Dynamic routing protocols

ROUTE TO SUCCESS

FRITZ REICHMANN

Dynamic routing protocols are very important for computers that must be accessible to the world at all times. What these protocols are and how they are configured is the subject of this article.

There are many computers that must always be accessible even when there is a breakdown in the network. Examples are mail servers, database servers and e-commerce systems. Secondly, maintaining fixed tables of routes between networks on a constantly-changing Internet would be an impossibly complex task. Anyone who needs a resilient network that can find its own way around any breakdowns or bottlenecks will need dynamic routing protocols.

Routing protocols are protocols that enable two routers to exchange notes with each other as to which networks can be accessed through them. By this means, and some clever algorithms, routers are able to do this job all by themselves, without administrative intervention, adapting the routes used whenever the network changes. In most cases routing protocols run on special hardware and software. But it is possible to achieve something similar under Unix/Linux.

The theory

The many demands imposed on routing protocols, plus the fact that the problem has been around a long time, has led to a whole range of protocols being developed. The main difference in terms of demand is between "internal" and "external" routing protocols. Internal protocols are designed to manage and distribute routing data within a small – or not so small – system of routers and/or computers. One example could be the network of a company with several departments at various locations. The job of an internal routing protocol would be to inform the entire company network how, for example, the databank server can be accessed from any location on the network.

If this company network was then to be connected to a larger network such as the Internet, it would be the job of an external protocol to distribute information across this larger network as to how the network of this company can be accessed. The company network is regarded from outside as one unit, and can be treated as an "autonomous system".

This working principle is similar in all routing protocols: A router has some kind of network connected to one of its interfaces. So it is also aware of how to access this network and informs its neighbours of this using the routing protocol. The neighbours then remember that they know someone that knows how to access this network and, in the man-

ner of village gossip (but more truthfully, we hope) they then in turn inform *their* neighbours. They remember that they know someone, who knows someone, who knows how to access the network and so on, until eventually everyone knows.

On this principle, a router will quite often receive messages from several of its neighbours that they know a route to the target network. The routes may be different, though all may be correct. From these routes, the router must select one that appears to be most suitable according to certain characteristics. In so doing, it must take care to avoid so-called "routing loops" which would result in the data going round in a circle. It must do this quickly so that the time taken until all the routers have the latest information – known as the "convergence period" – is as short as possible.

Routing Information Protocol

We will look in more detail at the three routing protocols RIP, OSPF and BGP because of their importance nowadays and their free availability. The "Routing Information Protocol", RIP for short, is perhaps the best known of the three. It exchanges routing information at pre-defined intervals of time and regards a path as optimal when it leads to the target via as few intervening nodes (known as *hops*) as possible. The choice of paths is worked out using the *distance vector algorithm*.

RIP has a number of disadvantages. Firstly, the pre-set time interval must elapse before RIP recognises and can act on an altered situation such as a failed connection. Secondly, the choice of routes may not be ideal if a diversion via several routers that have fast connections is competing with a route via few routers with slow connections. In this case RIP goes the slowcoach route and requires manual intervention to give preference to the diversion. Thirdly, RIP regards a router 16 hops away as unreachable, which means that the diameter of a network run using RIP cannot be larger than 15 routers. Fourthly, RIP in its old version 1 works only for TCP/IP address classes A, B and C without network masks. This makes version 1 useless for present requirements. Version 2 has at least resolved this last point, which is why RIP has been the most popular internal routing protocol until now.

Shortest Path First

"Open Shortest Path First", OSPF for short, is a more powerful internal routing protocol. "Open" in this context is to be understood in the sense of "Open Source" since OSPF is an open standard for the "Shortest Path First" algorithm. OSPF is a so-called "Link State Protocol". It is capable of processing network masks and can distribute data about the availability of connections faster than RIP. It takes into account, when selecting the optimal path, the speed of the connections in between, and, furthermore, the size of the network can in principle be as large as you like.

In order to be able to perform this task efficiently, OSPF sub-divides the system into three classes of domains. The first class is an *area*, which is a collection of just about any routers, networks and computers which exchange routing information with each other. The second class is the *backbone*, which connects all areas together into one autonomous system. Unlike areas, there is only one backbone. The areas are numbered, the backbone is then implicitly given number 0. The third class of domains are known as the *stub areas* which are domains from which only a single router leads to the backbone. The point of this sub-division is that the tables which must be maintained to control the

Configuration of fred, susie and cisco

Configuration of fred:

The Ethernet:

```
ifconfig eth0 192.168.0.1 netmask 255.255.255.0 broadcast 192.168.0.25➥
5 up
```

The serial link to susie: Because we used a store-bought null modem cable, we had to do without hardware handshaking:

```
pppd /dev/ttyS0 57600 nocrtscts persist local lock nodefaultroute \

   netmask 255.255.255.252 192.168.1.1:192.168.1.2 > /dev/null &
```

The dummy interface:

```
ifconfig dummy 10.0.0.1 netmask 255.255.255.240 broadcast 10.0.0.15 up
```

Switch on IP forwarding:

```
echo "1" > /proc/sys/net/ipv4/ip_forward
```

Configuration of susie

The serial link to fred:

```
pppd /dev/ttyS1 57600 nocrtscts persist local lock nodefaultroute \

   netmask 255.255.255.252 192.168.1.2:192.168.1.1 > /dev/null &
```

The Ethernet:

```
ifconfig eth0 192.168.0.3 netmask 255.255.255.0 broadcast 192.168.0.25➥
5 up
```

The dummy interface:

```
ifconfig dummy 10.0.2.1 netmask 255.255.255.240 broadcast 10.0.2.15 up
```

Switch on IP forwarding:

```
echo "1" > /proc/sys/net/ipv4/ip_forward
```

Configuring Cisco

```
interface Loopback0
ip address 10.0.1.1 255.255.255.0
no ip directed-broadcast
interface Ethernet0/0
ip address 192.168.0.2 255.255.255.0
no ip redirects
no ip directed-broadcast
no shutdown
```

routing information can be reduced in size. This means that not only less memory is needed, but also the data packets are processed more rapidly. In short, OSPF is more effective and more modern than RIP, but also a bit more complicated.

Border Patrol

The "Border Gateway Protocol", BGP for short, is an example of an external protocol. In this role it generally, though not exclusively, runs at the junctions (known as *peers*) between autonomous systems and processes data about the way in which other autonomous systems can be reached. Since, in so doing, it lists all the autonomous systems which have to be crossed on the way to the target, it is known as a *path vector protocol*.

BGP has various options for selecting an optimal route which allow it to take into account not so much technical but rather politically motivated grounds such as, for example, the cost of using a particular connection. Two BGP neighbours start off by exchanging their entire routing tables. After that they will only transmit amendments and "keep alive" messages, which are intended to monitor the availability of the connection between the BGP neighbours themselves. This method makes it possible for BGP to manage the routing information in a way that conserves resources. Nowadays BGP acts as the link in the Internet. It runs on most of the backbone routers of the big network operators.

In practice

There are programs that run under Unix and/or Linux which can execute routing protocols and even do it at no cost. The best known is the program *routed*, which comes as standard with Unix and is dedicated to the execution of RIP. Less well-known, but far more powerful, is *gated*, which has its own web page from where it can be downloaded. Still at the development stage, but also worth mentioning, is *zebra*, which unlike *gated* is a GNU project. This also has its own web page.

Because of the greater maturity of the program we will restrict our discussion to *gated*, and show by means of simple examples how you can configure the protocols RIPv2, OSPF and BGP in order to distribute routing information, and how you can replace a failed connection by means of a second connection without manual intervention. For this, a simple home network will serve, which in our example consists of a K6-400 running SuSE 6.2, a 486DX-80 running RedHat 6.0, each with its own 10BaseT network card, and a Cisco 2610. (Thanks to my boss for the 2610, and thanks to my girlfriend for putting up with all the mess in the living room!) For cabling we used a null modem cable to link the two PCs linked, together with crossed twisted-pair cable for the Ethernet interfaces.

Before the free-style comes the compulsory section, and this means that the kernels of the Linux PCs must be prepared for the hardware in the form of the network cards, the routing of the IP packets and the operation of a serial cable connection using PPP. The requirements are essentially the same as those for a Linux PC which is intended to connect a local network via an analogue modem to the Internet. In order to have a bit more room to manoeuvre for the configuration of network addresses the item "dummy-interfaces" should also be compiled.

With the kernel thus prepared, it is time to move on to the installation of the *gated* software. Download the latest openly available source code from version 3.5: at the time of writing this was the file *gated-3-5-11.tar.gz*. (Source code is important because BGP is not supported by the precompiled binaries.) The code is unpacked using *tar xzvf gated-3-5-11.tar.gz*, at which point you will have a new directory called *gated-3-5-11*. Unfortunately, *gated-3-5-11* doesn't have an easy *./configure; make; make install*, so for once it will be appropriate to actually read the file *INSTALL*.

The fastest way to get going is to enter the command sequence:

```
cd gated-3-5-11
mkdir src/obj
cp src/configs/linux-2.0 src/obj/Config
vi src/obj/Config
```

In this file the comment symbol before the line:

```
protocols      bgp  icmp ospf rip egp
```

should be deleted and the line underneath commented out. After this, compile the program with a simple *make*. Then installation can start with a *make install*. Unfortunately the binary *gdc* is written into */etc*, so it would be a good idea to move it using the command *mv /etc/gdc /usr/sbin* to a place where (in my opinion) a control program for a routing demon belongs. (Note that the version *gated-public-3_6*, which came out recently, has adopted the easy *configure* mechanism.)

Setup

Dummy interfaces should be set up on both Linux computers. These are logical interfaces to which one can assign IP addresses and they have the advantage of not failing as long as the computer shows the slightest sign of life. These *dummy* interfaces are given the IPs 10.0.0.1/28 (*fred*), 10.0.2.1/28 (*susie*) and 10.0.1.1/28 (*cisco*). Between the two connections, network connections are configured. The Ethernet of *fred* receives 192.168.0.1/24, the Ethernet of Cisco gets 192.168.0.2/24, the Ethernet of *susie* gets 192.168.0.3/24. The serial interface of *fred* is given 192.168.1.1/30, the serial interface of *susie* gets 192.168.1.2/30. We set the serial connections to run at 57600 baud (it can do more, but this is fast enough for our purposes.)

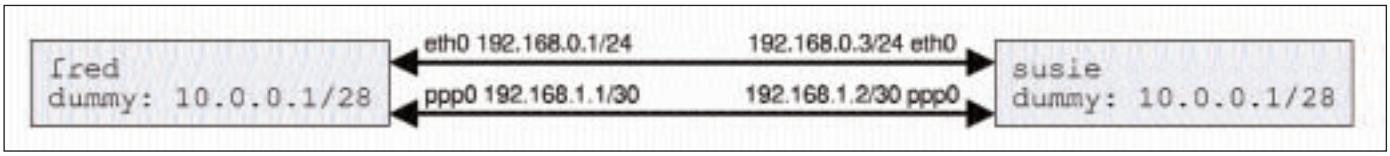

Fig. 1: Simple configuration

Having completed these preparations we now have a serial link between *fred* and *susie* and an Ethernet link, which we can construct with one crossed cable either between *fred* and *susie* or *fred* and Cisco. For the first two examples the Ethernet connection between *fred* and *susie* is to be used. Cisco can be switched off until then, which also provides some respite from its noisy power pack fan!

The computers should now exchange the addresses of their *dummy* interfaces via the routing protocol, because they cannot find these out simply from the configuration of the Ethernet and serial interfaces. Refer to Figure 1.

Setting up RIP between *fred* and *susie* is quick and simple. The files */etc/gated.conf* of *fred* and *susie* are identical:

```
rip yes {
      interface eth0
      version 2
      authentication simple “RIP“;
};

redirect no;
```

The command *rip yes* switches RIP on (this is the default anyway in *gated*.) Using the *interface* command, RIP is switched to the Ethernet. Next, we specify that we want to use RIP version 2. The command *authentication simple* followed by a string provides a simple way for the two computers to check each other, not as a security measure but to avoid any unintentional mis-configuration of a third router. The *redirect no* command at the end prevents the two computers changing the routes by means of ICMP redirects and thus getting our nice RIP all tangled up.

That's about it: *fred* learns via RIP the information that 10.0.2.1/28 is located on *susie* on the *dummy* interface and conversely *susie* learns that 10.0.0.1/28 is on *fred* on the *dummy* interface. If you try a *ping* on these IP addresses it runs through.

It is even more impressive with OSPF between *fred* and *susie*. In this case we have two connections between *fred* and *susie*: a fast Ethernet connection and a slow serial connection. What could be more obvious than taking the slow connection as an emergency backup if the fast one fails? OSPF can do that, because it also takes account of the speeds of the connections used. The files */etc/gated.conf* on *fred* and *susie* can be seen in Listing 1 and 2 respectively.

The command *routerid* defines the IP address under which the router sends its packets. If this is not specified, *gated* takes the IP address of the first interface it finds at random. In this instance we must take the IP of the *dummy* interface. If we were to take the address of the Ethernet interface and the Ethernet failed, the serial link could no longer leap in as an emergency solution because the packets are apparently being sent to the IP of the Ethernet adapter which in this scenario has just failed. Using *rip no* the RIP switched in by default is switched off since we want to play with OSPF now.

Our computers *fred* and *susie* are not backbone, so they will form part of area 1. The whole thing should run on the interfaces *eth0* and *ppp0*, again with a simple authentication string. At the end there is another *export* instruction. This is necessary because OSPF only passes on routes from home which it has learnt via OSPF. In order that it will also pass on the directly connected networks to the *dummy* interface, these direct routes have to be exported to OSPF.

Listing 1: */etc/gated.conf* from *fred*

```
routerid 10.0.0.1;

rip no;

ospf yes {
      area 1 {
            authtype simple;
            interface eth0 ppp0 {
                  authkey “OSPF“;
            };
      };
};

redirect no;

export proto ospfase type 2 {
      proto direct {
            ALL;
      };
};
```

Listing 2: */etc/gated.confauf susie*

```
routerid 10.0.2.1;

rip no;

ospf yes {
     area 1 {
           authtype simple;
           interface eth0 ppp0 {
                 authkey “OSPF“;
           };
     };
};

redirect no;

export proto ospfase type 2 {
      proto direct {
            ALL;
      };
};
```

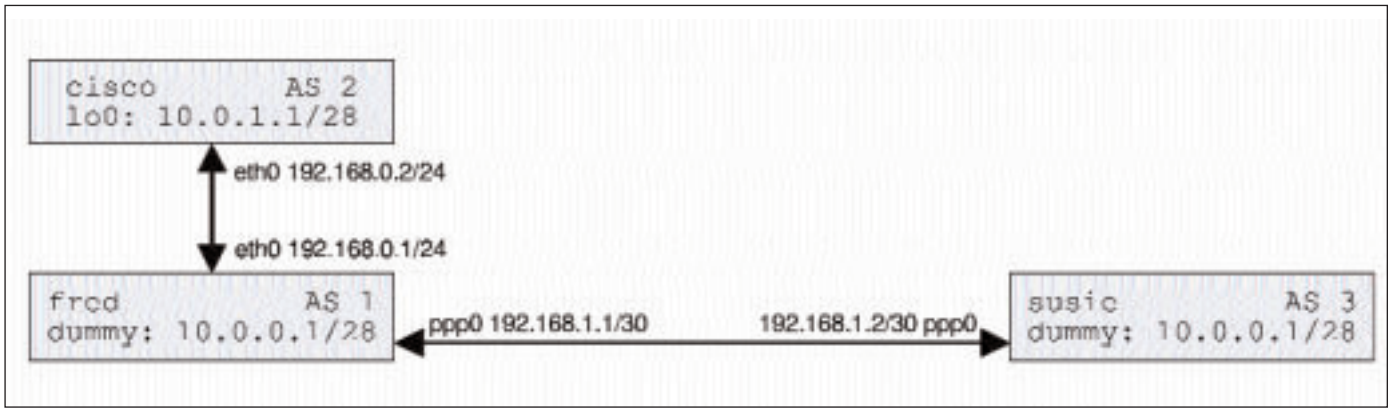

Fig. 2: A somewhat more complex situation

Now *susie* and *fred* again learn reciprocally via OSPF the IPs of the respective *dummy* interfaces. It gets exciting now, when we start a *ping 10.0.2.1* on *fred*. This runs through as expected. Now, we simulate a connection failure by simply pulling the Ethernet cable out of the computer. At first, there is no answer to the *ping*. After about thirty seconds another one turns up, but this time with a delay which is no longer just 1-2, but some 50 milliseconds. *fred* has learnt from OSPF that the way to the dummy interface of *susie* is no longer via the Ethernet, but the serial cable. This is certainly slower, but now the best possible way.

Into the big wide world

To liven things up we shall now connect the routers as follows: *fred* with *susie* via the serial cable and *fred* with Cisco via the crossed Ethernet cable. This means we have three computers in a row. *susie* is meant to be autonomous system number 3, *fred* the one with number 1 and Cisco will be given the number 2. The whole thing looks like in Figure 2.

/etc/gated.conf on *susie*

```
autonomoussystem 3;

routerid 10.0.2.1;

rip no;

bgp yes {
      preference 50;
      group type external peeras 1 {
      peer 192.168.1.1;
      };
};

redirect no;

export proto bgp as 1 {
      proto direct;
```

Configuration of Cisco

```
router bgp 2
redistribute connected
neighbor 192.168.0.1 remote-as 1
no auto-summary
```

/etc/gated.conf on *fred*

```
autonomoussystem 1;

routerid 10.0.0.1;

rip no;

bgp yes {
      preference 50;

      group type external peeras 2 {
            peer 192.168.0.2;
      };
      group type external peeras 3 {
            peer 192.168.1.2;
      };
};

redirect no;

export proto bgp as 2 {
      proto bgp as 3 {
            all;
      };
      proto direct;
};

export proto bgp as 3 {
      proto bgp as 2 {
            all;
      };
      proto direct;
```

This is pretty similar to the previous OSPF configuration. Firstly, the membership of the autonomous system is defined on each computer. *routerid* defines the IP of the *dummy* interface as the source address from which the data packets are sent by BGP. RIP is switched off again and BGP switched on with *bgp yes*. The *preference* command sets the routes learnt via BGP to a somewhat higher preference than is used as standard so that the BGP routes are not overwritten (by ICMP redirects, for example.)

Next to be defined are the IP addresses at which the respective neighbouring autonomous systems can be reached. Since the BGP implementation of *gated* doesn't pass on the routes to other autonomous systems from home, we must force them to be passed on using *export* commands as are the directly connected *dummy* interfaces. For BGP this was already the case, after which, using *ping* and *traceroute* you will see that it is possible to reach each of the other computers from any one of them. ■

Info

Merit Gated Consortium
http://www.gated.org

GNU Zebra
http://www.zebra.org

■