

### 2.3 The advanced router

The AdvancedRouter class extends the AbstractRouter adding it advanced functionalities by instanting a single AdvancedSwitchingFabric and an AdvancedQueueManager for each OutputPort thread.

The AdvancedSwitchingFabric can move router addressed datagrams to an appropriate AbstractCustomProtocol inherited class instance, such as the Tunneling Protocol class which is the main class for the Tunneling Protocol package. Every virtual datagram contains a String that specifies the custom protocol the datagram is to be moved to; the switching fabric verifies if the router has an implementation of the custom protocol application requested then moves it to the associated application or reject it. Custom protocols may be used to implement QoS and Multicast strategies.

The AdvancedQueueManager for each OutputPort thread offers support for QoS policies by implementing the Weighted Fair Queuing (WFQ) algorithm which introduces multiple queues to manage different traffic classes and their priorities, and the Random Early Detection (RED) algorithm which is a proactive management of each single queue.

The current Virtual Routing application does not implement any bandwidth control algorithm (such as the leaky-bucket or token-bucket techniques) on the output ports of advanced routers. Again, bandwidth control is not necessary for the Tunneling Protocol application test to work properly; however, the Virtual Routing application can be extended in order to support one of these algorithms.

#### 2.3.1 The WFQ implementation

According to the WFQ algorithm as shown in [3], multiple FIFO queues are associated to a single AdvancedQueueManager instance.

- When the OutputPort object asks the AdvancedQueueManager object for a new datagram to send, the queue manager does a Round-Robin cycle on the FIFO queues, increasing a different counter for the first out datagram of each queue (and increasing it by the value corresponding to the weight of that queue); every Round-Robin cycle is referred to as a 'virtual service cycle'.

- When one of these counter reaches the value of the corresponding datagram bit length, the whole datagram is taken from the queue and is returned to the OutputPort object.
In an advanced router, the appropriate weighted queue of the selected output port is chosen, according to the priority that every datagram can request. The 0-priority value identifies the default queue which is assigned to normal 'best-effort' traffic.

2.3.2 The RED implementation

According to the RED algorithm as shown in [3], a minimum and a maximum length thresholds are set for the FIFO queues.

- When the AdvancedSwitchingFabric object asks an AdvancedQueueManager instance to put a datagram into the weighted queue (associated to that datagram priority) of its output port, a check on the queue length is performed:
  - If queue length is under (minimum) threshold, the datagram is put into the queue.
  - If queue length is over (maximum) threshold, the datagram is rejected.
  - If queue length is between minimum and maximum thresholds, the datagram is reject with a probability which is proportional to the queue length itself.

Thanks to the RED proactive behaviour, when observing an increasing percentage of datagram loss, a data flow receiver can asks its source to reduce its bitrate in order to avoid congestion before it’s too late.

3 Tunneling Protocol

The Tunneling Protocol package adds Multicast (one-to-many) communication support to an AdvancedRouter instance by implementing the AbstractCustomProtocol class with the TunnelingProtocol class. In addition, it defines a simple QoS policy that uses the advanced queue management shown in the last section of this article.

3.1 Protocol description

The main idea behind the protocol is that the set of all advanced router can be a virtual backbone in the network emulated by the Virtual Routing application distributed deployment. This means that an end-entity wanting to subscribe to a particular multicast group has to contact the nearest advanced router and to signal its interest to it.

The advanced router will provide to that end-entity (and to any other entity that is interested in receiving it) all datagrams of the multicast data flow via an equal number of unicast end-to-end communications. Interested entities can also be other advanced neighbor routers to whom other end-entities have subscribed for the same multicast data flow; in any case, datagrams are embedded in other unicast datagrams serving as a tunnel between the two entities\(^1\); in that way, unicast communications can be made via a certain number of basic routers that completely ignore the multicast data flow that datagrams belong to.

When an advanced router doesn’t know the multicast data flow it is requested to forward, it communicates with the nearest advanced routers in the network (referred to as the ‘advanced neighborhood’\(^1\)) to find that requested data flow and, if necessary, this procedure is iteratively repeated by the last contacted routers. In this way, multicast spanning tree are created from subscribers to the publisher as shown in figure 3, and the publisher simply forward its multicast data flow to its nearest advanced router via another unicast communication.

![Figure 3: Example of tunneling on a very simple multicast spanning tree.](image)

The tunneling procedure is not transparent to the publisher/subscribers end-entities of a multicast data flow, because they have to communicate in tunneling mode with their nearest advanced routers.

3.1.1 The multicast datagram forwarding

Every advanced router can be a node in different spanning trees so it has to maintain some state information about every multicast data flow, each one corresponding to a different tree. This is why a TunnelingProtocol instance has a collection of DataFlow instance objects, each one maintaining:

- The multicast D-class IP address that uniquely identifies a multicast data flow.

\(^1\)The tunneling technique described here is similar to those used by IPv6 enabled routers in order to communicate via normal IPv4 routers, or to provide security for a VPN through IPSec gateways. Also the MBone multicast project uses tunneling to forward multicast datagram between different ASs supporting multicast.
The IP address of the multicast data flow source, which can be the parent advanced router in the tree or the publisher end-entity.

The list of all subscribers’ IP addresses the multicast data flow has to be forwarded to; subscribers can be end-entities or other advanced routers; a soft state is associated with each subscriber, as it will be shown later in this article.

In addition to that, every AdvancedRouter instance maintains a list of every router in the advanced neighborhood, that is every advanced neighbor router, directly connected to that router in the virtual backbone; two advanced routers are directly connected in the backbone (they are said neighbors) if no other advanced router is on the path between them (even though basic routers can be on that path).

When the advanced router receives a tunnel datagram that an advanced neighbor router (or the publisher end-entity) has addressed to him, the switching fabric moves it to the TunnelingProtocol instance that de-embed the inner multicast datagram and uses its D-class address to index the DataFlow objects list. From the selected DataFlow object, subscribers’ IP addresses are taken and the multicast datagram is embedded into an equal number of tunnel datagrams to be forwarded to those addresses; the RoutingTable object is used from the switching fabric in order to send tunnel datagrams to the right output ports.

As previously shown, every datagram packet may specify a priority value that WFQ scheduling in advanced router can use as a weight; basic routers in the path, however, doesn’t care about datagram priorities. This is why the TunnelingProtocol instance calculates the priority of the outer tunnel datagram as the priority of the inner multicast datagram multiplied by the hops number of the best route selected from the routing table, which is the number of basic router in the path to the subscriber.

This implements a simple QoS policy in an heterogeneous network where advanced routers, between datagrams with the same priority, give more service time to those that will pass through a greater number of basic routers that will not consider that priority. The Tunneling Protocol QoS policy is discussed later in this article.

3.1.2 The publish/subscribe protocol

In addition to the forwarding of tunnel datagrams, the TunnelingProtocol instance has to manage control messages in order to build the spanning tree and to maintain it (pruning and grafting its branches). This task is achieved by a MessageManger instance, using the state associated with each subscriber and two message types, instances of the SubscribeMessage and the PublishMessage classes, both inheriting from the AbstractMessage classes. Messages are embedded into outer unicast datagram (such as for multicast datagram in tunneling mode).

The protocol is subscriber oriented, because the first path of a tree is build when the first subscriber wants to receive a particular multicast data flow; other branches are grafted when other subscribers want to receive the same flow. The subscribers/publisher end-entities are supposed to act as advanced routers in the protocol: in addition to tunneling capabilities for multicast datagram receiving/sending (as seen before), subscribers have to maintain knowledge of their advanced neighbor routers, publisher has to be inserted into the advanced neighborhood knowledge of the nearests advanced routers.

When an end-entity wants to subscribe for a multicast data flow, it sends a subscribe message (embedded into an unicast datagram) to its advanced neighborood; the message contains the D-class IP address of the multicast data flow that the subscriber wants to receive.

When an advanced router receives the subscribe message, it verify if it already has a DataFlow object corresponding to the multicast D-class IP address specified. If that, it adds the new subscriber’s IP address to the DataFlow instance receivers list, associating it a 0 state which means: ‘to confirm’. Then it sends a publish message to the subscriber, containing the D-class IP address of the multicast data flow; this message indicates that the router is able to forward multicast datagrams of that flow.

When the chosen advanced router receives that second subscribe message, it update to 1 its state which means: ‘confirmed’.

As seen before, if an advanced router does not have a DataFlow object for the multicast D-class IP address that is specified in a received subscribe message query, it has to search for that flow, behaving exactly the same as an end-entity wanting to subscribe for it. The protocol is iterated until an advanced router that already manages the data flow replies with a publish messages; a chain
of publish reply propagate thorough the network back to the first router that searched for the flow and then it can finally sends a publish message to the end-entity subscriber.

The state associated with each subscriber of a multicast data flow is a soft-state: a timeout is set every time the state is updated; when its deadline comes, the state value is decreased by 1 (from 1 to 0 and from 0 to -1; a state diagram for the publish/subscribe protocol is shown in figure 4):

- When in 1-state, the subscriber is (re-) confirmed, and tunneling forwarding of multicast datagrams is activated for him.

- When in 0-state, the subscriber has to be (re-) confirmed, but forwarding is still activated for him; a publish message is sent to him.

- When in -1-state, the subscriber is considered no more interested in receiving that multicast flow: datagram forwarding is deactivated for him and its place in the subscribers’ list can be used for another entity to subscribe.

![State diagram for the publish/subscribe protocol](image)

3.2 A simple test

The Test package contains the Installer application class that has to be used in order to perform a very simple test of the Virtual Routing and the Tunneling Protocol packages. This test can be deployed over a LAN network with three clients at least, each acting as a router running the Virtual Routing and (eventually) the Tunneling Protocol applications locally. Two clients act as advanced routers (R1 and R3) and one client act as a basic router in the middle (R2), as shown in figure 5.

![The virtual network topology for the simple Test package](image)

The Installer class sets the topology of the virtual network; depending on the IP address of the client over which it is running, it gives each router the virtual network knowledge it must have in order to perform the test: virtual links, routing table and (eventually) advanced neighborhood. Moreover, the Installer class emulates the publisher (P) and the subscribers (S1, S2) end-entities of a multicast data flow M.

The R1 advanced router is initialized as it already had a DataFlow object for M with the S1 subscriber in a 1-state; a simple multicast datagram forwarding is tested from P to S1 via R1 (P and S1 being emulated by the Installer application running on R1’s client).

The S2 subscriber (emulated by the Installer application running on R3’s client) sends a subscribe message to R3, requesting the forwarding of the multicast data flow M, which R3 doesn’t have knowledge of. This unleash the publish/subscribe protocol:

- R3 creates a new DataFlow object for M, inserting S2 in it with a 0-state (‘to confirm’), and it sends a subscribe message for M to R2 (in tunneling via R2), which is its only advanced neighbor (if R3 have had other advanced neighbors, the subscribe message would be sent to the whole advanced neighborhood).

- R1 receives the subscribe message from R1 (via R2); yet having a DataFlow object for M, R1 inserts R3 in it with a 0-state (‘to confirm’), and it replies to R3 (via R2) with a publish message for M.

- R3 receives the publish message from R1 as the first publish message for M (because R1 is inci-
dentally its only advanced neighbor), so it elects R1 as its publisher for M, sending a subscribe confirmation message to R1 (via R2) and propagating the publish message to S2.

- R1 receives the subscribe confirmation message from R3 (via R2) and it updates R3 to the 1-state (‘confirmed’) in the M DataFlow object.
- The M DataFlow timeout in R1 reaches its deadline: both S1 and R3 1-state are decreased to 0-state and a publish confirmation request message for M is sent both to S1 and to R3 (via R2).
- R3 receives the publish message from R1; having a DataFlow object for M with at least one subscriber in a state that is greater or equal to a 0-state, R3 replies to R1 with a subscribe confirmation message sent via R2.
- R1 receives the subscribe confirmation message from R3 (via R2) and it re-confirm its 1-state in the M DataFlow object.
- The M DataFlow timeout in R3 reaches its deadline: S2 has still a 0-state, because it has not replied to R3’s publish message (this can simulate three different scenarios: S2 was down, it was no more interested in M, or it replied to another publish message for M that another hypothetical advanced router had sent to him before R3); the decreasing of the 0-state for S2 to a -1-state causes the M DataFlow object deletion, being S2 its only subscriber.
- The M DataFlow timeout in R1 reaches its deadline: S1 has still a 0-state, because it has not replied to R1’s publish confirmation request message, and the 0-state for S1 is decreased to a -1-state, which means that it can be replaced in the DataFlow by another subscriber; S2 has a 1-state, because it replied to the R3’s publish message; its state is decreased to a 0-state and another publish confirmation request message for M is sent to R3 (via R2).
- The M DataFlow timeout in R3 reaches its deadline: R3 has still a 0-state, because it has not replied to R1’s publish confirmation request message (having its M DataFlow deleted); the decreasing of the 0-state for R3 to a -1-state causes the M DataFlow object deletion, being R3 the only subscriber that had remained.

Similar other simple tests have been performed in a LAN over a limited number of hosts; more accurate tests would require a further project development, including a better support for virtual network topology specification through the deployment of the Installer application.

### 3.3 QoS and Multicast remarks

The QoS management in the Tunneling Protocol application running on the Virtual Router emulation is very simple and naive. There is no negotiation of resources on the tree\(^2\), but just a differentiation between different datagram priorities is made basing upon a value set in the Datagram object by the publisher of each data flow. Only advanced routers support priority management, but basic routers do not; in such an heterogeneous scenario, QoS levels cannot be guaranteed, but advanced routers can supply to basic routers deficiencies.

This priority value, as seen before, is multiplied by a factor which is the number of basic routers in the path to the next advanced router of the tree. This local policy implements a simple ‘compensation principle’: if a datagram has a certain priority over normal ‘best-effort’ traffic, every basic router that it will pass through, ignoring this priority, will serve it the same manner as for normal ‘best-effort’ traffic, causing an extra-delay to the datagram; the greater is the priority value, the greater the extra-delay is. Predicting this, an advanced router multiply the priority of the datagram by the number of extra-delays that the datagram will accumulate during its path to the next advanced router.

This compensation principle works only because the delay metric is addictive: from the source to the receiver (from the publisher to the subscriber) a datagram accumulates a total delay which is the sum of delays that each router determines in the path. It was previously noticed that Virtual Routing application doesn’t consider the bandwidth metric; even if it was, the Tunneling Protocol wouldn’t be able to apply a compensation principle on bandwidth, because of its nature, which is not addictive. The bandwidth value of a path is that of the worst link, which is a bottleneck to the whole path; this is why bandwidth is referred to as an hyperbolic (not addictive) metric.

The multicast spanning tree build up by the Tunneling Protocol application is obviously a suboptimal tree\(^3\).

\(^2\)As with an INTSERV approach with the RSVP protocol.

\(^3\)Finding an optimal (minimum) spanning tree for just a node subset in a graph is an NP-hard problem known as the finding of the Steiner tree; this is not the same problem of finding the optimal spanning tree on the whole graph, which is resolvable with simpler algorithms (such as the Prim algorithm). This is why all standard protocols for Multicast communications over the
A trade-off between two optimality criteria is considered; minimizing the delay for each subscriber and minimizing the total network traffic due to the multicast data flow:

- When the first subscriber to a multicast data flow find its path to the publisher, this path is chosen because it involves the maximum number of advanced routers with the minimum delay on the path: every advanced router choose the next advanced router on the way to the source by sending a subscribe query on the advanced neighborhood and confirming the first neighbor router that reply with its publish message, which is the next router on the path with the minimum RTT (Round Trip Time) between the publisher and subscriber end entities.

- When a new end-entity wants to subscribe to a multicast flow which already has one or more other subscribers, the minimum delay path to the publisher end-entity is diverted, ‘attracted’ to the existing tree which is already forwarding that flow, then grafted to it: when at least one advanced neighbor of an advanced router is already forwarding that flow to some subscribers, it will be obviously the first to reply with a publish message, so it will be the one that will be confirmed, even if another advanced neighbor may had have the minimum RTT path to the source.

Until it is not attracted to the existing tree, the path is chosen in order to minimize the delay; when grafted to the existing tree, the path minimizes the network traffic.

Every advanced router in the tree has a soft-state in order to maintain it; as seen before, a timeout-based mechanism oblige every subscriber to confirm its interest in receiving the flow to its publisher. For the same reason above, every time a deadline is reached, the same publisher will be confirmed from a subscriber. This means that the tree is not dinamically changed if some (basic) routers in it are causing congestion for some subscribers. This design choice is perhaps critical for maintaining a certain QoS level for a multicast data flow that is active for a long time, but it is necessary in order to minimize the computational and network traffic overhead to maintain its multicast tree.

4 Internet build suboptimal trees instead of Steiner trees.

5 This is somewhat similar to the Core-Based Trees technique, used by several standard protocols for Multicast as shown in [2], [3], [4].

4 Conclusion

The Java project presented in this article has the purpose of testing the proposed Tunneling Protocol into the heterogeneous scenario of the Virtual Routing emulator.

The QoS and Multicast support is quite simple, but running the test demonstrates that even that naive approach to these topics produces a certain overhead in advanced router datagram management compared to much simpler basic routers datagram forwarding.

Even though a Java application-level emulation of a router is not comparable to an optimized implementation on ad-hoc network hardware, the test confirmed that a trade-off exists between routers performance and sophistication, limiting the complexity that can be injected into the network layer.

References