THE ANNALS OF "DUNĂREA DE JOS" UNIVERSITY OF GALATI
FASCICLE III, 2012, VOL. 35, NO. 1, ISSN 1221-454X
ELECTROTECHNICS, ELECTRONICS, AUTOMATIC CONTROL, INFORMATICS

ROBUST ROUTING INFORMATION UPDATING IN IPV6 NETWORKS

Cosmin Adomnicăi, Viorel Mînzu

Faculty of Automatics, Computer science, Electrical and Electronic Engineering, University "Dunarea de Jos" of Galaţi, Romania

Abstract: The mechanism that classical routing protocols use to transmit routing updates and to maintain adjacency can be susceptible to packets loss and to bandwidth limitation. We propose a new method for transmitting routing information and hello information which can assure delivery even if the links are lossy or their capacity is limited. This method is adapted to IPv6 packets and, with the implementation of a routing algorithm, it builds the routing tables.

Keywords: networking, IPv6, routing, stability, bandwidth.

INTRODUCTION

Classical routing protocols like RIP (Routing Information Protocol), OSPF (Open Shortest Path First), EIGRP (Enhanced Interior Gateway Routing Protocol) use discrete packets to transmit hello information or to transmit routing information updates. If the link's capacity can accommodate user traffic, management traffic and routing protocol's traffic, the routers should not lose adjacency. On the other hand, if the links bandwidth does not suffice to the traffic needs, packets are lost in the transit. Along with user packets, routing protocol's packets can be lost and this leads to adjacency loss.

The most common cause for instability in a network is the link failure (Markopoulou, *et al.*, 2004). In many cases the customers are connected to the backbone using leased lines which are more volatile (Shaikh, *et. al.*, 2002). In a study, published by Watson, *et al.* (2003), made by monitoring OSPF behavior in a service provider network, a major cause of instability was the unstable links. By analyzing the behavior of some routing protocols under lossy links condition, we can see that their stability is susceptible to packet loss. The simulations made by Adomnicăi and Mînzu (2012) measured the percent of lost traffic due to instability induced by lossy links (Table 1).

RIP routing protocol, in the default configuration, implements long timers until a neighbor is declared down. So, even if some Hello packets are lost, until the timer expires, there is a large probability that a Hello packet will arrive and the adjacency will be maintained. The main disadvantage is that RIP will react to slow to a failure in the network

For OSPF the degree of time in which adjacency is lost increases with the bandwidth limitation. At 40% packet loss, OSPF loses adjacency for 75.91% of the simulation time.

EIGRP, for 10% packet loss, loses already adjacency for 29.75% of the simulation time. This percent increases with the degree of packet loss and, at 40% packet loss, for EIGRP, traffic is not forwarded for 70.17% of the simulation time.

Table 1 Total time of lost traffic for RIP, OSPF and EIGRP using lossy links

	Percent of time for which traffic is interrupted due to adjacency lost			
Routing protocol	10% loss	20% loss	30% loss	40% loss
RIP	0%	0%	0%	0%
OSPF	2.91%	8.75%	13.58%	75.91%
EIGRP	29.75%	29.84%	69.75%	70.17%

In the simulations made by Adomnicăi (2012) it was shown that bandwidth limitation can have a destabilizing role, depending on the degree of limitation. For 40 Mbps generated traffic with 10 Mbps bandwidth limitation the values from Table 2 were obtained. As it can be seen, RIP does not loose adjacency because of its the long timers.

OSPF and EIGRP, even if there are two links between the routers, lose adjacency and traffic is not forwarded. For OSPF, with a single link between the routers, the adjacency is lost for 24% of the simulation time. If another link is added between the routers, the adjacency is lost for 34% of the simulation time.

For EIGRP, the percent in which the traffic is lost due to the adjacency loss is higher than OSPF. If between the routers is a single link, traffic is not forwarded between the routers for 82% of the simulation time. If another link is added then traffic is not forwarded for 47% of the simulation time.

Table 2 Total time of lost traffic under bandwidth limitation for RIP, OSPF and EIGRP

Routing protocol	Number of links between two simulated routers	Total time of lost traffic	Percent of lost traffic time
RIP	1	0 seconds	0%
	2	0 seconds	0%
OSPF	1	288 seconds	24%
	2	408 seconds	34%
EIGRP	1	984 seconds	82%
	2	564 seconds	47%

ROBUST UPDATING USING DESTINATION OPTIONS EXTENDED HEADER

As we saw, routing protocols are sensitive to traffic conditions. If the links are broken and this induces packet loss or if the traffic is bandwidth limited, routing protocols can think that the corresponding neighbors are down and, as a consequence, the routes through them are withdrawn. In a packet loss scenario this can be a good think. If the link is faulty then routing protocols should avoid it. But, if the link is functional and if it is a leased line and its speed is much lower than the standard 100 Mbps 100Base-T then routing protocols, in a situation of heavy user traffic, can think that the corresponding neighbor is down and the traffic is not forwarded using this link. Adomnicăi and Danilescu (2011) and Adomnicăi and Mînzu (2012) proposed and tested a robust method for transmitting routing information and hello packets. This method was developed for IPv6 networks and uses an extended header. From all the available extended headers the Destination Options Extended Header was chosen. The other headers can change or can be eliminated from packets when traversing a router. The Destination Options header is used when additional information needs to be transmitted between source and destination and will not be changed or dropped by routers on transit (see Deering, 1998; Kozierok, 2005).

This method, called INFXCHG (INFormation eXCHanGe), transmits updates sequentially. To each packet that leaves the network interface the 40 bytes Destination Options extended header is attached which contains the following fields:

- 4 Standard fields: Next header (1 byte), Header length (1 byte), Option Type (1 byte), Option Length (1 byte).

- 6 additional fields: Network (16 bytes), Mask (1 byte), Cost (1 byte), Network acknowledge (16 bytes), Mask acknowledge (1 byte); Route Operation (1 byte).

The principle, used in transmission of the updates, is based on persistent ARQ strategy (useful information in Fairhurst and Wood, (2002); Peterson and Davie, 2011). Using this mechanism (Fig.1.), an update is sent continuously until an acknowledge is received. After this, the next update will be send or the updating mechanism will stop.

If there is no update or acknowledge to be sent, the 6 additional fields in Destination Options Extended header are filled with zeros. This kind of packet will act as a Hello packet maintaining adjacency between neighboring routers.

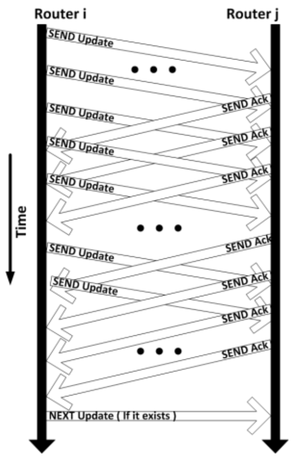

Fig.1. Updating mechanism for INFXCHG

When a packet is received (Fig.2.), INFXCHG verifies if it contains Destination Options Extended Header. After that it verifies if there is an update or acknowledge. If there is an update the routing buffer and the routing table are modified and the update is added to the buffer with acknowledges that need to be sent. If in the Extended Header an acknowledge is found then the corresponding entry from the buffer with updates that need to be sent is removed.

When a packet needs to be sent (Fig.3.), INFXCHG verifies the buffer with updates to be sent and the buffer with the acknowledges to be sent. If there are found any entries, the last ones are added to the Extended Header.

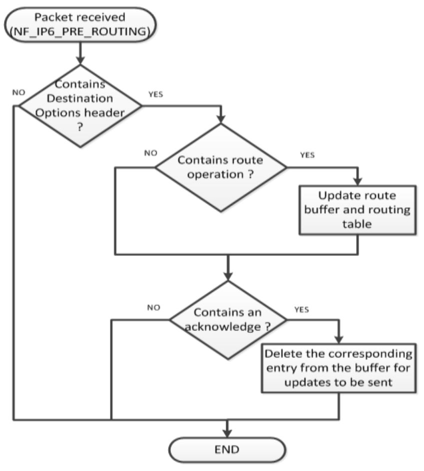
Packet received
(NF_IP6_PRE_ROUTING)
Contains Destination Options header ?
NO
YES
Contains route operation ?
NO
YES
Update route buffer and routing table
Contains an acknowledge ?
NO
YES
Delete the corresponding entry from the buffer for updates to be sent
END

Fig.2. Operations done when receiving a packet

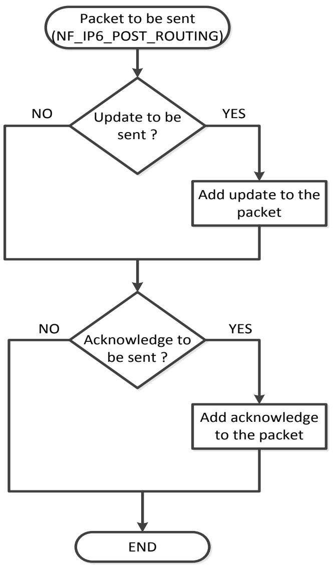

Fig.3. Operations done when sending a packet

If there is no user traffic then the updates cannot be sent. Therefore, a traffic generator had to be implemented in order to provide the means to send updates and to maintain adjacency between routers.

IMPLEMENTATION AND SIMULATION OF INFXCHG

The implementation for INFXCHG was done in C++ and the software was made as a Linux Kernel module. This choice was done in order to benefit of the processing speed. Processes that run in the kernel memory space have greater priority than the processes that run in the user memory space.

In order to add the extra header packets must be captured. For this, the NETFILTER platform was used. Two functions were registered: a function that treats incoming packets (Fig.2.) and a function for outgoing packets (Fig.3.). The development was done using KDevelop on OpenSUSE 11.3 Linux distribution.

Debugging when developing a kernel module was difficult because breakpoints cannot be inserted. So the work was done inside a virtual machine in order to save the running state before testing different stages. If the running of INFXCHG would cause a kernel hang-up, the virtual machine could be restored at the previous state. In this way the system restart could be avoided.

For each important event that appeared in the functioning of INFXCHG a debug message was generated. Along with the message, the Linux kernel attached a timestamp to the message. The events that generated a debug message are:

- Initializing messages;
- Interface UP/DOWN events;
- New routing update; this event triggers a listing of the routing table;
- Messages on INFXCHG exit.

For each active network interface a thread is created in order to generate dummy traffic in the periods of user inactivity. Each thread watches for traffic and if there is a pause it starts generating until user traffic starts to pass the router.

In each test scenario (Fig.4. – Fig.7.), for every router, a virtual machine was created. The number of network interfaces added to the virtual machine was according to the number of links. The end stations (S1 – S4) were simulated by running processes on the virtual machine itself. Otherwise, for the five routers test scenario would have been necessary nine virtual machines. In order to virtually connect the network interfaces multiple LAN Segments were created. The virtual network interfaces were connected to the LAN segments in order to communicate between them.

The bandwidth limitation could be achieved in two ways:

- Using the virtual machine software's options to manage network interface's bandwidth;

- By applying the limitation directly in the virtual machine by means of using traffic shaping options in Linux.

The first choice in which the limitation was applied at the virtual machine level was not stable enough. After testing this method, we saw that there were fluctuations from the established value.

The second choice was more stable and the variations from the established value were diminished.

Using the timestamp provided with each message we could compute the time difference between different events. Therefore, to compute the stabilization time, the time difference the module initialization and the last update received.

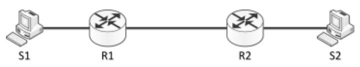

Fig.4. Two routers test scenario

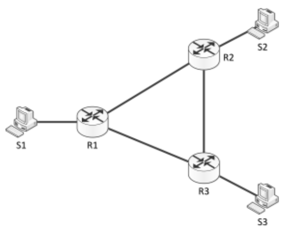

Fig.5. Three routers test scenario

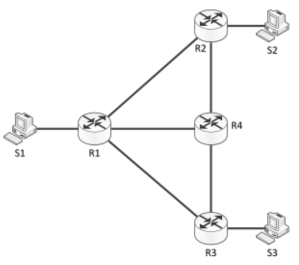

Fig.6. Four routers test scenario

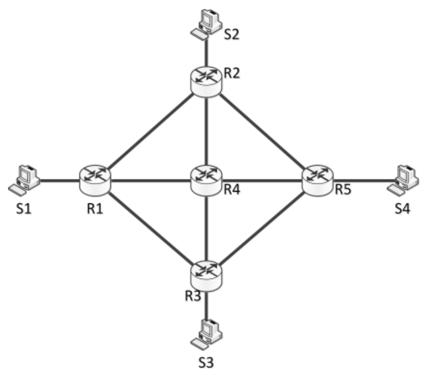

Fig.7. Five routers test scenario

Based on the received routing updates, using Bellman-Ford algorithm, the routes are constructed. In this implementation each link has a cost of 1 and, thus, the route cost is the number of hops between current router and destination. Updates are sent step by step until all the routers have the same image of the network. In choosing the links on which to send an update, the Split Horizon principle is used. This means that an update is not sent on the same interface on which it arrived.

When an update arrives, the cost is incremented, the routing table is updated and a new update with the new cost is constructed and sent to the neighbors. In order to update the kernel's routing tables the NETLINK platform was used.

By having implemented a routing algorithm, INFXCHG became a routing protocol. Using Bellman-Ford algorithm INFXCHG is a distance vector routing protocol.

The forwarding process is done in kernel. When a packet arrives, based on the routing table, the kernel's networking subsystem will choose the outgoing interface.

To determine the stabilization time, all the routers, excepting the last, were started. We waited for the routing tables to stabilize and R_n router (last router) was started and we measured the time difference between the INFXCHG start-up time and the time when the network stabilized.

If T_j, $j \in \{n-1,n\}$, is the time at which the routing table of routers R_1, R_2, …, R_j is in a stable state, then the stabilization time is:

$$(1)\ \Delta = T_n - T_{n-1}$$

After computing all the times for different bandwidth limitations, the values for INFXCHG from the Table 3 are obtained.

COMPARATIVE STUDY BETWEEND INFXCHG AND OTHER ROUTING PROTOCOLS

To calculate the initial stabilization time S_t for OSPF and EIGRP we used:

$$(2)\ S_t = T_S - T_{Start}, \text{ where}$$

T_s – the time moment at which the network had stabilized;

S_t – time difference between T_s and the start time moment of routing protocol on the last router in the network;

In the simulations T_{Start}=20. In each scenario, between routers, 100 Mbps of traffic is generated.

Applying the bandwidth limitations, the values for S_t from Table 3 were obtained.

For OSPF and EIGRP the default parameters are used. With these default parameters, OSPF stabilizes initially in over 50 seconds. As the bandwidth limitation is increased, the stabilization times for OSPF and EIGRP also increase. INFXCHG is the fastest because of its design. As the bandwidth limitation increases, for INFXCHG, the stabilization time increases but at a much slower rate than other routing protocols.

Table 3 Initial stabilization time for OSPF, EIGRP and INFXCHG

		S_t		
Test scenario	Bandwidth limitation	*OSPF*	*EIGRP*	*INFXCHG*
2 routers	100Mbps	55.32	0.0000181	0.000007
	10Mbps	85.23	26.19	0.000009
	4Mbps	85.23	55.00	0.000018
	2Mbps	85.23	120.07	0.000017
	1Mbps	97.71	123.08	0.000017
	512Kbps	65.23	15.00	0.000017
	256Kbps	65.23	15.00	0.000018
	128Kbps	65.23	15.01	0.000019
3 routers	100Mbps	65.01	10.00	0.004102
	10Mbps	75.06	95.00	0.009043
	4Mbps	157.46	67.99	0.014
	2Mbps	251.97	66.00	0.02
	1Mbps	247.66	79.04	0.04
	512Kbps	253.90	255.75	0.06
	256Kbps	65.06	30.01	0.18
	128Kbps	65.06	30.02	0.65
4 routers	100Mbps	65.01	10.00	0.007
	10Mbps	75.00	76.20	0.01
	4Mbps	75.01	210.00	0.02
	2Mbps	75.01	135.08	0.03
	1Mbps	92.03	258.59	0.05
	512Kbps	253.69	76.95	0.09
	256Kbps	65.02	30.06	0.32
	128Kbps	65.03	30.11	0.70
5 routers	100Mbps	55.23	10.00	0.009929
	10Mbps	242.49	41.20	0.02
	4Mbps	75.24	33.00	0.05
	2Mbps	75.24	30.00	0.10
	1Mbps	252.92	177.13	0.20
	512Kbps	251.05	110.36	0.42
	256Kbps	264.28	40.03	0.98
	128Kbps	108.60	35.00	1.96

The number of routers has not such big impact on the stabilization time. The main factor than lead to the increase of S_t is the bandwidth limitation (Fig.8.).

Another set of simulations is done for RIP, OSPF, EIGRP and INFXCHG in order to measure the stabilization time without user traffic or bandwidth limitation (Table 4 and Fig.9.). The links were 100BaseT at 100 Mbps. For INFXCHG, in the absence of user traffic, the generators started in order to transmit the updates and to maintain adjacency. The other routing protocols benefit from the advantage of sending multiple updates at once.

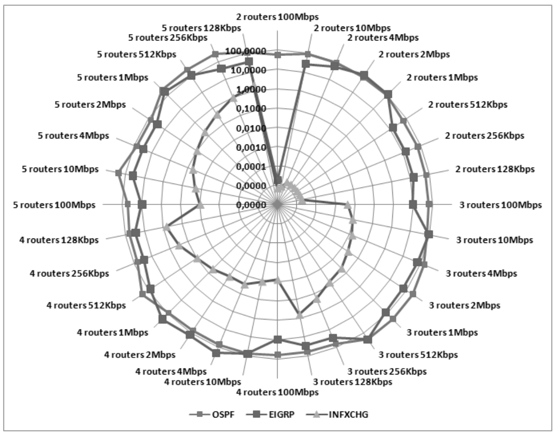

Fig.8. Initial stabilization time for OSPF, EIGRP and INFXCHG

OSPF stabilizes last in all scenarios because of its design. In two and three routers test scenarios the fastest is RIP, but as the number of routers increases, it will stabilize more slowly.

Overall, EIRPG is the fastest. In two and three routers test scenarios it is second, but in four and five routers test scenarios it is the fastest.

INFXCHG has overall good performance. The updates are sent sequentially and this is the reason for which it stabilizes slower than EIGRP. In the absence of bandwidth limitation, the stabilization time increases with the number of routers. This is normal; as the network diameter increases the time needed to propagate de information increases.

Table 4 Stabilization time for RIP, OSPF, EIGRP and INFXCHG without traffic and bandwidth limitation

Scenario	2 routers	3 routers	4 routers	5 routers
RIP	0.00001	0.00001	20.00001	20.00001
OSPF	55.37	55.43	55.14	55.41

EIGRP	0.00004	0.00004	0.00004	0.00005
INFXCHG	0.000007	0.004	0.007	0.009

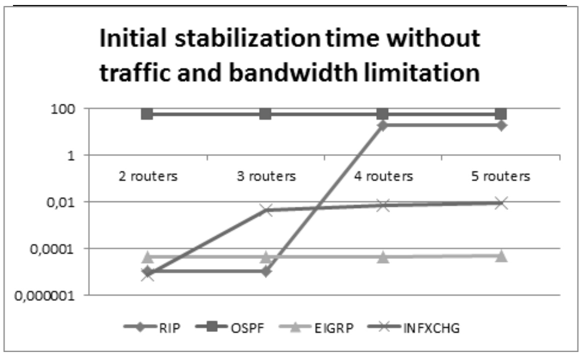

Fig.9. Stabilization time for RIP, OSPF, EIGRP and INFXCHG without traffic and bandwidth limitation

CONCLUSION

If the links between routers are faulty and they cause packet loss or if the links are bandwidth limited then routing protocols can enter an unstable state. In this state adjacency is lost and restored leading to route adding and withdrawing. The final consequence is that the user traffic could be lost because of the instability. In the links are lossy then some user traffic is lost anyway. But if the links are bandwidth limited then user traffic can be lost without the link being faulty.

INFXCHG was designed to avoid instability problems cause by lossy or bandwidth limited links. Its robustness derives from how the updates are sent. Each updates have to be acknowledged. In order to ensure that an update reaches its neighbor, it is sent until an acknowledge is received. The link load is approximately 2,66% with the advantage of robustness.

Another main advantage is that the adjacency is constantly maintained. So, if a neighbor or link goes down, INFXCHG detects this event immediately. This way, the duration of routing loops, until the network stabilizes, is minimized.

Being implemented as a kernel module, INFXCHG is fast and this has minimum impact on the packet processing latency.

For now, we consider only the number of hops in calculating the routes cost. For the future we want to consider other parameters like: link delay, link load, link bandwidth, link stability, router stability.

Because the updates are sent sequentially, INFXCHG is not the fastest routing protocol. Depending of the packet size, the Destination Options Header could be enlarged in order to accommodate multiple updates and acknowledges at once. This way, the network could stabilize more rapidly.

REFERENCES

Adomnicăi, C. and M. Danilescu (2011). Routing updates using Destination options network header in IPv6 networks. In: *Proceedings of International Conference on Computer Technology and Development*, **vol.2**, pp. 469-473, Chengdu, China.

Adomnicăi, C. (2012). Routing protocols behaviour under bandwidth limitation. In: *Proceedings of International Conference on Information and Computer Networks,* **vol. 27**, pp. 52-57, Singapore.

Adomnicăi, C. and V. Mînzu (2012). Method for adjacency information updating in IPv6 networks. In: *Proceedings of the 16th International Conference on System Theory, Control and Computing Joint Conference SINTES 16, SACCS 12, SIMSIS 16*, Sinaia, Romania.

Deering, S. and R. Hinden (1998). *Internet Protocol, Version 6 (IPv6) Specification, Request for comments 2460*, Internet Engineering Task Force.

Fairhurst, G. and L. Wood (2002). *Advice to link designers on link Automatic Repeat reQuest (ARQ), Request for comments 3366*, Internet Engineering Task Force.

Kozierok, Charles M. (2005). *The TCP/IP Guide: A Comprehensive, Illustrated Internet Protocols Reference*, William Polloc.

Markopoulou, A., G. Iannaccone, S. Bhattacharyya, C.-N. Chuah and C.Diot (2004), Characterization of failures in an IP backbone. In: *Proceedinngs of IEEE INFOCOM*, Hong Kong.

Peterson L. Larry and S. Bruce Davie (2011). *Computer Networks a system approach* (Fifth Edition), Morgan Kaufmann.

Shaikh, A., C. Isett, A. Greenberg, M. Roughan and J. Gottlieb (2002). A Case Study of OSPF Behavior in a Large Enterprise Network. In: *Proceedings of ACM SIGCOMM Internet Measurement Workshop*.

Watson, D., F. Jahanian and C. Labovitz (2003). Experiences with Monitoring OSPF on a Regional Service Provider Network. In: *Proceedings of International Conference on Distributed Computing Systems*, pp. 204–213.