IRV–Tool

Christian Sternagel

2004-01-12

Abstract

The IRV–Tool (Internet Routing Visualization) is an tool designed to give students easy access to the fundamentals of routing. Two routing protocols and some of their features are implemented in the simulator. In the following sections, the usage of the IRV–Tool and its design are explained.

Contents

1 Routing In The Internet

Routing is the technique by which data is moved across an (inter–)network from a source to a destination. Along the way, at least one intermediate node is situated. Routing occurs at Layer 3 of the OSI reference model (the network layer). Thus routing is the structure that glues together the internet.

2 RIP - Distance Vector Routing

The most widely used 'interior gateway protocol' in the internet is probably RIP – which stands for Routing Information Protocol. It is a very simple protocol of the 'distance vector' family (DV). These protocols are based on a shortest path computation algorithm described by R. E. Bellman (see also [1]). Hence they are sometimes referred to as 'Bellman-Ford' protocols.

2.1 How It Works

Assume a simple network is initialized ('cold start'). After startup every node (e.g. a router in the internet) has only 'local knowledge', i.e. they remember their own address and the links (connections) to which they are attached. This information is stored in a so called routing table (consisting of tuples [to, via connection, cost]). At regular intervals every node sends his local data (the entries of the routing table, called 'distance vector' [DV]) across all of his connections that way reaching all his direct neighbors ('adjacent nodes'). Crossing a connection has typically a cost (or metric) of 1. Therefor reaching a node, that is two connections away has a cost of 2 etc. The above mentioned shortest path computing algorithm considers these costs for calculating a shortest path to a given destination. So after the whole procedure converges every node 'knows' on which of his connections to send a packet for a certain destination to get the packet their with a minimum of cost.

In a static network (where the topology never changes) the above procedure works correctly. But a realistic network is not static: nodes are removed, new ones are added or a link breaks for indefinite time. Hence a routing protocol has to respond to topology changes. RIP solves this problem by regularly sending updates. But sometimes these updates are not enough.

2.1.1 The Bouncing Effect

If there are connections of different costs in a network it is possible, that after a link break, erroneous DVs are spread across it (e.g. if an update is sent before a node notices a link break) thus leading to an effect known as 'bouncing effect', where node A sends all packets addressed to C over connection 1 (to node B) and B sends all packets addressed to C over connection 1 (to node A). The routing tables now include a loop and packets will bounce between A and B forth and back until their 'time to live' expires. This bouncing effect will last until the network converges on a new, coherent version of the routing table. Sometimes this can last very long.

2.1.2 Counting To Infinity

There are situations, where the cost of a connection at each exchange will just increase 2 units. This process is called 'counting to infinity' and can only be stopped by a convention on the representation of infinity as some very large distance: i.e., larger than the length of the worst possible path in the network. When the distance has reached this value, the entry in the table is considered infinitely remote and thus unreachable.

2.2 Enhancements

There are two enhancements to improve RIP. The bouncing effect and the long time taken for counting to infinity are very undesirable features of the distance vector protocols. Therefor 'split horizon' and 'triggered updates' where implemented in RIP.

Split Horizon: It is based on a very simple precaution: if node A is routing packets addressed to C through node B, it makes no sense for B to try to reach C through A. Thus it makes no sense for A to announce to B that C is only a short distance from A. The change is as follows: Instead of broadcasting the whole routing table of a node to all of its neighbors, different versions are send to each of them. There are two versions of 'split horizon' namely 'split horizon' and 'split horizon with poisonous reverse'. Using 'split horizon' a node sends only those entries of its routing table to a neighbor, that were not received from it (which means, that this neighbor is 'closer' to the destination mentioned in the entry than the node itself). Using 'split horizon with poisonous reverse' with every update a DV consisting of the whole routing table is sent to all neighbors, but if an entry was received from the neighbor which it is sent to now, the cost of this tuple is set to INF ('infinity') thus breaking loops more swifter by being more aggressive.

Triggered Updates: Without triggered updates, the DVs are sent to all neighbors regularly after a given time slice (for RIP it is 30 seconds; see [2]), therefore it takes sometimes very long to solve loops (by counting to infinity). The idea of 'triggered updates' is to send DVs to all neighbors every time, an entry of the local routing table has changed, thus responding faster to topology changes.

3 OSPF - Link State Routing

RIP is both limited and simple. OSPF (open shortest path first), on the other hand, is both very powerful and somewhat complex. OSPF is a member of the 'link state' (LS) family. Instead of exchanging distances to destinations, the nodes will all maintain a 'map' of the whole network that will be updated quickly after any change in the topology. These maps (the link state database) can be used to compute more accurate routes than can be computed with the distance vector protocols.

3.1 How It Works

Link state protocols require each router to maintain at least a partial map of the network. When a network link (connection) changes state (up to down or vice versa), a notification, called a link state advertisement (LSA) is 'flooded' throughout the network. All the routers note the change and recompute their routes accordingly. This method is more reliable, easier to debug and less bandwidth–intensive than DV. It is also more complex and more compute– and memory–intensive.

All nodes have a copy of the network map, which is regularly updated. The map is represented by a database and updates are 'flooded' to the network nodes using the flooding protocol. When the network is initialized, it is necessary to 'bring up adjacencies' (which is handled by the hello protocol) and afterwards exchange topology data in a way, that each node has a map of the whole topology (which is handled by the exchange protocol).

If a packet arrives at the router, the local map along with a 'shortest path first' algorithm is used to determine the next hop (the connection to which the packet should be forwarded) for that packet.

3.1.1 The Hello Protocol

Assume a network that is separated into two smaller networks (A and B) if a certain link fails. With OSPF each of these two networks will evolve their own map of the topology. As long, as the two networks stay separated that is not important; every network computes the correct shortest paths for 'their' nodes. But what happens if A and B reconnect (at this point B has no knowledge about any changes in A while A and B where separated and vice versa). Merely distributing the new link information would not be sufficient. Establishing connectivity between two nodes requires more than just sending one database record (as it happens in the flooding protocol). It must be guaranteed, that A and B end up with aligned Databases. This process is called 'bringing up adjacencies' in OSPF. The hello protocol is used for two purposes:

- to check that connections are operational, and

- to find new adjacent routers (neighbors).

After that a synchronization of the link state databases of a newly found neighbor and the router that found it is necessary. This task is adopted by the exchange protocol.

3.1.2 The Exchange Protocol

As mentioned above the exchange protocol responsible for the synchronization of the link state databases of two routers, after they have established two-way connectivity on a point-to-point link via the hello protocol. The initial synchronization is performed through the 'exchange' protocol; the 'flooding' protocol will then be used to maintain the two databases in synchronization.

3.1.3 The Flooding Protocol

When a connection changes state, the router responsible for that connection (sometimes 2 routers) will issue a new version of the connection state. This is achieved by using the flooding protocol. Every router receiving a flooding packet computes the included information and (if the packet is valid) forwards it to all of his connections, except the one from which the packet was received. That way topology changes are propagated very swiftly across the whole network.

3.2 Features

There is one feature (also supported by the IRV–Tool) that is presented in [3]:

Equal-Cost Multipath: Sometimes the discussion about shortest paths is simplified by considering only a single route to any destination (e.g. RIP). In reality, if multiple equal-cost routes to a destination exist, they should be used to improve performance and reduce congestion. With equal-cost multipath a router potentially has several available next hops towards any given destination.

4 The Class Structure Of The IRV–Tool

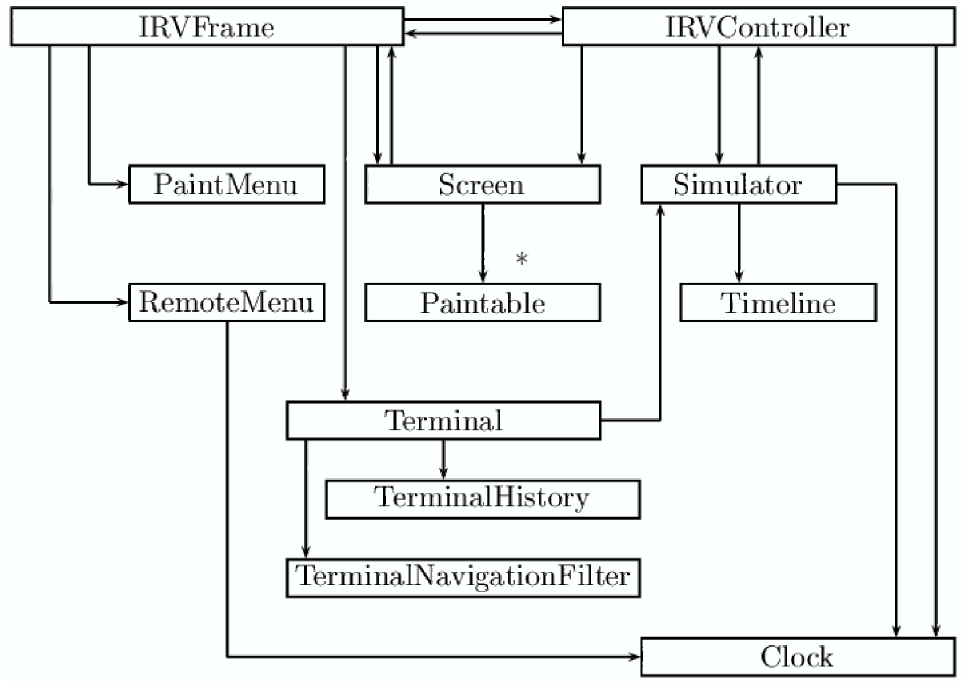


Figure 1: The Class Structure Of The IRV–Tool

Address is used to generate continuous addresses for nodes and connections.

Clock is a graphical component that is used to display an integer (interpreted as seconds) in either of the following ways (the first is default): the number of seconds with maximal seven digits or the time as 'hh:mm:ss'. The mode is switched by clicking on the display. This class is used by the Simulator, the RemoteMenu and the IRVController.

Computer is implemented by Host and Node and by itself implements the Paintable interface (see figure 4). It represents all attributes that are shared by routers and hosts.

Connection implements also Paintable. Together with Router and Host these are the graphical represented components of a network topology. As you can see in figure 4, a Connection always has two Computers (nodes) at its end-points.

ConnectionEditPane is an interactive dialog that can be invoked by either right-clicking on a connection or left-clicking on it, after the Edit–

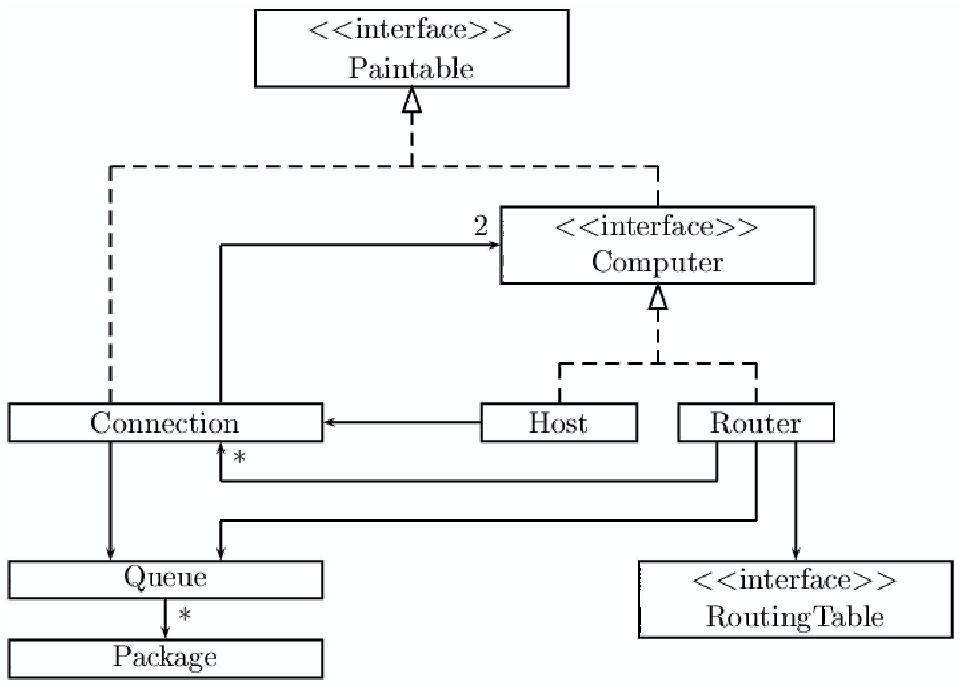


Figure 2: The Paintable Interface

Tool (see section A.3.2) has been chosen. All possible modifications of a Connection are achieved with this dialog (or optional with the Terminal).

Consts holds all constants that are used by the IRV–Tool. Those that concern a certain routing protocol can be modified via the menu (see section A.3.2).

DistanceVectorTable is the implementation of RIP used in the IRV–Tool. It implements the RoutingTable interface.

EditPane is the parent class of all edit panels (ConnectionEditPane, HostEditPane, RouterEditPane).

Host represents a 'normal' PC or simply a computer that is not involved in routing respectively. It implements the Computer interface.

HostEditPane is invoked like the ConnectionEditPane and can be used to change properties of a certain Host.

InfoPane serves as a Window, where text (in the IRV–Tool the GNU license) can be shown.

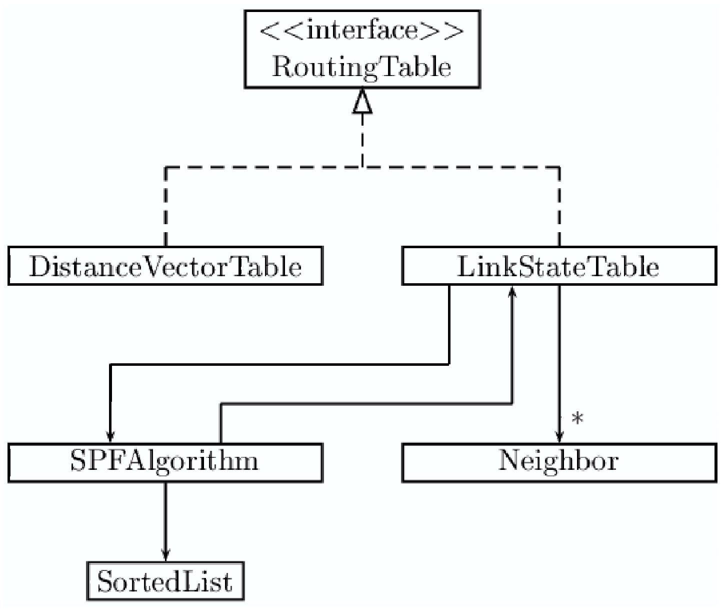


Figure 3: The Routing Table Interface

IPPackage represents all packets that are not involved in routing (e.g. ping, traceroute and all messages). It is a child class of Package.

IRVController is the central post of control, where all the callback functions for event listeners are implemented. As shown in figure 4 the IRVController has access to the Simulator and vice versa.

IRVDialog is used to wrap all edit panels.

IRVFilter is a FileFilter, that only accepts *.IRV files.

IRVFrame acts as the main window for the IRV–Tool and aligns all other graphical components (see also section A.3.2).

JPGFilter matches the IRVFilter except that it only accepts *.JPG files.

LinkStateTable is the implementation for the OSPF protocol. It uses (depending on its configuration) a certain SPFAlgorithm (see figure 4) and implements the RoutingTable interface.

Neighbor represents the nodes that are directly connected to a OSPF router. In the implementation of OSPF it is used for the process called 'bringing up adjacencies'.

OSPFExchangePackage is used to exchange data of two link state tables that should be synchronized.

OSPFFloodingPackage is used to 'flood' new information (like after a state change of a connection) as quickly as possible across the network.

OSPFHelloPackage is used to detect neighbors and initializes an exchange (by OSPFExchangePackages) if a new one was found.

OSPFVariableEditPane is invoked from the menu and used to set several variables (and constants) used by OSPF.

Package is the parent class for all packets (IPPackage, OSPFExchangePackage, OSPFFloodingPackage, OSPFHelloPackage and RIPPackage). It holds information about source, destination, type, TTL and content of the packet.

Paintable is the interface that must be implemented by each object that should be plotted on the Screen (see figure 4).

PaintMenu is the graphical representation of the tool bar situated at the left hand side of the IRV–Tool.

Path represents a path, consisting of several nodes and an assigned cost.

Queue is a simple FIFO (first in first out) queue that is used by Connections and Routers. Every router has a queue for incoming packets and every connection one for packets that are to be transmitted by it.

RemoteMenu is the menu for the Simulator (see section A.3.2).

RIPPackage is used to send a DV from one router to another (adjacent one).

RIPVariableEditPane conforms with OSPFVariableEditPane except that it is used to modify RIP specific variables (and constants).

Router represents a node in the network that is a router and thus maintains a routing table.

RouterEditPaneOSPF see HostEditPane.

RouterEditPaneRIP see HostEditPane.

RoutingTable is the interface for all kinds of routing tables (e.g. link state databases ...).

Screen is the component on which the topology can be drawn and the simulations can be studied.

Simulator is the core of every simulation. All required calculations are done within the Simulator.

SortedList represents a list of paths that are sorted according to their costs.

SPFAlgorithm represents the algorithm used by OSPF to calculate a shortest path.

Splash . The splash–screen.

Terminal is a text based command interpreter that can optionally be used to configure a given topology (see section A.3.7).

TerminalHistory is used to give easy access to commands that are often used.

TerminalNavigationFilter limits the positions where the curser can be set within the command line.

Timeline is a simple counter that holds the elapsed time of a simulation.

5 RIP In The IRV–Tool

Every time a new DV arrives at a router the following procedure is applied (pseudo–code):

```
FOR all entries of the DV DO
  distance is distance in entry plus the cost of the connection
  IF there is already an entry for the given destination THEN
    IF the distance is smaller than in the current entry or the
    new one was received from the router THEN
      add the entry
      IF distance has changed THEN
        mark change
      END IF
    END IF
  ELSE
    add the entry and mark change
  END IF
OD
IF a change was marked and 'triggered updates' is
active THEN
  broadcast an update
```

6 OSPF In The IRV–Tool

There are some variables used by OSPF in the IRV–Tool:

HELLO_INTERVAL: The number of seconds between two broadcasts of hello packets to all neighbors.

TIME: The age of a certain link state entry in seconds.

MAXAGE: As soon as TIME equals MAXAGE a link state entry expires and is removed.

Here is the procedure that is applied in the Simulator as pseudo–code:

```
FOR every second DO
  FOR all nodes that are in the neighbor list DO
    IF node did not answer for more than HELLO_INTERVAL
    seconds THEN
      remove the node from the neighbor list
    END IF
  OD
  FOR all elements of the link state table DO
    increment TIME
    IF TIME >= MAXAGE THEN
      remove entry from link state table
      and flood removal to all neighbors
      IF the entry referred to a neighbor THEN
        remove this neighbor from the neighbor list
      END IF
    END IF
  OD
  FOR all connections of this router DO
    IF the connection is down THEN
      remove the opposite node from the neighbor list
  OD
  IF HELLO_INTERVAL seconds have passed THEN
    broadcast hello packets over all active connections
OD
```

Every time a hello packet arrives at a router:

```
IF the sender of the packet is not in the neighbor list THEN
  add new neighbor to the neighbor list and add the new route
  to the link state database.
  send a exchange master packet to the sender
ELSE
```

```
  set the time this neighbor did not answer to zero
END IF
```

Every time an exchange packet arrives at a router:

```
IF the router was waiting and an initial master packet arrives THEN
  IF the address of the sender is less than the address of the
  router THEN
    solve a possible collision by not answering
  END IF
  declare to wait for an acknowledgement from the sender and
  send itself an initial slave packet
ELSE IF the router was not waiting but an initial master
packet arrives THEN
  declare to wait for an acknowledgement from the sender and
  send itself an initial slave packet
ELSE IF the router was waiting and an initial slave packet
arrives THEN
  declare that this router no longer waits and send the
  content of the link state table as a not initial master packet
ELSE IF the router was not waiting and a not initial slave
packet arrives THEN
  add the link state information of the packet to the local
  link state table
  IF all neighbors where found THEN
    flood the content of the link state table to all neighbors
    except the sender of the above packet
  END IF
ELSE IF the router was not waiting and a not initial master
packet arrives THEN
  add the link state information of the packet to the local
  link state table and send the local link state table to the
  master
  IF all neighbors where found THEN
    flood the content of the link state table to all neighbors
    except the sender of the above packet
  END IF
END IF
```

Every time a flooding packet is received:

```
FOR all entries of the LSA DO
  IF source or destination equals this router THEN
    ignore update because this router knows better about the
    local situation
  ELSE
    IF an entry for the given source and destination already
    exists THEN
      IF the TIME of the entry is greater than or equals
      MAXAGE THEN
        always accept the new entry to assure that expired
        routes exhale swiftly but do not mark a change
      ELSE
        IF the current entry is newer than the one received
        THEN
          do not mark a change
        ELSE
          add the new entry and mark a change
        END IF
      END IF
    ELSE
      IF the TIME of the entry is greater than or equals
      MAXAGE THEN
        do not mark a change
      ELSE
        add the new entry and mark a change
    END IF
  END IF
OD
IF a change was marked THEN
  flood them to all neighbors except to the sender of the
  above flooding packet.
END IF
```

A IRV–Tool Documentation

Author: Christian Sternagel (csac3692@uibk.ac.at)
Program: IRV–Tool version 1.0
LincenseInfo: Open Source (GPL)
Date: 2004-01-12

A.1 Introduction

The IRV–Tool allows to visualize Routing–Settings which could occur e.g. in the internet. Via the tool it is possible to setup network topologies with arbitrary complexity in a simple way. Furthermore it approves to accomplish various simulations of Routing–Behavior with two different types of Routing–Protocols (RIP of the distance vector protocol family and OSPF of the link state protocol family).

The use of the IRV–Tool is easily acquired - thus with minimal effort the basics of routing can be learned by students.

The following sections describe how to install and use the IRV–Tool (developed by me for my Baccalaureate–Project).

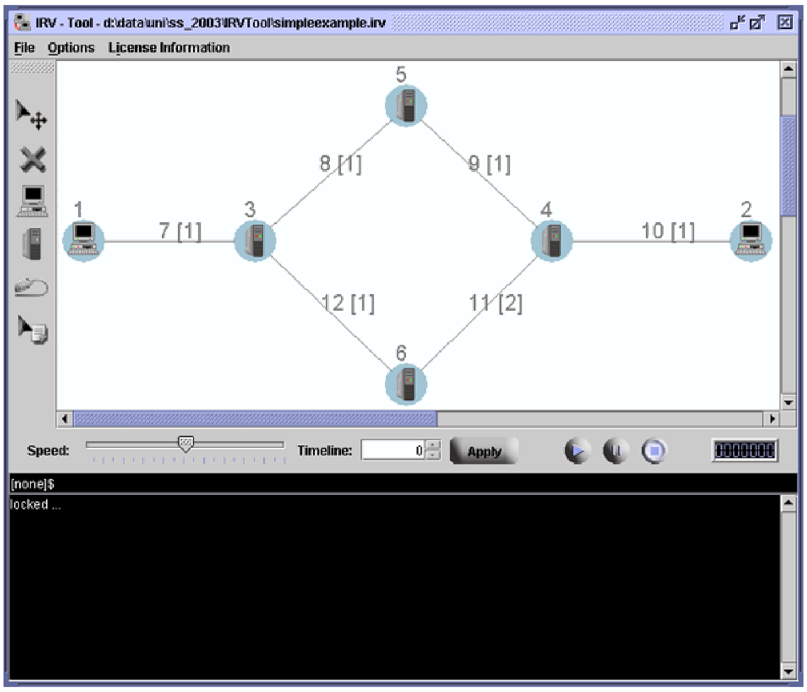
A.2 Installation

The IRV–Tool can be downloaded as a tar–Archive labeled 'irvtool.tar.gz' ore as a zip–Archive 'IRVTool.zip'. To extract the archive-file use:

```
$ tar -xzf irvtool.tar.gz
```

respectively

```
$ unzip IRVTool.zip
```



```
[none]$
locked ...
```

Figure 4: The IRV–Tool

Using windows only double–click on the zip–File to extract its contents. Now a jar–Archive ('irvtool.jar'), a documentation (irvt-doku.pdf), a simple example ('simpleexample.irv'), a batch-file ('irvtool.bat') for Windows as well as a Shell–Script ('irvtool.sh') for UNIX, a directory ('license') containing license information ('gpl.txt'), a directory ('src') where the sourcecode (*.java-files) can be found and this README–File are created in a new directory called 'IRVTool'.

To start the tool type:

UNIX

```
  $ chmod u+x irvtool.sh    # to make 'irvtool.sh' executable
  $ irvtool.sh              # to start the IRV-Tool
```

or

```
  $ java -jar irvtool.jar   # if the above command does not work
```

Windows

```
  > irvtool.bat             REM on older versions
```

or

```
  > irvtool.jar            REM WindowsXP
```

or

```
  > java -jar irvtool.jar  REM if none of the above commands works
```

In Windows it is also possible to execute the Tool by clicking on 'irvtool.bat' (older versions) or 'irvtool.jar' (WindowsXP).

A.3 Description

A.3.1 A Simple Example

Assume a Network–Topology like in Figure 5.

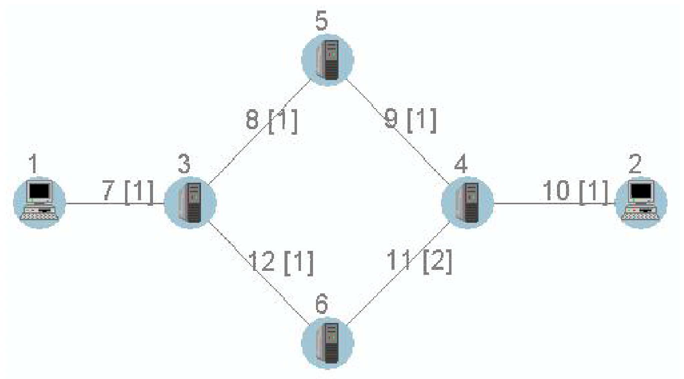


Figure 5: A Simple Topology

where 1 and 2 are hosts; 3, 4, 5 and 6 are routers and all connections except 11 (the link connecting router 4 to router 6) have a cost of 1. The connection 11 has a cost of 2. By means of this simple topology the basics of RIP should be shown. At first the routing tables contain the following data:

Router–3:

destination	cost	connection
3	0	local

Router–4:

destination	cost	connection
4	0	local

Router–5:

destination	cost	connection
5	0	local

Router–6:

destination	cost	connection
6	0	local

As we see at this stage the routers only know about themselves. After the first RIP–Update (in the order: 3, 4, 5, 6) the tables have changed to:

Router–3:

destination	cost	connection
3	0	local
1	1	7
5	1	8
6	1	12

Router–4:

destination	cost	connection
4	0	local
2	1	10
5	1	9
6	2	11

Router–5:

destination	cost	connection
5	0	local
1	2	8
2	2	9
3	1	8
4	1	9

Router–6:

destination	cost	connection
6	0	local
1	2	12
2	3	11
3	1	12
4	2	11

After the second RIP–Update at last, the tables contain following data:

Router–3:

destination	cost	connection
3	0	local
1	1	7
5	1	8
6	1	12
!2	3	8
4	2	8

Router–4:

destination	cost	connection
4	0	local
!2	1	10
5	1	9
6	2	11
1	3	11
3	2	9

Router–5:

destination	cost	connection
5	0	local
1	2	8
!2	2	9
3	1	8
4	1	9
6	2	8

Router–6:

destination	cost	connection
6	0	local
1	2	12
2	3	11
3	1	12
4	2	11
5	2	12

Now there exists a path from Host–1 to Host–2 via 3–5–4 (as highlighted by exclamation marks). This path is chosen at Router–3 because of its cumulative cost of 3 whereas the path 3–6–4 has a cumulative cost of 4 thus it is too expensive. At equal cost the order of updates determines the result. The above example can be visualized with the IRV–Tool. For that purpose the file 'simpleexample.irv' is needed (if you don't want to design the given topology by yourself). By simulating that file you can also observe how RIP responds to a modification in the topology. In 'simpleexample.irv' the connection 8 goes down from second 50 to second 70.

A.3.2 The User Interface

In terms of the IRV–Tool a network consists of hosts (symbol: computer with screen), routers (symbol: tower) and connections (symbol: network cable) which connect two nodes respectively. Here, the term node denotes hosts as well as routers. Hosts can only possess a single connection to a single router and cannot be connected among each other. Routers can possess arbitrary connections to arbitrary nodes. The toolbar on the left hand side serves to design and edit a topology quickly. There are following control elements (top down; see also Figure 6):

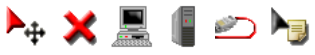


Figure 6: The Tools

Marker–Tool: Used to select (and afterwards delete with $\boxed{\text{DEL}}$) a single node or connection or to move several nodes together with all connections attached to them.

Remove–Tool: Used to remove several nodes and all connections attached to them by enframing them with pressed left mouse button and releasing the button when ready.

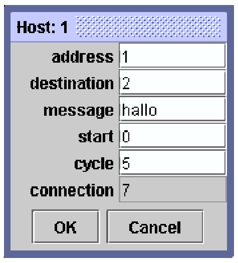
Host–Tool: Used to place a host on the drawing area.

Router–Tool: Used to place a router on the drawing area.

Connection–Tool: Used to connect two nodes to each other.

Edit–Tool: Used to look at and/or modify properties of any entity (node or connection). Instead of the Edit–Tool the right mouse button can be used as well.

The toolbar below controls all facets of simulations within the IRV–Tool. By modifying 'speed' the execution speed can be adjusted. At 'timeline' the current point in time is recorded (with seconds as unit). By modifying this value and clicking 'apply' the simulation can be moved to the given point in time (WARNING: going backwards in time is not possible, because the simulation is not deterministic concerning user input). Clicking 'play' starts a simulation. Within a simulation RIP–Packets are red and standard IP-Packets are green. 'pause' pauses a simulation and 'stop' aborts it while newly initializing all routing-tables (thus it is necessary to reopen a used *.irv–File for receiving Table–Entries as stored to the file).

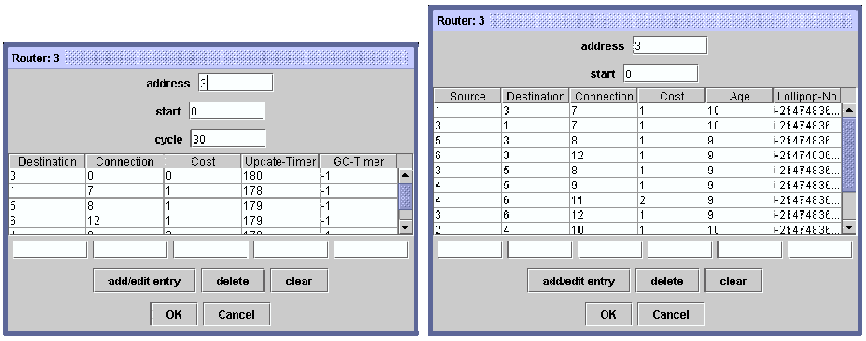


address	1
destination	2
message	hallo
start	0
cycle	5
connection	7

Figure 7: Host–Edit–Dialog

A.3.3 Edit Options

Hosts: The Host–Edit–Dialog (see Figure 7) allows to modify the address (only unambigous addresses are valid), the destination to which packets are sent (0 means 'no destination'), the content of messages from this host (default: 'hallo'), the acceleration time for sending IP-Packets (preconditioned that a destination was set) and the periodicity (every 'cylce' seconds) of these sendings. Moreover the connection to which this host is connected is listed.



address	3
start	0
cycle	30

Destination	Connection	Cost	Update-Timer	GC-Timer
3	0	0	180	-1
1	7	1	178	-1
5	8	1	179	-1
6	12	1	179	-1

address	3
start	0

Source	Destination	Connection	Cost	Age	Lollipop-No
1	3	7	1	10	-21474836...
3	1	7	1	10	-21474836...
5	3	8	1	9	-21474836...
6	3	12	1	9	-21474836...
3	5	8	1	9	-21474836...
4	5	9	1	9	-21474836...
4	6	11	2	9	-21474836...
3	6	12	1	9	-21474836...
2	4	10	1	10	-21474836...

Figure 8: Router–Edit–Dialog (left: RIP, right: OSPF

Routers: The Router–Edit–Dialog (see 8) allows to modify the address, the acceleration time for updates (-1 means 'never') and their periodicity (only for RIP; RIP–Default is 30 seconds). Also the Routing–Table of this router can be observed and modified.

Connections: The Connection–Edit–Dialog (see Figure 9) allows to modify the address and the cost (default value: 1). Via the cost the distance between to nodes is calculated. If there are two nodes connected by a single connection with cost 5, then these nodes are 5 units away from each

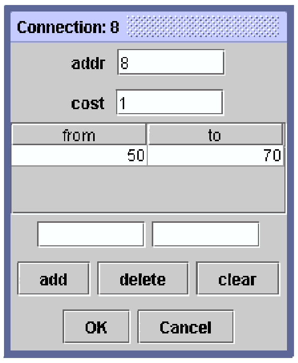


Figure 9: Connection–Edit–Dialog

other (unless there is another path between these two nodes with minor cost). Additionally there is a list of tuples of points in time in the manner (MIN, MAX) for each connection. Within this list any number of entries can be stored, indicating from which point in time (MIN) to which other (MAX) this connection should be down (interrupted connections are displayed dashed). Connections that are down behave in a manner as if they do not exist.

A.3.4 Saving A Topology As A Bitmap

By choosing 'Save As ...' from the File–Menu and adjusting the combobox 'Files of Type:' to '*.jpg files' it is possible, to save a topology as a bitmap (JPEG–Format). It must be pointed out that the size (width and height in pixels) of the bitmap depends on the portion of the network visible on the screen. In other words the whole visible canvas (the widget on which the painting is done) is saved as bitmap. Thus by changing the size of the Main–Window it is possible to control the size of the resulting bitmap.

A.3.5 Options

There are several options intended for controlling the exact behavior of routing in a given network (topology). First of all there are to Routing–Protocols that can be chosen.

RIP – Routing Information Protocol: This is the first and simpler protocol that can be chosen. More exhaustive information on RIP can be achieved from [2] and [1] or in minor complexity from the documentation of the IRV–Tool. At this point it is worth mentioning that RIP is a member of the Distance–Vector–Protocol–Family (DVP). Thus a distributed algorithm is used to calculate information about 'best' paths. Information is spread across the network by sending Distance–Vectors (DVs) to direct neighbors. These DVs consist of triples [to, cost, via], one for each destination node.

Settings: Currently implemented are three optional features as extension of RIP all based on existing features described in [2] and [1]. Thereby it concerns notably:

- Triggered Updates
- Split Horizon
- Split Horizon with poisonous Reverse

RIP Variables: There are also three variables that can be set for the RIP. By name:

The Garbage–Collection–Timer: Number of seconds an entry of a routing table should be kept after it has exceeded its validity.

Infinity: The number that should be treated as infinity.

Timer: Number of seconds a newly registered entry is valid without further acknowledgements.

After starting the IRV–Tool all these variables are set to default values like mentioned in [2] and [1].

OSPF – Open Shortest Path First: The second protocol is the OSPF protocol. As OSPF is a member of the Link–State–Protocol-family (LSP) every node holds a table (the so called 'link state table') containing information about the whole topology. That is the reason why every node can calculate the best path locally. OSPF is much more complex than RIP. Changes in the topology are advanced by a Flooding–Protocol that is part of OSPF and sends Link–State–Advertisments (LSA) across the network.

Settings: There is only one feature currently implemented:

Equal-cost Multipath (see [3]): This feature makes it possible to use more than one path if there are multiple paths between source and destination that have the same cost.

OSPF Variables: Four variables can be set for OSPF. By default they have values like in [3]. The variables are:

Hello–Interval: Number of seconds between checks for new or lost Neighbors.

Maximal-Age: The maximal age in seconds of a table entry befor it is discarded.

Maximal–Age–Difference: If an LSA with same sequencenumbers as the existing LSA arrives at a node it is always accepted in the case its age equals Maximal–Age. If the ages of the LSAs differ by more than Maximal–Age–Difference the LSA with the smaller age is considered as most recent otherwise the old one is kept.

Infinity: This number is used in the Lollipop–Algorithm (see [1]).

A.3.6 Simulating With The IRV–Tool

After designing a topology by arranging nodes and connection on the 'Canvas' a simulation can be started by pressing the 'Play' button. With 'Pause' it is possible to break a simulation and later resume it. And 'Stop' at last discards a simulation by setting all routing tables to their default values and the time to zero. To make a simulation faster it is possible to go forward in time by choosing the desired point in time and clicking 'Apply'. The clock (in the lower right corner of the IRV–Tool) can be switched between two modes by clicking on it. By default it shows the elapsed time in seconds. But it is also possible to show the time in the format 'hh:mm:ss'.

ATTENTION: After clicking 'Stop' all routing tables are empty, so it is necessary to reload a *.irv–File (by opening it from File→Open) to get the assignments of the routing tables that were saved to the file.

A.3.7 The Terminal

The Terminal is more in its test stage than completed nevertheless all described commands are fully functional (besides 'sleep' which only works correct if you let run the simulator with animation but not if you jump to a point of time by using 'Apply'). Some things that are very tedious without the Terminal (e.g. configuration of multiple nodes and connections) can be easily achieved by using the Terminal. First of all the Terminal only responds to user input if a simulation has been started (The 'Play' button was pressed without pressing 'Stop' afterwards). By using the Arrow–Keys (Arrow–Up and Arrow–Down) it is possible to access the Command–History of the Terminal (UNIX shell alike). Following commands are supported:

broadcast: you have to be logged in at a router

Syntax:

```
broadcast
```

When using RIP the RIP–Update of the current router is sent to all adjacent nodes. When using OSPF an OSPF–Hello–Packet is sent to each adjacent router.

clear: Syntax:

```
clear
```

Clears the Output–TextField of the IRV–Tool.

echo: Syntax:

```
echo <text>
```

Displays the specified text (terminated with a newline).

exit: Syntax:

```
exit
```

Closes the IRV–Tool.

help: Syntax:

```
help
```

Displays a list of all commands.

login: Syntax:

```
login <address-of-a-node>
```

Some commands can only be used if you are logged in at a node. When logged in a node, the prompt shows the address of this node. E.g. '[1]$'.

logout: Syntax:

```
logout
```

After logout, you are not logged in any node. The prompt shows '[none]$'.

ls (list): Syntax:

```
ls <address-of-an-entity>
```

All properties of an entity (node or connection) are listed.

ping: you have to be logged in
Syntax:

```
ping <address-of-a-node>
```

A Ping-Packet is sent from the node you are logged in to the node specified by <address–of–a–node>. The destination returns a Pong–Packet to the sender and a message is displayed on the Output–TextField.

route: Syntax:

```
route <address-of-a-router>
```

Displays the Routing–Table of the specified router.

run: Syntax:

```
run <path-of-script-file>
```

Executes the specified Script–File (see A.3.7).

send: you have to be logged in
Syntax:

```
send <address-of-a-node> [<message>]
```

Sends a packet from the current node to the node specified. The optional parameter <message> specifies a text-string to send. This String is displayed when the packet arrives at the destination.

set: used for nodes when logged in, otherwise for connections
Syntax for connections:

```
set <addr> {address | clear | cost | down | remove} [<value>]
    address ... set the address of connection <addr> to <value>
    clear   ... clear the list of Link-Failures from the connection
    cost    ... set the cost of the connection to <value>
    down    ... add a Link-Failure to the list where <value> is a
                pair of timestamps (separated by a blank) that
                specify from when to when the failure should occure
    remove  ... remove a Link-Failure specified by a pair of
                timestamps
```

Syntax for routers:

```
set address <new-addr>
```

Syntax for hosts:

```
set {address | cycle | destination | message | start} <valuet>
    Sets the specified property of the current host to the value
    <value>.
```

sleep: Syntax:

```
sleep <secondst>
```

Should only be used in Script–Files.

traceroute: you have to be logged in
Syntax:

```
traceroute <address-of-a-node>
```

Like the UNIX command 'traceroute'. A list of all nodes situated on the path between source and destination is displayed.

Running Script Files: With the 'run' command it is possible to execute ASCII–Files holding Terminal-commands. In addition there are some more options ('#' starts a comment until the end of the line):

repeat: Syntax:

```
repeat <times>
    # arbitrary code
;;
```

Repeats the code between <times> and ';;' <times> times.

for: Syntax #1:

```
for all {connections | hosts | nodes | routers}
    # arbitrary code
;
```

Repeats the code between the line with the for–command and ';' for all nodes (or connections or hosts ...; as specified). Within the for–block, the keyword 'this' can be used to access the current entity.

Syntax #2:

```
for {connections | hosts | nodes | routers} <address-list>
    # arbitrary code
;
```

Where <address–list> is a (blank separated) list of addresses.

For can be used within repeat, but not vice versa. To make the use of Script–Files more clearly here is an example.

```
# this is a comment
repeat 10
  for all routers
    broadcast
  ;
  sleep 10
;;
```

The above example shows a script that causes all routers to broadcast 10 times in succession and make a pause of 10 seconds between each time. This is another example:

```
clear
for all connections
  echo i am connection this
  ls this
;
for routers 2 4 6
  echo i am router this
  route this
;
```

Here all connections display their names and their properties after clearing the screen. Then the routers 2, 4 and 6 display their routing tables.

References

[1] C. Huitema, *Routing in the Internet, 2nd ed.* Prentice Hall, 2000.

[2] C. Hedrick, *RFC 1058 - Routing Information Protocol.* Rutgers University, 1988.

[3] J. Moy, *RFC 2328 - OSPF Version 2.* Ascend Communications, Inc., 1998.