OSPF TOI

Contents

OSPF Basics and Live Network Analysis Using OSPF Simulator:

OSPF Basics:

Like most IP protocols, OSPF routers communicate in terms of protocol packets. OSPF runs directly over the IP network layer, doing without the services of UDP or TCP (which are used by RIP and BGP, respectively). When a router receives an IP packet with IP protocol number equal to 89, the router knows that the packet contains OSPF data. Stripping off the IP header, the router finds an OSPF packet. One particular type of OSPF packet, together with its IP encapsulation, is shown below.

OSPF uses services of the IP header as follows:

- Since most OSPF packets travel only a single hop, namely, between neighboring routers or peers, the TTL in the IP packet is almost always set to 1. This keeps broken or misconfigured routers from mistakenly forwarding OSPF packets.

- The Destination IP address in the IP header is always set to the neighbor's IP address or to one of the OSPF multicast addresses AllSPFRouters (224.0.0.5) or AllDRouters (224.0.0.6).

- An OSPF router uses IP fragmentation/reassembly when it has to send a packet that is larger than a network segment's MTU. However, most of the time, a router can avoid sending such a large packet, sending an equivalent set of smaller OSPF packets instead (this is sometimes called semantic fragmentation). For example, IP fragmentation cannot be avoided when a router is flooding and LSA that is itself larger (or close to larger) than the network segment's MTU. That is to say, there is no semantic fragmentation for OSPF LSAs. IP fragmentation is also unavoidable when the router has so many neighbors on a broadcast segment that the size of the Hello packet exceeds the segment's MTU.

- A router sends all OSPF packets with IP precedence of Internetwork Control, in hope that this setting will cause OSPF packets to be preferred over data packets (although in practice, it seldom does).

All OSPF packets begin with a standard 24-byte header, which provides the following functions.

- *An OSPF packet type field.* There are five separate types of OSPF protocol packets: Hello packets (type = 1), used to discover and maintain neighbor relationships and Database Description packets (type = 2), Link State Request packets (type = 3), Link State Update packets (type = 4), and Link State Acknowledgment packets (type = 5), all used in link-state database synchronization.

  The OSPF Router ID of the sender. Thus the receiving router can tell which OSPF router the packet came from.

  A packet checksum. This allows the receiving router to determine whether the packet has been damaged in transit; if so, the packet is discarded.

Authentication fields. For security, these fields allow the receiving router to verify that the packet was indeed sent by the OSPF router whose Router ID appears in the header and that the packet's contents have not been modified by a third party.

An OSPF Area ID. This enables the receiving router to associate the packet to the proper level of OSPF hierarchy and to ensure that the OSPF hierarchy has been configured consistently.

Sample OSPF packet has been shown below.

Neighbor Discovery And Maintenance:

All routing protocols provide a way for a router to discover and maintain neighbor relationships (also sometimes called peer relationships). A router's neighbors, or peers, are those routers with which the router will directly exchange routing information.

In OSPF, a router discovers neighbors by periodically sending OSPF Hello packets out all of its interfaces. By default, a router sends Helios out an interface every 10 seconds, although this interval is configurable as the OSPF parameter HelloInterval. A router learns the existence of a neighboring router when it receives the neighbor's OSPF Hello in turn.

The part of the OSPF protocol responsible for sending and receiving Hello packets is called OSPF's Hello protocol (not to be confused with the old NSFNET routing protocol of the same name). The transmission and reception of Hello packets also enables a router to detect the failure of one of its neighbors; if enough time elapses (specified as the OSPF configurable parameter RouterDeadInterval, whose default value is 40 seconds) without the router's receiving a Hello from a neighbor, the router stops advertising the connection to the router and starts routing data packets around the failure.

Besides enabling discovery and maintenance of OSPF neighbors, OSPF's Hello protocol also establishes that the neighboring routers are consistent in the following ways.

- The Hello protocol ensures that each neighbor can send packets to and receive packets from the other. In other words, the Hello protocol ensures that the link between neighbors is bidirectional.

- The Hello protocol ensures that the neighboring routers agree on the HelloInterval and RouterDeadInterval parameters. This ensures that each router sends Helios quickly enough so that losing an occasional Hello packet will not mistakenly bring the link down.

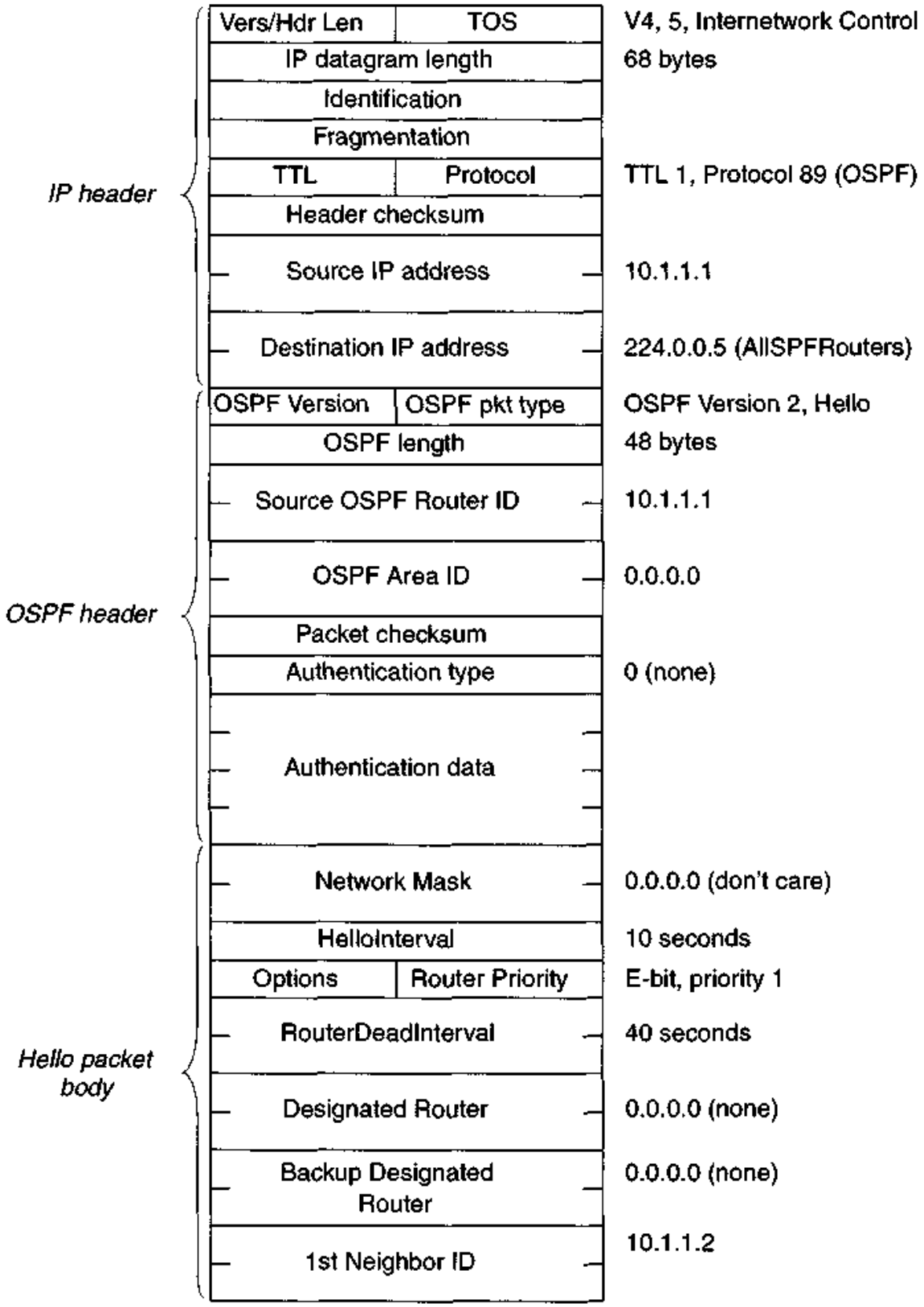


Figure 4.6 An OSPF Hello packet.

Database Synchronization:

Database synchronization in a link-state protocol is crucial. As long as the database remains synchronized, a link-state protocol's routing calculations ensure correct and loop-free routing. It is no surprise, then, that most of the protocol machinery in a linkstate protocol exists simply to ensure and to maintain synchronization of the link-state database. Of the 5 OSPF protocol packet types, 4 are used for database synchronization.

Database synchronization takes two forms in a link-state protocol. First, when two neighbors start communicating, they must synchronize their databases before forwarding data traffic over their shared link, or routing loops may ensue. Second, there is the continual database resynchronization that must occur as new LSAs are introduced and distributed among routers. The mechanism that achieves this resynchronization is called *reliable flooding*.

Link State Advertisements (LSAs):

Each OSPF router originates one or more LSAs to describe its local part of the routing domain. Taken together, the LSAs form the link-state database, which is used as input to the routing calculations. In order to provide organization to the database and to enable the orderly updating and removal of LSAs, each LSA must provide some bookkeeping information, as well as topological information. All OSPF LSAs start with a 20-byte common header, shown below , which carries this bookkeeping information.

LS Age
Options LS Type
Link State ID
Advertising Router
LS Sequence Number
LS Checksum
Length

Ls Type: **router LSA, network LSA, summary LSA, ABR LSA, AS external LSA… etc.**

Following is the sample format/example of router LSA:

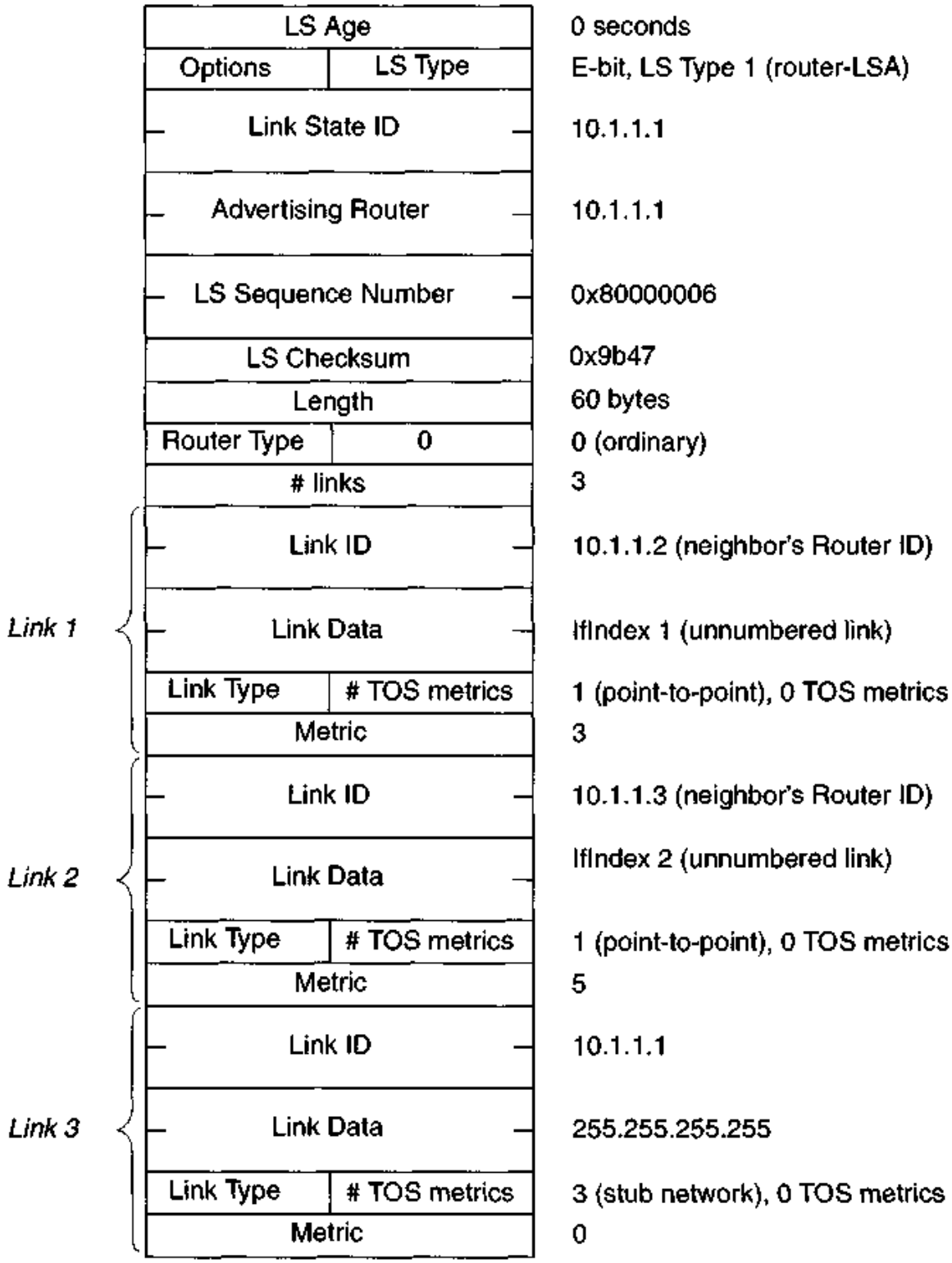


Figure: Sample Router-LSA

Live Network Analysis Using OSPF Simulator:

Will explain initial database synchronization and reliable flooding using simulating following sample network. Whole network has been simulated with OPEN OSPF SIMULATOR running internally OPEN OSPF stack. Below mentioned picture is GUI of OPEN OSPF SIMULATOR.

Sample network:

1) Two network each having p-p link in their own area (area 0 and area 1) has been shown below.

Area 0 (Called backbone are) contains following nodes:

10.0.0.1
10.0.0.2
10.0.0.3
10.0.0.4

Area 1 contains following nodes:

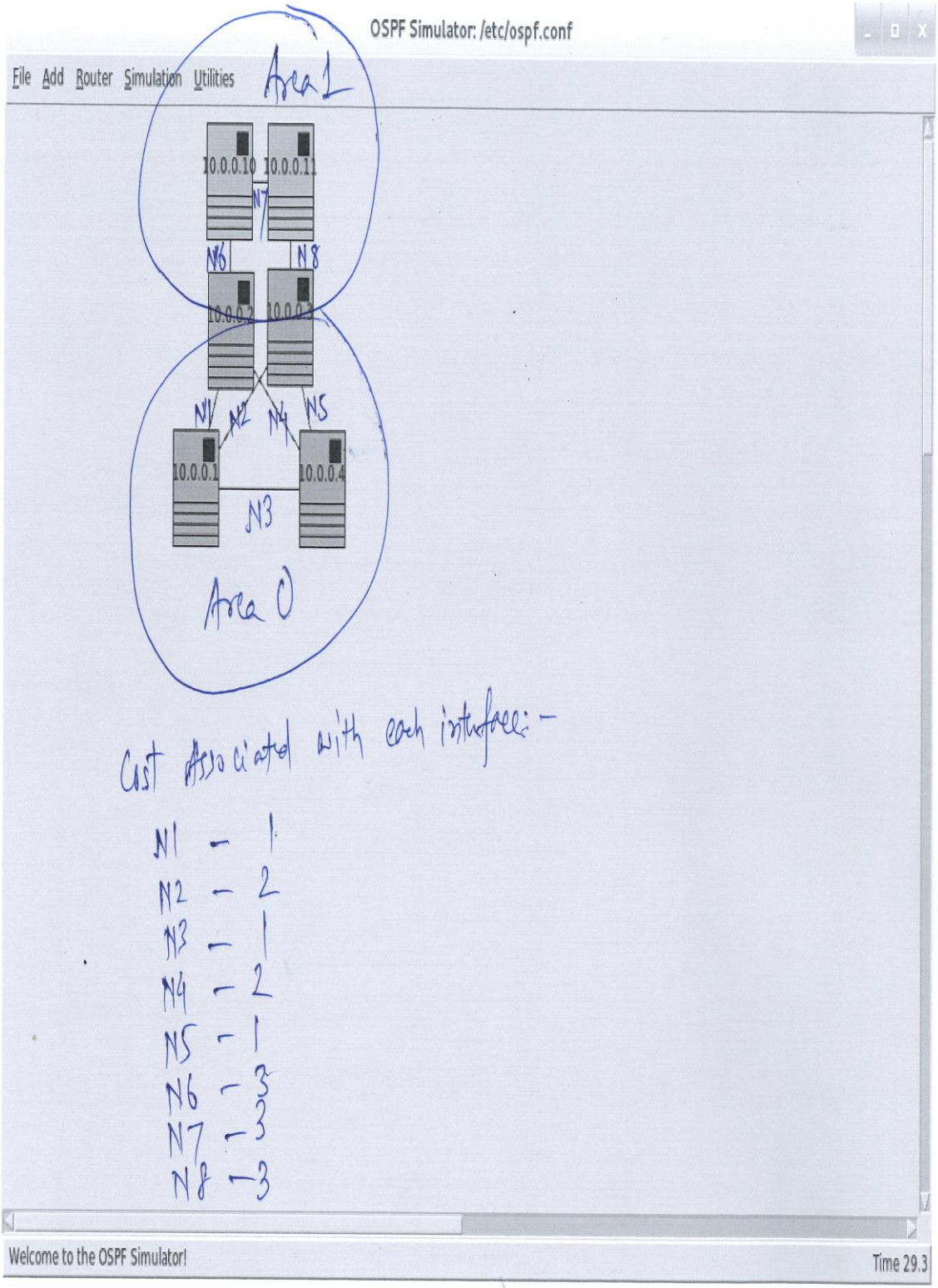
10.0.0.2
10.0.0.3
10.0.0.10
10.0.0.11

Will explain database synchronization within area 0 and within area 1. Then will observe flooding within each area. Clearly 10.0.0.2 and 10.0.0.3 acts as area border router (ABR).

Cost associated with each point to point interface in both direction are as follows:

Interface Idx	Cost
N1	1
N2	2
N3	1
N4	2
N5	1
N6	3
N7	3
N8	3

Clearly total 8 interfaces, 5 (N1, N2, N3,N4,N5) within area 0 and 3 (N6,N7,N8) within area 1.



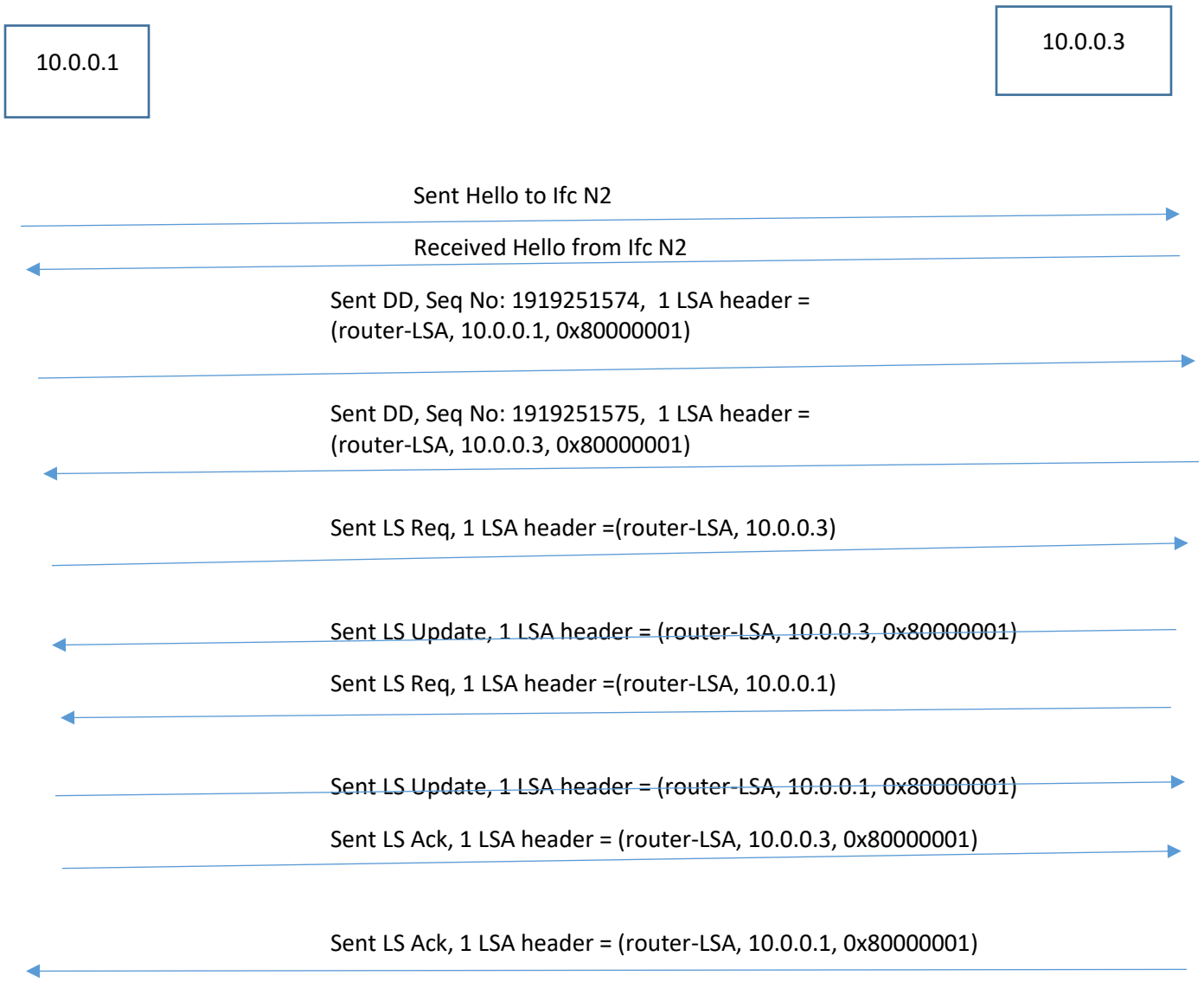
Cost Associated with each interface:-

N1 - 1
N2 - 2
N3 - 1
N4 - 2
N5 - 1
N6 - 3
N7 - 3
N8 - 3

Database synchronization within area 0 and Reliable Flooding:

Once network comes up with open network simulator, following is the sequence of DD initialization between various nodes. As interfaces are configured and added, hello exchange happens between neighbors. When 2 way exchange has happened, dd exchange starts between neighbor nodes. Successful completion of dd exchange leads neighbor being fully adjacent post which it will start participating in reliable flooding.

DD sync between node 10.0.0.1 and node 10.0.0.4:



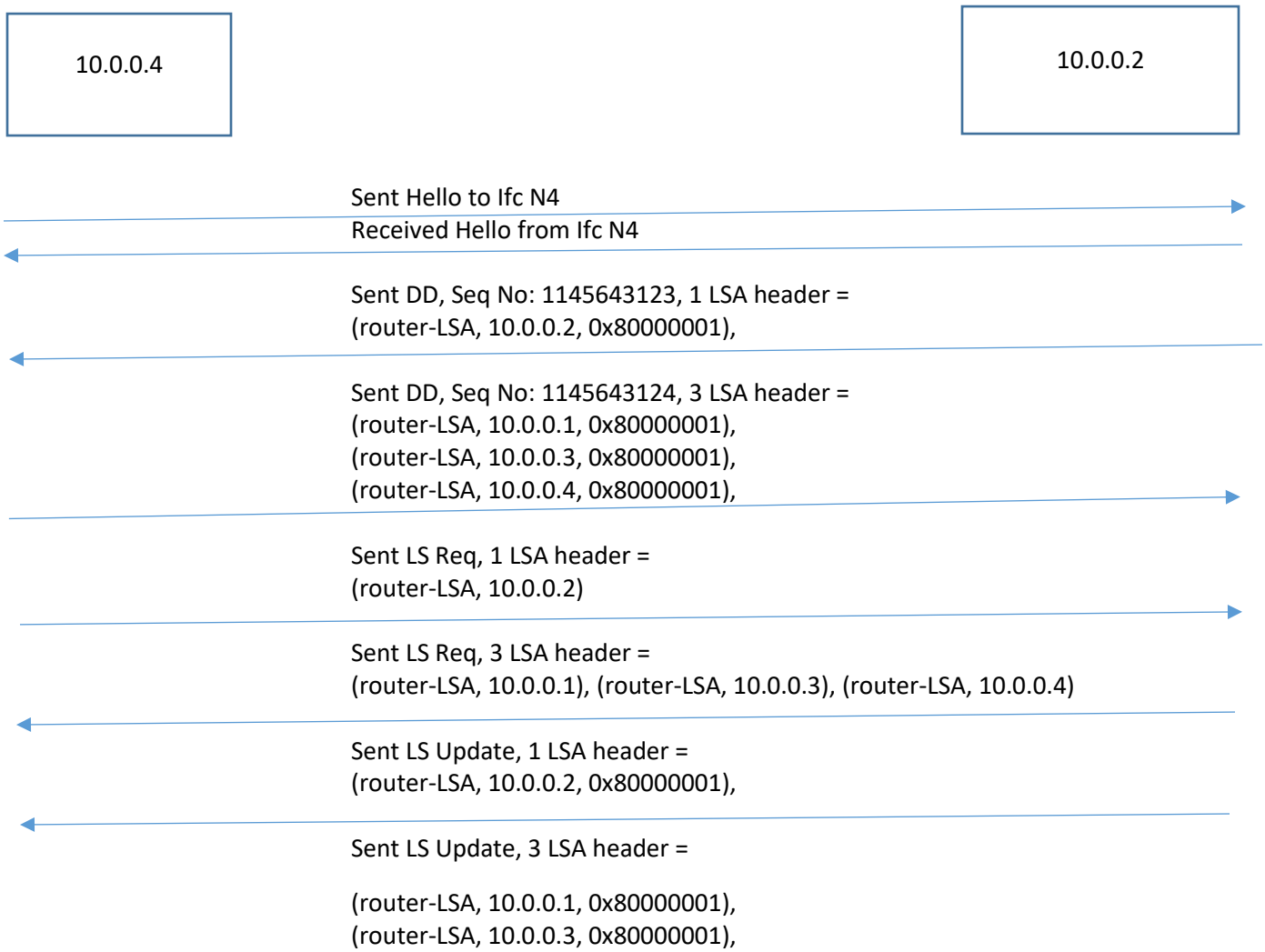
After this, both nodes are fully adjacent and on both nodes, in their LSA database, they have router-LSA of others.

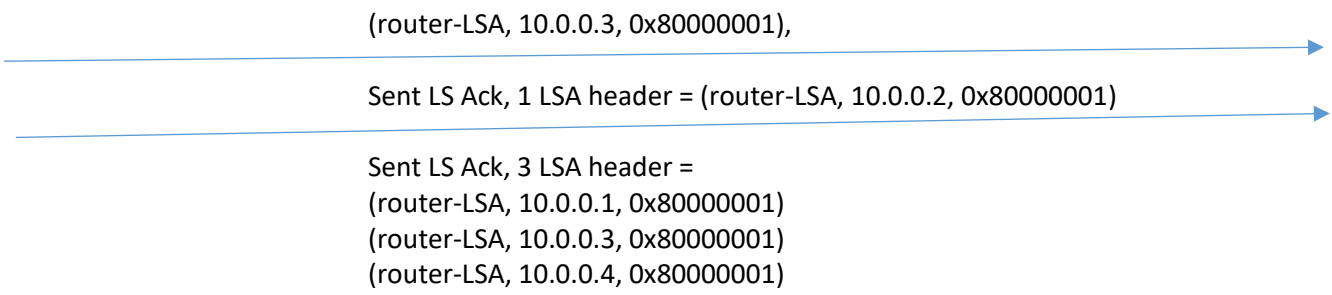
DD sync between node 10.0.0.3 and node 10.0.0.4:

At this point, node 3 and 4 are fully adjacent and node 3 has 3 router LSA in it's database: (router-LSA,10.0.0.1), (router-LSA, 10.0.0.3), (router-LSA, 10.0.0.4). But node 1 has only 2 router-LSAs. Since node 1 and 3 are already adjacent, hence node 3 will do **reliable flooding (router-LSA, 10.0.0.4)** over interface N2 as evident from following logs of **open ospf stack**:

"(10.0.0.3) Sent LsUpd 1.0.0.3->224.0.0.5 Router ID 10.0.0.3 Area ID 0.0.0.0 Packet Length 64 Count 1 LS_Upd_Pkt_Length 36 LSA(1,10.0.0.4,10.0.0.4)Seq no 0x80000001 Ifc N2 .
(10.0.0.1) OSPF.203: New LSA(1,10.0.0.4,10.0.0.4) .
(10.0.0.1) OSPF.202: Sent LsAck 1.0.0.1->224.0.0.5 Router ID 10.0.0.1 Area ID 0.0.0.0 Packet Length 44 Ls_Ack_Pkt_Length 20 LSA(1,10.0.0.4,10.0.0.4)Seq no 0x80000001 Ifc N2. "

DD sync between node 10.0.0.4 and node 10.0.0.2:





After this, node 2 and 4 will become fully adjacent. With this, both node 4 and node 2 will have following router-LSAs in their database:

(router-LSA, 10.0.0.1, 0x80000001)
(router-LSA, 10.0.0.2, 0x80000001)
(router-LSA, 10.0.0.3, 0x80000001)
(router-LSA, 10.0.0.4, 0x80000001)

Since 4 is adjacent to 3 and 3 is adjacent to 1, hence new router-LSA for 10.0.0.2 will be **reliably flooded** back to node 1 via node 3.

" (10.0.0.4) OSPF.202: Sent LsUpd 1.0.0.4->224.0.0.5 Router ID 10.0.0.4 Area ID 0.0.0.0 Packet Length 64 Count 1 LS_Upd_Pkt_Length 36 LSA(1,10.0.0.2,10.0.0.2)Seq no 0x80000001 Ifc N5
(10.0.0.3) OSPF.201: Received LsUpd 1.0.0.4->224.0.0.5 Router ID 10.0.0.4 Area ID 0.0.0.0 Packet Length 64 Count 1 LS_Upd_Pkt_Length 36 LSA(1,10.0.0.2,10.0.0.2)Seq no 0x80000001.
(10.0.0.3) OSPF.203: New LSA(1,10.0.0.2,10.0.0.2)
(10.0.0.3) OSPF.202: Sent LsUpd 1.0.0.3->224.0.0.5 Router ID 10.0.0.3 Area ID 0.0.0.0 Packet Length 64 Count 1 LS_Upd_Pkt_Length 36 LSA(1,10.0.0.2,10.0.0.2)Seq no 0x80000001 Ifc N2
(10.0.0.1) OSPF.201: Received LsUpd 1.0.0.3->224.0.0.5 Router ID 10.0.0.3 Area ID 0.0.0.0 Packet Length 64 Count 1 LS_Upd_Pkt_Length 36 LSA(1,10.0.0.2,10.0.0.2)Seq no 0x80000001
(10.0.0.1) OSPF.203: New LSA(1,10.0.0.2,10.0.0.2)
(10.0.0.1) OSPF.202: Sent LsAck 1.0.0.1->224.0.0.5 Router ID 10.0.0.1 Area ID 0.0.0.0 Packet Length 44 Ls_Ack_Pkt_Length 20 LSA(1,10.0.0.2,10.0.0.2)Seq no 0x80000001 Ifc N2
(10.0.0.3) OSPF.202: Sent LsAck 1.0.0.3->224.0.0.5 Router ID 10.0.0.3 Area ID 0.0.0.0 Packet Length 44 Ls_Ack_Pkt_Length 20 LSA(1,10.0.0.2,10.0.0.2)Seq no 0x80000001 Ifc N5. "

At this point, all 4 nodes namely 1-3, 3-4 and 2-4 are adjacent over interface N2, N5 and N4 and all have four router-LSA in their database. As and when new adjacencies will form, LSA database size will grow, reliable flooding will happen over existing adjacencies and in this way, at some point, all nodes in the area will be adjacent to their neighbors and all will have at some pint similar no of entries in their LSA database and the checksum of all LSAs should be equal on all nods in the same area.

Sequence number of a particular LSA may change if timer expires and new instance of that LSA is generated or if a particular interface goes down and comes up in which case new router LSA with incremented seq no is generated.

Following log from open OSPF stack shows that router-LSA of node 10.0.0.1 has changed most probably because of addition of new link N1 (between 10.0.0.1 to 10.0.0.2) and hence it is regenerating the it's router-LSA with incremented new sequence no as follows:

"(10.0.0.1) OSPF.204: Originating LSA(1,10.0.0.1,10.0.0.1).
(10.0.0.4) OSPF.201: Received LsUpd 1.0.0.3->224.0.0.5 Router ID 10.0.0.3 Area ID 0.0.0.0 Packet Length 76 Count 1 LS_Upd_Pkt_Length 48 LSA(1,10.0.0.1,10.0.0.1)Seq no 0x80000002
(10.0.0.3) OSPF.201: Received LsUpd 1.0.0.1->224.0.0.5 Router ID 10.0.0.1 Area ID 0.0.0.0 Packet Length 76 Count 1 LS_Upd_Pkt_Length 48 LSA(1,10.0.0.1,10.0.0.1)Seq no 0x80000002
(10.0.0.2) OSPF.201: Received LsUpd 1.0.0.4->224.0.0.5 Router ID 10.0.0.4 Area ID 0.0.0.0 Packet Length 76 Count 1 LS_Upd_Pkt_Length 48 LSA(1,10.0.0.1,10.0.0.1)Seq no 0x80000002.
(10.0.0.4) OSPF.203: New LSA(1,10.0.0.1,10.0.0.1)
(10.0.0.3) OSPF.203: New LSA(1,10.0.0.1,10.0.0.1)
(10.0.0.2) OSPF.203: New LSA(1,10.0.0.1,10.0.0.1)
(10.0.0.1) OSPF.202: Sent LsUpd 1.0.0.1->224.0.0.5 Router ID 10.0.0.1 Area ID 0.0.0.0 Packet Length 76 Count 1 LS_Upd_Pkt_Length 48 LSA(1,10.0.0.1,10.0.0.1)Seq no 0x80000002 Ifc N2
(10.0.0.4) OSPF.202: Sent LsUpd 1.0.0.4->224.0.0.5 Router ID 10.0.0.4 Area ID 0.0.0.0 Packet Length 76 Count 1 LS_Upd_Pkt_Length 48 LSA(1,10.0.0.1,10.0.0.1)Seq no 0x80000002 Ifc N4
(10.0.0.3) OSPF.202: Sent LsUpd 1.0.0.3->224.0.0.5 Router ID 10.0.0.3 Area ID 0.0.0.0 Packet Length 76 Count 1 LS_Upd_Pkt_Length 48 LSA(1,10.0.0.1,10.0.0.1)Seq no 0x80000002 Ifc N5. "

Now sequence no of router-LSA 10.0.0.1 has increased from 0x80000001 to 0x80000002 . Now this router-LSA will be flooded back to all other adjacent nodes within the area. Soon all nodes in the area will have recent instance of router-LSA (router-LSA, 10.0.0.1, 0x80000002) in their database.

The **open network simulator** provides facility to attach to the monitoring ports of any of the four nodes in the area 0 (10.0.0.1, 10.0.0.2, 10.0.3, 10.0.0.4) and inspect their LSA database live. LSA database of any node will give full network view within that area.

Attaching to the node 10.0.0.1:

```
[root@Centos7 src]# /usr/local/bin/ospfd_mon -p 48849
Connecting to Centos7:48849 ...connected

OSPF Router ID: 10.0.0.1          # AS-external-LSAs:      0
ASE checksum:  0x0                # ASEs originated:       0
ASEs allowed:   0                 # Dijkstras:             7
# Areas:        1                 # Nbrs in Exchange:      0
MOSPF enabled:          yes               Inter-area multicast:     yes
Inter-AS multicast: no            In overflow state:       no
ospfd version:   2.16

10.0.0.1> areas
Area ID       #Ifcs   #Routers #LSAs  Xsum      Comments
0.0.0.0       3       4        4      0x25a97   demand-capable
10.0.0.1> interface
Phy          Addr          Area          Type    State    #Nbr #Adj Cost
N1           0.0.0.0       0.0.0.0        P-P        1    1    1
N2           0.0.0.0       0.0.0.0        P-P        1    1    2
N3           0.0.0.0       0.0.0.0        P-P        1    1    1
10.0.0.1> neighbor
Phy          Addr          ID            State    #DD  #Rq  #Rxmt
N1           1.0.0.2       10.0.0.2       Full    0    0    0
N2           1.0.0.3       10.0.0.3       Full    0    0    0
N3           1.0.0.4       10.0.0.4       Full    0    0    0
10.0.0.1> database 0
Type        LS_ID       ADV_RTR     Seqno   Xsum  Age
  1      10.0.0.1      10.0.0.1 0x80000002 0x4d6a  732
  1      10.0.0.2      10.0.0.2 0x80000003 0xd9f6  729
  1      10.0.0.3      10.0.0.3 0x80000003 0x4881  729
  1      10.0.0.4      10.0.0.4 0x80000003 0xeab6  728
                  # LSAs: 4
                  Database xsum: 0x25a97

10.0.0.1> adv 1 10.0.0.1 10.0.0.1 0
          LS age:             785
          LS Options:         0x26
          LS Type:            1
          Link State ID:      10.0.0.1
          Advert. Rtr.:       10.0.0.1
          LS Seqno:           0x80000002
          LS Xsum:            0x4d6a
          LS Length:          72
                  // Router-LSA body
                  Router type:      0x0
                  # links:  4
```

```
                // Link #0
                Link ID: 10.0.0.4
                Link Data:      0.0.0.5
                Link type:      1
                # TOS metrics:  0
                Link cost:      1
                // Link #1
                Link ID: 10.0.0.3
                Link Data:      0.0.0.3
                Link type:      1
                # TOS metrics:  0
                Link cost:      2
                // Link #2
                Link ID: 10.0.0.2
                Link Data:      0.0.0.1
                Link type:      1
                # TOS metrics:  0
                Link cost:      1
                // Link #3
                Link ID: 1.0.0.1
                Link Data:      255.255.255.255
                Link type:      3
                # TOS metrics:  0
                Link cost:      0
10.0.0.1> adv 1 10.0.0.2 10.0.0.2 0
        LS age:         820
        LS Options:     0x26
        LS Type:        1
        Link State ID:  10.0.0.2
        Advert. Rtr.:   10.0.0.2
        LS Seqno:       0x80000003
        LS Xsum:        0xd9f6
        LS Length:      60
                // Router-LSA body
                Router type:    0x0
                # links:  3
                // Link #0
                Link ID: 10.0.0.4
                Link Data:      0.0.0.7
                Link type:      1
                # TOS metrics:  0
                Link cost:      2
                // Link #1
                Link ID: 10.0.0.1
                Link Data:      0.0.0.2
                Link type:      1
                # TOS metrics:  0
                Link cost:      1
```

```
                // Link #2
                Link ID: 1.0.0.2
                Link Data:      255.255.255.255
                Link type:      3
                # TOS metrics:  0
                Link cost:      0
10.0.0.1> adv 1 10.0.0.3 10.0.0.3 0
        LS age:         838
        LS Options:     0x26
        LS Type:        1
        Link State ID:  10.0.0.3
        Advert. Rtr.:   10.0.0.3
        LS Seqno:       0x80000003
        LS Xsum:        0x4881
        LS Length:      60
                // Router-LSA body
                Router type:    0x0
                # links:  3
                // Link #0
                Link ID: 10.0.0.4
                Link Data:      0.0.0.9
                Link type:      1
                # TOS metrics:  0
                Link cost:      1
                // Link #1
                Link ID: 10.0.0.1
                Link Data:      0.0.0.4
                Link type:      1
                # TOS metrics:  0
                Link cost:      2
                // Link #2
                Link ID: 1.0.0.3
                Link Data:      255.255.255.255
                Link type:      3
                # TOS metrics:  0
                Link cost:      0
10.0.0.1> adv 1 10.0.0.4 10.0.0.4 0
        LS age:         851
        LS Options:     0x26
        LS Type:        1
        Link State ID:  10.0.0.4
        Advert. Rtr.:   10.0.0.4
        LS Seqno:       0x80000003
        LS Xsum:        0xeab6
        LS Length:      72
                // Router-LSA body
                Router type:    0x0
                # links:  4
```

```
                // Link #0
                Link ID: 10.0.0.3
                Link Data:      0.0.0.10
                Link type:      1
                # TOS metrics:  0
                Link cost:      1
                // Link #1
                Link ID: 10.0.0.2
                Link Data:      0.0.0.8
                Link type:      1
                # TOS metrics:  0
                Link cost:      2
                // Link #2
                Link ID: 10.0.0.1
                Link Data:      0.0.0.6
                Link type:      1
                # TOS metrics:  0
                Link cost:      1
                // Link #3
                Link ID: 1.0.0.4
                Link Data:      255.255.255.255
                Link type:      3
                # TOS metrics:  0
                Link cost:      0
10.0.0.1>
```

So, node 10.0.0.1 has one area, three interface, three neighbor and 4 router-LSAs which are global within that area.

Details of router-LSA ID 10.0.0.2 is as follows: It has three point-to-point links each having peer link id and cost the details of which is as follows:

Peer Link ID	Cost
10.0.0.1	1
10.0.0.4	2
10.0.0.2	0

To experiment, restarted node 4 and then examined LSA database of node 1:

```
 [root@Centos7 src]# /usr/local/bin/ospfd_mon -p 48849
Connecting to Centos7:48849 ...connected

OSPF Router ID:10.0.0.1          # AS-external-LSAs:      0
ASE checksum:  0x0               # ASEs originated:       0
ASEs allowed:  0                 # Dijkstras:             9
# Areas:       1                 # Nbrs in Exchange:      0
MOSPF enabled:          yes              Inter-area multicast:     yes
Inter-AS multicast: no           In overflow state:        no
ospfd version:  2.16

10.0.0.1> database 0
Type       LS_ID       ADV_RTR     Seqno   Xsum  Age
  1      10.0.0.1      10.0.0.1 0x80000003 0x8157   21
  1      10.0.0.2      10.0.0.2 0x80000004 0xb83c   25
  1      10.0.0.3      10.0.0.3 0x80000004 0xec02   25
  1      10.0.0.4      10.0.0.4 0x80000003 0xeab6 1201
                  # LSAs: 4
                  Database xsum: 0x3104b
10.0.0.1> adv 1 10.0.0.2 10.0.0.2 0
          LS age:          55
          LS Options:      0x26
          LS Type:         1
          Link State ID:   10.0.0.2
          Advert. Rtr.:    10.0.0.2
          LS Seqno:        0x80000004
          LS Xsum:         0xb83c
          LS Length:       48
                  // Router-LSA body
                  Router type:    0x0
                  # links:  2
                  // Link #0
                  Link ID: 10.0.0.1
                  Link Data:      0.0.0.2
                  Link type:      1
                  # TOS metrics:  0
                  Link cost:      1
                  // Link #1
                  Link ID: 1.0.0.2
                  Link Data:      255.255.255.255
                  Link type:      3
                  # TOS metrics:  0
                  Link cost:      0
10.0.0.1>
```

Few observations: Except, LSA id 10.0.0.4, all other LSA ids sequence no has restarted and age is fresh. With node 4 going down, all other router LSAs changed and hence the observed behavior. Route-LSA ID 10.0.0.2 has only 2 entries now instead of 3.

Later on, node 4 came back. Then LSA database on node 1 is as follows:

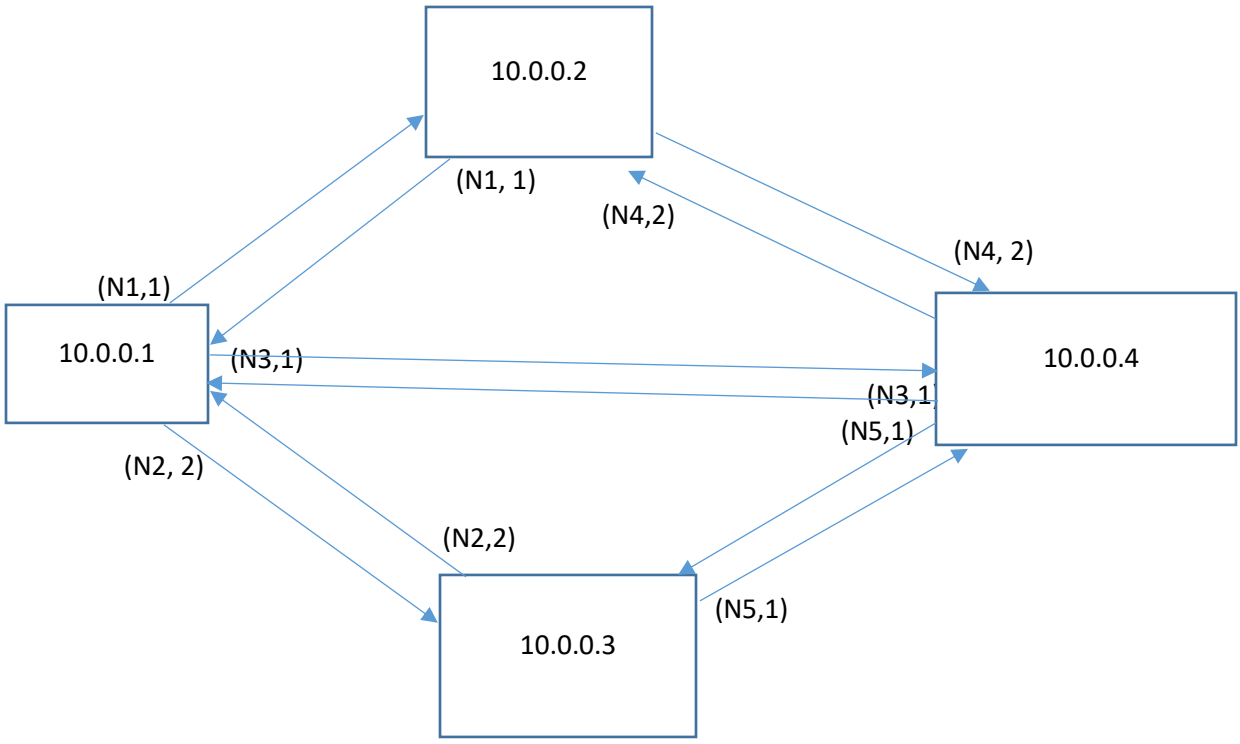
```
[root@Centos7 src]# /usr/local/bin/ospfd_mon -p 48849
Connecting to Centos7:48849 ...connected

OSPF Router ID:10.0.0.1          # AS-external-LSAs:      0
ASE checksum:  0x0               # ASEs originated:       0
ASEs allowed:  0                 # Dijkstras:             13
# Areas:       1                 # Nbrs in Exchange:      0
MOSPF enabled:         yes              Inter-area multicast:     yes
Inter-AS multicast: no           In overflow state:     no
ospfd version:  2.16

10.0.0.1> database 0
Type       LS_ID       ADV_RTR     Seqno   Xsum  Age
  1      10.0.0.1      10.0.0.1 0x80000004 0x496c  13
  1      10.0.0.2      10.0.0.2 0x80000005 0xd5f8  13
  1      10.0.0.3      10.0.0.3 0x80000005 0x4483  16
  1      10.0.0.4      10.0.0.4 0x80000005 0xe6b8  12
                  # LSAs: 4
                  Database xsum: 0x24a9f
10.0.0.1> adv 1 10.0.0.2 10.0.0.2 0
          LS age:          45
          LS Options:      0x26
          LS Type:         1
          Link State ID:   10.0.0.2
          Advert. Rtr.:    10.0.0.2
          LS Seqno:        0x80000005
          LS Xsum:         0xd5f8
          LS Length:       60
                  // Router-LSA body
                  Router type:     0x0
                  # links:  3
                  // Link #0
                  Link ID: 10.0.0.4
                  Link Data:       0.0.0.7
                  Link type:       1
                  # TOS metrics:  0
                  Link cost:       2
                  // Link #1
                  Link ID: 10.0.0.1
                  Link Data:       0.0.0.2
```

Link type: 1
TOS metrics: 0
Link cost: 1
// Link #2
Link ID: 1.0.0.2
Link Data: 255.255.255.255
Link type: 3
TOS metrics: 0
Link cost: 0
10.0.0.1>

For all LSA Ids, sequence no has again incremented and age is relatively less. Route-LSA ID 10.0.0.2 has now 3 entries since 10.0.0.4 is reachable now.

Path Cost and Routing Table Calculation (SPF):

Based on LSA database, an implied directed graph can be made on each node. For the above mentioned example, directed graph will be as follows:



From the above graph, if pkt has to go from 10.0.0.1 to 10.0.0.4, then three path available: direct path via N3 with cost 1, indirect path via N1 and N4 with cost 3 and indirect path via N2 and N5 with cost 3. Path having least cost will be via N3 with cost 1.

Shortest path algorithm(for example Dijkstra algorithm) is applied on this graph to find least cost path between current node and various other point to point node or broadcast network.

For L3 router, pruned graph is made between current node and all other point to point node or broadcast network. Based on pruned graphs, radix tree or any other prefix tree is constructed which associate each interface with probable network addresses. This is called FIB (Forward information base) which is pushed to data plane and incoming packets are switched accordingly through the correct outgoing interface.

For L1 router, there is no switching plane. Given a set of two end points and constraints, least cost path is calculated from directed graph. Now based on this path, any signaling protocol (like RSVP) is used to go to each node on the path and create required software/hardware XCON on that.

Routing table calculation done by open OSPF stack on node 10.0.0.1 is as follows:

```
10.0.0.1> routes
Prefix          Type    Cost    Ifc     Next-hop        Mpaths
1.0.0.1/32      SPF     0               n/a
1.0.0.2/32      SPF     1       N1      0.0.0.2
1.0.0.3/32      SPF     2       N2      0.0.0.4         2
1.0.0.4/32      SPF     1       N3      0.0.0.6
10.0.0.1>
```

Summary LSA generation :

In the sample network shown above, there are two areas, 0 and 1. We saw router LSAs generated and exchanged within area 0 and final router LSA database for area 0. Similarly, router LSAs are generated and exchanged within area 1. Let's go to node 10.0.0.2 which is area border router between area 0 and area 1 and dump LSA databases on that as follows:

```
[root@Centos7 src]# /usr/local/bin/ospfd_mon -p 37426
Connecting to Centos7:37426 ...connected

OSPF Router ID:10.0.0.2         # AS-external-LSAs:     0
ASE checksum:  0x0              # ASEs originated:      0
ASEs allowed:  0                # Dijkstras:            8
# Areas:       2                # Nbrs in Exchange:     0
MOSPF enabled:          yes             Inter-area multicast:   yes
Inter-AS multicast: no          In overflow state:      no
```

```
ospfd version:   2.16

10.0.0.2> area
Area ID       #Ifcs   #Routers #LSAs  Xsum      Comments
0.0.0.0       2       4       8      0x4d092    demand-capable
0.0.0.1       1       4       12     0x78bb4    demand-capable
10.0.0.2> interfaces
Phy           Addr          Area          Type    State    #Nbr #Adj Cost
N1            0.0.0.0       0.0.0.0       P-P       1    1    1
N4            0.0.0.0       0.0.0.0       P-P       1    1    2
N6            0.0.0.0       0.0.0.1       P-P       1    1    3
10.0.0.2> neighbor
Phy           Addr          ID            State    #DD  #Rq  #Rxmt
N1            1.0.0.1       10.0.0.1       Full    0    0    0
N4            1.0.0.4       10.0.0.4       Full    0    0    0
N6            1.0.0.10      10.0.0.10      Full    0    0    0
10.0.0.2> database 0
Type       LS_ID       ADV_RTR     Seqno   Xsum  Age
  1      10.0.0.1      10.0.0.1 0x80000002 0x4d6a  147
  1      10.0.0.2      10.0.0.2 0x80000002 0xdef1  146
  1      10.0.0.3      10.0.0.3 0x80000002 0x4d7c  149
  1      10.0.0.4      10.0.0.4 0x80000002 0xecb5  148
  3      1.0.0.10      10.0.0.2 0x80000001 0x938b  146
  3      1.0.0.10      10.0.0.3 0x80000001 0xab6f  144
  3      1.0.0.11      10.0.0.2 0x80000001 0xa773  142
  3      1.0.0.11      10.0.0.3 0x80000001 0x8399  148
                    # LSAs: 8
                    Database xsum: 0x4d092
10.0.0.2> database 1
Type       LS_ID       ADV_RTR     Seqno   Xsum  Age
  1      10.0.0.2      10.0.0.2 0x80000001 0x36b6  163
  1      10.0.0.3      10.0.0.3 0x80000001 0x687d DNA+3
  1     10.0.0.10      10.0.0.10 0x80000002 0x5649  160
  1     10.0.0.11      10.0.0.11 0x80000002 0x3364 DNA+2
  3      1.0.0.1       10.0.0.2 0x80000001 0xd950  161
  3      1.0.0.1       10.0.0.3 0x80000001 0xdd4a DNA+3
  3      1.0.0.2       10.0.0.2 0x80000001 0xc564  166
  3      1.0.0.2       10.0.0.3 0x80000001 0xdd48 DNA+3
  3      1.0.0.3       10.0.0.2 0x80000001 0xd94c  162
  3      1.0.0.3       10.0.0.3 0x80000001 0xb572 DNA+5
  3      1.0.0.4       10.0.0.2 0x80000001 0xc560  162
  3      1.0.0.4       10.0.0.3 0x80000001 0xb570 DNA+3
                    # LSAs: 12
                    Database xsum: 0x78bb4
10.0.0.2> adv 3 1.0.0.10 10.0.0.2 0
          LS age:             219
          LS Options:         0x26
          LS Type:            3
```

```
        Link State ID:    1.0.0.10
        Advert. Rtr.:     10.0.0.2
        LS Seqno:         0x80000001
        LS Xsum:          0x938b
        LS Length:        28
                // Summary-LSA body
                Network Mask: 255.255.255.255
                Cost:             3
10.0.0.2> adv 3 1.0.0.10 10.0.0.2 0
        LS age:           227
        LS Options:       0x26
        LS Type:          3
        Link State ID:    1.0.0.10
        Advert. Rtr.:     10.0.0.2
        LS Seqno:         0x80000001
        LS Xsum:          0x938b
        LS Length:        28
                // Summary-LSA body
                Network Mask: 255.255.255.255
                Cost:             3
10.0.0.2> routes
Prefix          Type    Cost    Ifc     Next-hop      Mpaths
1.0.0.1/32      SPF     1       N1      0.0.0.1
1.0.0.2/32      SPF     0               n/a
1.0.0.3/32      SPF     3       N1      0.0.0.1       2
1.0.0.4/32      SPF     2       N1      0.0.0.1       2
1.0.0.10/32     SPF     3       N6      0.0.0.11
1.0.0.11/32     SPF     6       N6      0.0.0.11
10.0.0.2>
```

Node 10.0.0.2 has two areas and in each area, it maintain it's own LSA database. Hence it maintain two sets of directed graph for each area on which it applies SPF separately. Each area will generate summary LSA for each of the point to point node or broadcast network with associated cost and flood it to other area. Summary LSA has type of 3.

In area 0, 4 summary LSAs are flooded for area 1 for node 1.0.0.10 and 1.0.0.11 from each advertising router 10.0.0.2 and 10.0.0.3 respectively with associated cost.

Let's say, a packet has to be sent from node 10.0.0.1 in area 0 to 10.0.0.11 in area 1. From summary LSA in area 0, it knows that there are two path to 10.0.0.11, via 10.0.0.2 with cost 6 and via 10.0.0.3 with cost 3. Now, within area 0, from 10.0.0.1 to 10.0.0.2, cost is 1 and from 10.0.0.1 to 10.0.0.3, cost is 2. Hence total cost via 10.0.0.2 is 7(thru N1) and via 10.0.0.3 is 5(thru N2). Hence route table on 10.0.0.1 will choose N2 to send packet to 10.0.0.11 as evident from following route table:

```
10.0.0.1> route
```

Prefix	Type	Cost	Ifc	Next-hop	Mpaths
1.0.0.1/32	SPF	0		n/a	
1.0.0.2/32	SPF	1	N1	0.0.0.2	
1.0.0.3/32	SPF	2	N2	0.0.0.4	2
1.0.0.4/32	SPF	1	N3	0.0.0.6	
1.0.0.10/32	SPFIA	4	N1	0.0.0.2	
1.0.0.11/32	SPFIA	5	N2	0.0.0.4	2

OSPF Protocol in Detail (Most of the Stuff taken from RFC):

Protocol Data Structures:

Protocol data structure has following elements:

Router ID
A 32-bit number that uniquely identifies this router in the AS. One possible implementation strategy would be to use the smallest IP interface address belonging to the router.

Area structures
Each one of the areas to which the router is connected has its own data structure. This data structure describes the working of the basic OSPF algorithm. Remember that each area runs a separate copy of the basic OSPF algorithm.

Virtual links configured
The virtual links configured with this router as one endpoint. In order to have configured virtual links, the router itself must be an area border router.

List of external routes
These are routes to destinations external to the Autonomous System, that have been gained either through direct experience with another routing protocol (such as BGP), or through configuration information, or through a combination of the two (e.g., dynamic external information to be advertised by OSPF with configured metric).

List of AS-external-LSAs
Part of the link-state database. These have originated from the AS boundary routers.

The routing table
Derived from the link-state database. Each entry in the routing table is indexed by a destination, and contains the destination's cost and a set of paths to use in forwarding packets to the destination.

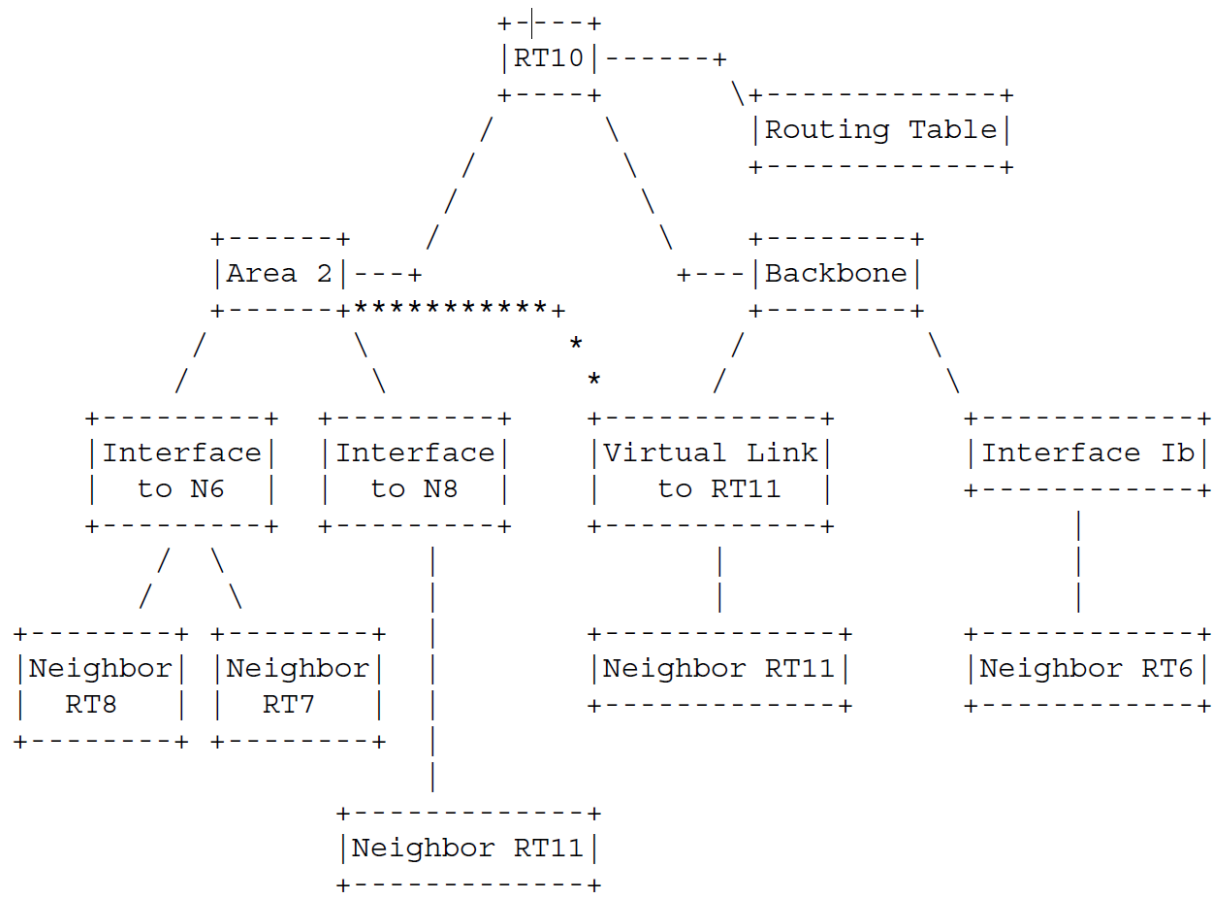


Figure : A Typical Router, say, RT10's Data structures

Area Data Structure:

Area data structure will have following elements:

Area ID
List of area address ranges

Associated router interfaces
This router's interfaces connecting to the area. A router interface belongs to one and only one area (or the backbone).

List of router-LSAs
A router-LSA is generated by each router in the area. It describes the state of the router's interfaces to the area.

List of network-LSAs
One network-LSA is generated for each transit broadcast and NBMA network in the area. A network-LSA describes the set of routers currently connected to the network.

List of summary-LSAs
Summary-LSAs originate from the area's area border routers. They describe routes to destinations internal to the Autonomous System, yet external to the area (i.e., inter-area destinations).

Shortest-path tree
The shortest-path tree for the area, with this router itself as root. Derived from the collected router-LSAs and network-LSAs by the Dijkstra algorithm.

TransitCapability
This parameter indicates whether the area can carry data traffic that neither originates nor terminates in the area itself.

ExternalRoutingCapability
Whether AS-external-LSAs will be flooded into/throughout the area.

StubDefaultCost
If the area has been configured as a stub area, and the router itself is an area border router, then the StubDefaultCost indicates the cost of the default summary-LSA that the router should advertise into the area.

Protocol Packet Processing

This section discusses the general processing of OSPF routing protocol packets. It is very important that the router link-state databases remain synchronized. For this reason, routing protocol packets should get preferential treatment over ordinary data packets, both in sending and receiving.

Sending protocol packets

The IP destination address for the packet is selected as follows. On physical point-to-point networks, the IP destination is always set to the address AllSPFRouters. On all other network types (including virtual links), the majority of OSPF packets are sent as unicasts, i.e., sent directly to the other end of the adjacency. In this case, the IP destination is just the Neighbor IP address associated with the other end of the adjacency. The only packets not sent as unicasts are on broadcast networks; on these networks Hello packets are sent to the multicast destination AllSPFRouters, the Designated Router and its Backup send both Link State Update Packets and Link State Acknowledgment Packets to the multicast address AllSPFRouters, while all other routers send both their Link State Update and Link State Acknowledgment Packets to the multicast address AllDRouters.

Receiving protocol packets

Whenever a protocol packet is received by the router it is marked with the interface it was received on. For routers that have virtual links configured, it may not be immediately obvious which interface to associate the packet with.

If the receiving interface connects to a broadcast network, Point-to-MultiPoint network or NBMA network the sender is identified by the IP source address found in the packet's IP header. If the receiving interface

connects to a point-to-point network or a virtual link, the sender is identified by the Router ID (source router) found in the packet's OSPF header.

Interface Details:

Interface Data Structure:

The following data items are associated with an interface.

Type
State
IP interface address
IP interface mask
Area ID
HelloInterval
RouterDeadInterval
InfTransDelay
Router Priority
Hello Timer
Wait Timer
List of neighboring routers
Designated Router
Backup Designated Router
Interface output cost(s)
RxmtInterval

Interface State Machine:

Interface bringup is guided by it's own interface state machine.

Various Interface states are as follows:

```
enum {
   IFS_DOWN = 0x01,     // Interface is down
   IFS_LOOP = 0x02,     // Interface is looped
   IFS_WAIT = 0x04,     // Waiting to learn Backup DR
   IFS_PP  = 0x08,      // Terminal state for P-P interfaces
   IFS_OTHER = 0x10,    // Mult-access: neither DR nor Backup
   IFS_BACKUP = 0x20,   // Router is Backup on this interface
   IFS_DR  = 0x40,      // Router is DR on this interface

   N_IF_STATES = 7,     /* # OSPF interface states */

   /* Terminal multi-access states */
   IFS_MULTI = (IFS_OTHER | IFS_BACKUP | IFS_DR),
   IFS_ANY = 0x7F,      /* Matches all states */
};
```

Events causing interface state changes are as follows:

```
enum {
   IFE_UP  = 1,   // Interface has come up
   IFE_WTIM,            // Wait timer has fired
   IFE_BSEEN,           // Backup DR has been seen
   IFE_NCHG,            // Associated neighbor has changed state
   IFE_LOOP,            // Interface has been looped
   IFE_UNLOOP,          // Interface has been unlooped
   IFE_DOWN,            // Interface has gone down
   // # OSPF interface events
   N_IF_EVENTS = IFE_DOWN,
};
```

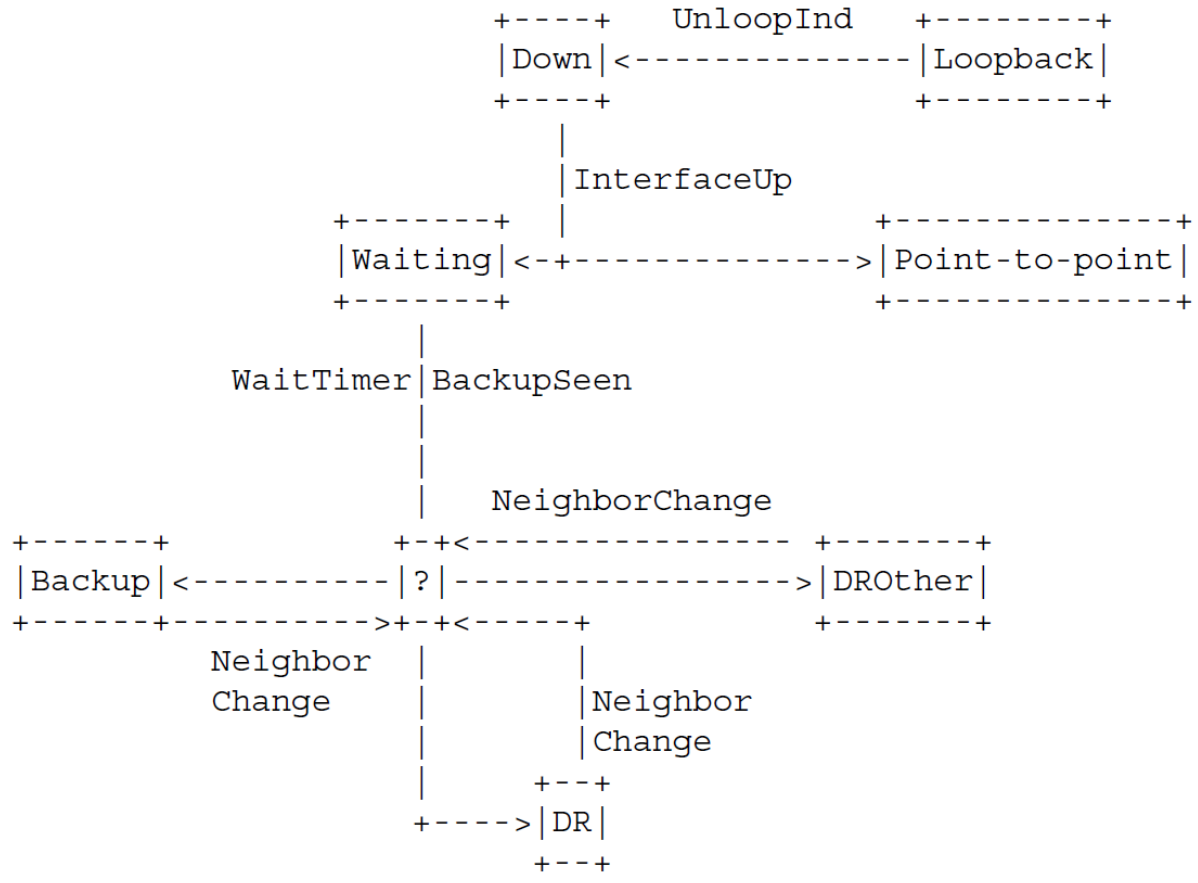


Figure: Interface State Changes

OSPF interface FSM actions are as follows:

```
enum {
   IFA_START = 1,          // Start up the interface
   IFA_ELECT,              // Elect the Designated Router
   IFA_RESET,              // Reset all interface variables
};
```

A generic OSPF transition table can be represented as follows:

```
struct FsmTran {
   int states;             // Set of states to match
   int    event;           // Event to match
   int    action;          // Resulting action to take
   int    new_state;       // New state
};
```

With above details, complete Interface state machine can be constructed as follows:

```
FsmTran          IfcFsm[] = {
   { IFS_DOWN,  IFE_UP,          IFA_START,      0},
   { IFS_WAIT,  IFE_BSEEN,       IFA_ELECT,      0},
   { IFS_WAIT,  IFE_WTIM,        IFA_ELECT,      0},
   { IFS_WAIT,  IFE_NCHG,        0,              0},
   { IFS_MULTI, IFE_NCHG,        IFA_ELECT,      0},
   { IFS_ANY,   IFE_NCHG,        0,              0},
   { IFS_ANY,   IFE_DOWN,        IFA_RESET,      IFS_DOWN},
   { IFS_ANY,   IFE_LOOP,        IFA_RESET,      IFS_LOOP},
   { IFS_LOOP,  IFE_UNLOOP,      0,              IFS_DOWN},
   { 0,   0,                  -1,             0},
};
```

The same state machine has been depicted graphically above.

Neighbor Details:

The Neighbor Data Structure:

The following data items are associated with a neighbor.

State
Inactivity Timer
Master/Slave
DD Sequence Number
Last received Database Description packet

Neighbor ID
Neighbor Priority
Neighbor IP address
Neighbor Options
Neighbor's Designated Router
Neighbor's Backup Designated Router
Link state retransmission list
Database summary list
Link state request list

Neighbor State Machine:

Various neighbor states are as follows:

```
enum {
   NBS_DOWN = 0x01,   // Neighbor down
   NBS_ATTEMPT = 0x02,          // Attempt to send hellos (NBMA)
   NBS_INIT = 0x04,       // 1-Way communication
   NBS_2WAY = 0x08,     // 2-Way communication
   NBS_EXST = 0x10,       // Negotiate Master/Slave
   NBS_EXCH = 0x20,       // Start sending DD packets
   NBS_LOAD = 0x40,       // DDs done, now only LS reqs
   NBS_FULL = 0x80,       // Full adjacency

   NBS_ACTIVE = 0xFC,   // Any state but down and attempt
   NBS_FLOOD = NBS_EXCH | NBS_LOAD | NBS_FULL,
   NBS_ADJFORM = NBS_EXST | NBS_FLOOD,
   NBS_BIDIR = NBS_2WAY | NBS_ADJFORM,
   NBS_PRELIM = NBS_DOWN | NBS_ATTEMPT | NBS_INIT,
   NBS_ANY = 0xFF,        // All states
};
```

Events causing neighbor state changes are as follows:

```
enum {
   NBE_HELLO = 1,         // Hello Received
   NBE_START,             // Start sending hellos
   NBE_2WAY,              // Bidirectional indication
   NBE_NEGDONE,           // Negotiation of master/slave, seq #
   NBE_EXCHDONE,          // All DD packets exchanged
   NBE_BADLSREQ,          // Bad LS request received
   NBE_LDONE,             // Loading Done
   NBE_EVAL,              // Evaluate whether should be adjacent
   NBE_DDRCVD,                    // Received valide DD packet
   NBE_DDSEQNO,           // Bad sequence number in DD process
   NBE_1WAY,              // Only 1-way communication
   NBE_DESTROY,           // Destroy the neighbor
```

```
   NBE_INACTIVE,        // No hellos seen recently
   NBE_LLDOWN,                  // Down indication from link level
   NBE_ADJTMO,                  // Adjacency forming timeout
};
```

Neighbor FSM actions are as follows:

```
enum {
   NBA_START   = 1,     // Start up the neighbor
   NBA_RST_IATIM,       // Reset inactivity timer
   NBA_ST_IATIM,        // Start inactivity timer
   NBA_EVAL1,           // Start DD process? (Internal request)
   NBA_EVAL2,           // Start DD process? (External request)
   NBA_SNAPSHOT,        // Take snapshot of LSDB, and start sending
   NBA_EXCHDONE,        // Actions associated with exchange done
   NBA_REEVAL,          // Reevaluate whether nbr should be adj.
   NBA_RESTART_DD,      // Restart DD process
   NBA_DELETE,          // Delete the neighbor
   NBA_CLR_LISTS,       // Clear database lists
   NBA_HELLOCHK,        // Start, stop NBMA hellos
};
```

A generic OSPF transition table can be represented as follows:

```
struct FsmTran {
   int states;          // Set of states to match
   int     event;       // Event to match
   int     action;      // Resulting action to take
   int     new_state;   // New state
};
```

With this, the complete neighbor state machine can be constructed as follows:

```
FsmTran NbrFsm[] = {
   { NBS_ACTIVE,        NBE_HELLO,     NBA_RST_IATIM,        0},
   { NBS_BIDIR,         NBE_2WAY,      0,                    0},
   { NBS_INIT,          NBE_1WAY,      0,                    0},
   { NBS_DOWN,          NBE_HELLO,     NBA_ST_IATIM,         NBS_INIT},
   { NBS_DOWN,          NBE_START,     NBA_START,            NBS_ATTEMPT},
   { NBS_ATTEMPT,       NBE_HELLO,     NBA_RST_IATIM,        NBS_INIT},
   { NBS_INIT,          NBE_2WAY,      NBA_EVAL1,            0},
   { NBS_INIT,          NBE_DDRCVD,    NBA_EVAL2,            0},
   { NBS_2WAY,          NBE_DDRCVD,    NBA_EVAL2,            0},
   { NBS_EXST,          NBE_NEGDONE,NBA_SNAPSHOT,            NBS_EXCH},
   { NBS_EXCH,          NBE_EXCHDONE,NBA_EXCHDONE,           0},
   { NBS_LOAD,          NBE_LDONE,     0,                    NBS_FULL},
```

```
  { NBS_2WAY,          NBE_EVAL,       NBA_EVAL1,          0},
  { NBS_ADJFORM,       NBE_EVAL,       NBA_REEVAL,         0},
  { NBS_PRELIM,        NBE_EVAL,       NBA_HELLOCHK,       0},
  { NBS_ADJFORM,       NBE_ADJTMO,     NBA_RESTART_DD,     NBS_2WAY},
  { NBS_FLOOD,         NBE_DDSEQNO,NBA_RESTART_DD,         NBS_2WAY},
  { NBS_FLOOD,         NBE_BADLSREQ,NBA_RESTART_DD,        NBS_2WAY},
  { NBS_ANY,           NBE_DESTROY,    NBA_DELETE,         NBS_DOWN},
  { NBS_ANY,           NBE_LLDOWN,     NBA_DELETE,         NBS_DOWN},
  { NBS_ANY,           NBE_INACTIVE,   NBA_DELETE,         NBS_DOWN},
  { NBS_BIDIR,         NBE_1WAY,       NBA_CLR_LISTS,      NBS_INIT},
  { 0,    0,           -1,             0},
};
```

OSPF neighbor state machine has two phases. In the first phase, it ensures that communication over a link becomes bidirectional and capabilities and properties matched using Hello Protocol mechanism so that link can participate in OSPF control traffic. In the second phase, it ensures that neighbors become fully adjacent using DD exchange process. Both aspect of neighbor state machine has been depicted graphically below.

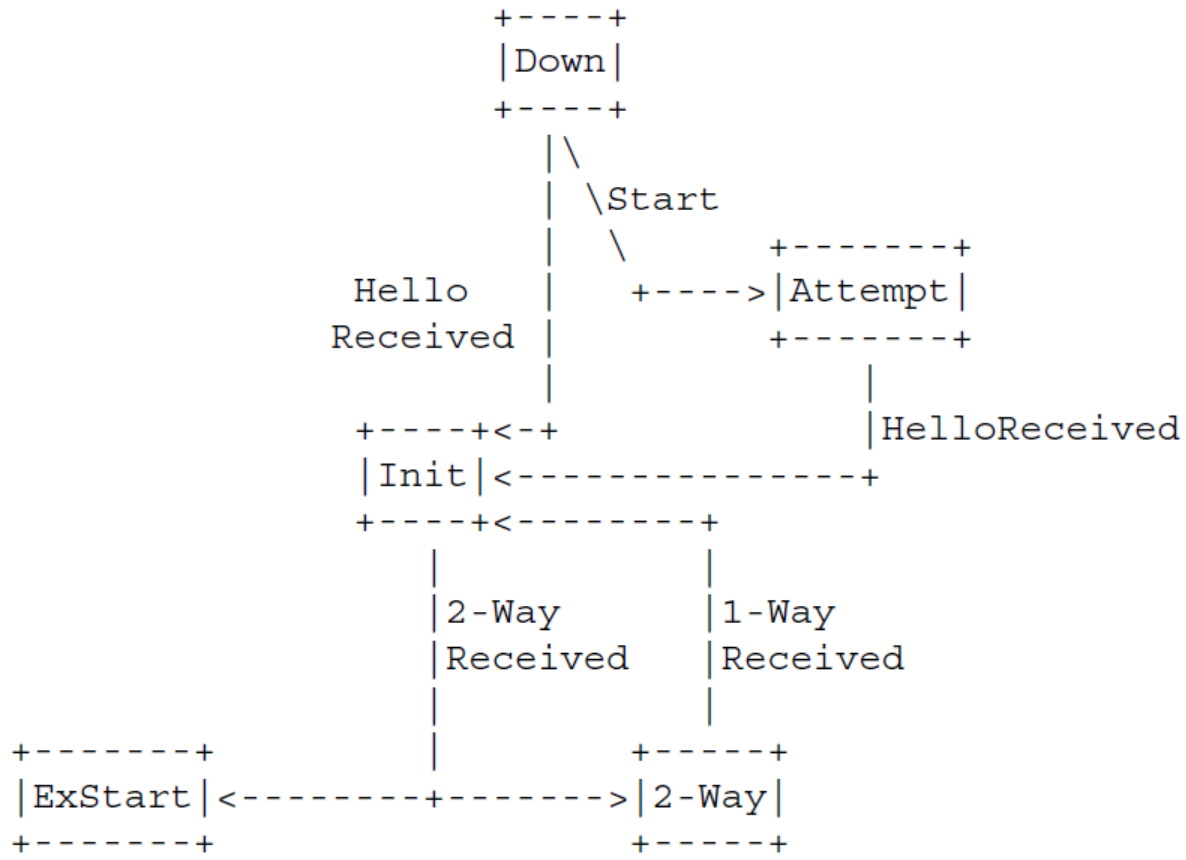


Figure: Neighbor state changes (Hello Protocol)

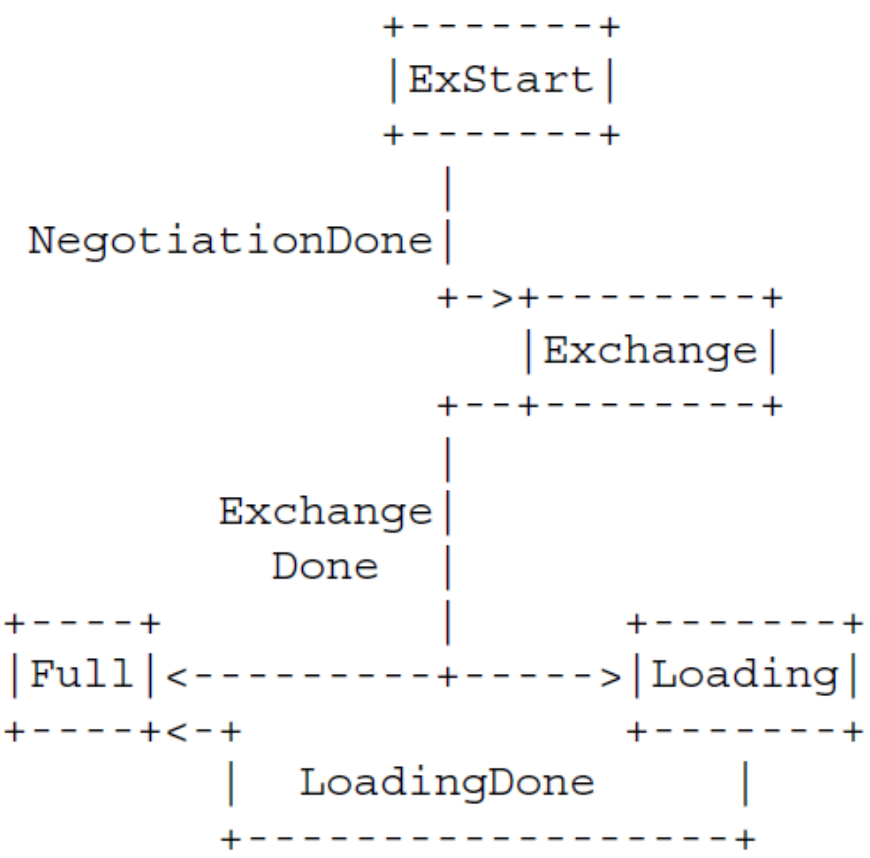


Figure: Neighbor state changes (Database Exchange)

The Hello Protocol:

The Hello Protocol is responsible for establishing and maintaining neighbor relationships. It also ensures that communication between neighbors is bidirectional. Hello packets are sent periodically out all router interfaces. Bidirectional communication is indicated when the router sees itself listed in the neighbor's Hello Packet. On broadcast and NBMA networks, the Hello Protocol elects a Designated Router for the network. Once bidirection is established, capabilities and properties exchanged using Hello protocol, that link can carry ospf control traffic.

Sending Hello packets

Hello packets are sent out each functioning router interface. They are used to discover and maintain neighbor relationships. On broadcast and NBMA networks, Hello Packets are also used to elect the Designated Router and Backup Designated Router.

The Hello Packet contains the router's Router Priority (used in choosing the Designated Router), and the interval between Hello Packets sent out the interface (HelloInterval). The Hello Packet also indicates how

often a neighbor must be heard from to remain active (RouterDeadInterval). Both HelloInterval and RouterDeadInterval must be the same for all routers attached to a common network. The Hello packet also contains the IP address mask of the attached network (Network Mask). On unnumbered point-to-point networks and on virtual links this field should be set to 0.0.0.0.

The Hello packet's Options field describes the router's optional OSPF capabilities. The E-bit of the Options field should be set if and only if the attached area is capable of processing AS-external-LSAs (i.e., it is not a stub area). If the E-bit is set incorrectly the neighboring routers will refuse to accept the Hello Packet. Unrecognized bits in the Hello Packet's Options field should be set to zero.

In order to ensure two-way communication between adjacent routers, the Hello packet contains the list of all routers on the network from which Hello Packets have been seen recently. The Hello packet also contains the router's current choice for Designated Router and Backup Designated Router. A value of 0.0.0.0 in these fields means that one has not yet been selected. On broadcast networks and physical point-to-point networks, Hello packets are sent every HelloInterval seconds to the IP multicast address AllSPFRouters. On virtual links, Hello packets are sent as unicasts (addressed directly to the other end of the virtual link) every HelloInterval seconds. On Point-to-MultiPoint networks, separate Hello packets are sent to each attached neighbor every HelloInterval seconds.

Receiving Hello Packets

The generic input processing of OSPF packets will have checked the validity of the IP header and the OSPF packet header. Next, the values of the Network Mask, HelloInterval, and RouterDeadInterval fields in the received Hello packet must be checked against the values configured for the receiving interface. Any mismatch causes processing to stop and the packet to be dropped. In other words, the above fields are really describing the attached network's configuration. However, there is one exception to the above rule: on point-to-point networks and on virtual links, the Network Mask in the received Hello Packet should be ignored.

The receiving interface attaches to a single OSPF area (this could be the backbone). The setting of the E-bit found in the Hello Packet's Options field must match this area's ExternalRoutingCapability. If AS-external-LSAs are not flooded into/throughout the area (i.e, the area is a "stub") the E-bit must be clear in received Hello Packets, otherwise the E-bit must be set. A mismatch causes processing to stop and the packet to be dropped. The setting of the rest of the bits in the Hello Packet's Options field should be ignored. At this point, an attempt is made to match the source of the Hello Packet to one of the receiving interface's neighbors. If the receiving interface connects to a broadcast, Point-to-MultiPoint or NBMA network the source is identified by the IP source address found in the Hello's IP header. If the receiving interface connects to a point-to-point link or a virtual link, the source is identified by the Router ID found in the Hello's OSPF packet header. The interface's current list of neighbors is contained in the interface's data structure. If a matching neighbor structure cannot be found, (i.e., this is the first time the neighbor has been detected), one is created. The initial state of a newly created neighbor is set to Down.

When receiving an Hello Packet from a neighbor on a broadcast, Point-to-MultiPoint or NBMA network, set the neighbor structure's Neighbor ID equal to the Router ID found in the packet's OSPF header. For these network types, the neighbor structure's Router Priority field, Neighbor's Designated Router field,

and Neighbor's Backup Designated Router field are also set equal to the corresponding fields found in the received Hello Packet; changes in these fields should be noted for possible use in the steps below. When receiving an Hello on a point-to-point network (but not on a virtual link) set the neighbor structure's Neighbor IP address to the packet's IP source address. Now the rest of the Hello Packet is examined, generating events to be given to the neighbor and interface state machines. These state machines are specified either to be executed or scheduled. For example, by specifying below that the neighbor state machine be executed in line, several neighbor state transitions may be effected by a single received Hello.

The Synchronization of Databases:

In a link-state routing algorithm, it is very important for all routers' link-state databases to stay synchronized. OSPF simplifies this by requiring only adjacent routers to remain synchronized. The synchronization process begins as soon as the routers attempt to bring up the adjacency. Each router describes its database by sending a sequence of Database Description packets to its neighbor. Each Database Description Packet describes a set of LSAs belonging to the router's database. When the neighbor sees an LSA that is more recent than its own database copy, it makes a note that this newer LSA should be requested. When two routers are fully adjacent, then only these are advertised in the two routers' router-LSAs.

Receiving Database Description Packets

This section explains the detailed processing of a received Database Description Packet. The incoming Database Description Packet has already been associated with a neighbor and receiving interface by the generic input packet processing. Whether the Database Description packet should be accepted, and if so, how it should be further processed depends upon the neighbor state. If a Database Description packet is accepted, the following packet fields should be saved in the corresponding neighbor data structure under "last received Database Description packet": the packet's initialize(I), more (M) and master(MS) bits, Options field, and DD sequence number. If these fields are set identically in two consecutive Database Description packets received from the neighbor, the second Database Description packet is considered to be a "duplicate" in the processing described below. If the Interface MTU field in the Database Description packet indicates an IP datagram size that is larger than the router can accept on the receiving interface without fragmentation, the Database Description packet is rejected.
Otherwise, if the neighbor state is:

Down
The packet should be rejected.

Attempt
The packet should be rejected.

Attempt
The packet should be rejected.

Init

The neighbor state machine should be executed with the event 2-WayReceived. This causes an immediate state change to either state 2-Way or state ExStart. If the new state is ExStart, the processing of the current packet should then continue in this new state by falling through to case ExStart below.

2-Way
The packet should be ignored.

ExStart
If the received packet matches one of the following cases, then the neighbor state machine should be executed with the event NegotiationDone (causing the state to transition to Exchange), the packet's Options field should be recorded in the neighbor structure's Neighbor Options field and the packet should be accepted as next in sequence and processed. Otherwise, the packet should be ignored.

Exchange
Duplicate Database Description packets are discarded by the master, and cause the slave to retransmit the last Database Description packet that it had sent.

Loading or Full
In this state, the router has sent and received an entire sequence of Database Description Packets. The only packets received should be duplicates.

Receiving Link State Request Packets

This section explains the detailed processing of received Link State Request packets. Received Link State Request Packets specify a list of LSAs that the neighbor wishes to receive. Link State Request Packets should be accepted when the neighbor is in states Exchange, Loading, or Full. In all other states Link State Request Packets should be ignored.

Sending Database Description Packets

This section describes how Database Description Packets are sent to a neighbor. The Database Description packet's Interface MTU field is set to the size of the largest IP datagram that can be sent out the sending interface, without fragmentation. Interface MTU should be set to 0 in Database Description packets sent over virtual links. The router's optional OSPF capabilities are transmitted to the neighbor in the Options field of the Database Description packet. The router should maintain the same set of optional capabilities throughout the Database Exchange and flooding procedures.

The sending of Database Description packets depends on the neighbor's state. In state ExStart the router sends empty Database Description packets, with the initialize (I), more (M) and master (MS) bits set. These packets are retransmitted every RxmtInterval seconds.

In state Exchange the Database Description Packets actually contain summaries of the link state information contained in the router's database. Each LSA in the area's link-state database (at the time the neighbor transitions into Exchange state) is listed in the neighbor Database summary list. Each new Database Description Packet copies its DD sequence number from the neighbor data structure and then

describes the current top of the Database summary list. Items are removed from the Database summary list when the previous packet is acknowledged.

Sending Link State Request Packets

In neighbor states Exchange or Loading, the Link state request list contains a list of those LSAs that need to be obtained from the neighbor. To request these LSAs, a router sends the neighbor the beginning of the Link state request list, packaged in a Link State Request packet. When the neighbor responds to these requests with the proper Link State Update packet(s), the Link state request list is truncated and a new Link State Request packet is sent.

When the Link state request list becomes empty, and the neighbor state is Loading (i.e., a complete sequence of Database Description packets has been sent to and received from the neighbor), the Loading Done neighbor event is generated.

An example of an adjacency forming is as follows:

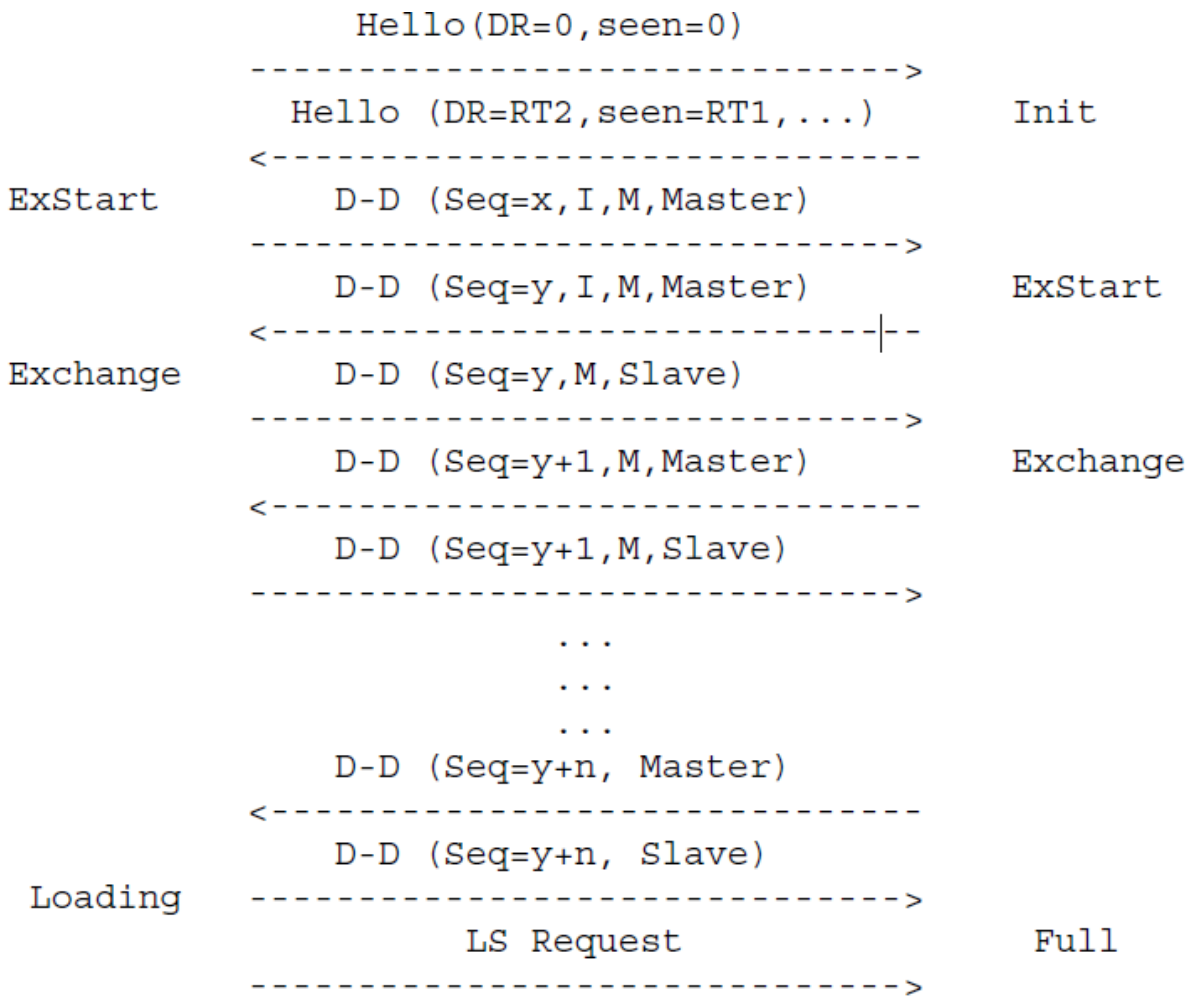


Figure: An adjacency bring-up example

Routing Table Structure:

The following data items are associated with a routing table:

Destination Type
Destination ID
Address Mask
Optional Capabilities
Area
Cost
Type 2 cost
Link State Origin
Next hop
Advertising router

A sample network (taken from RFC) is as follows:

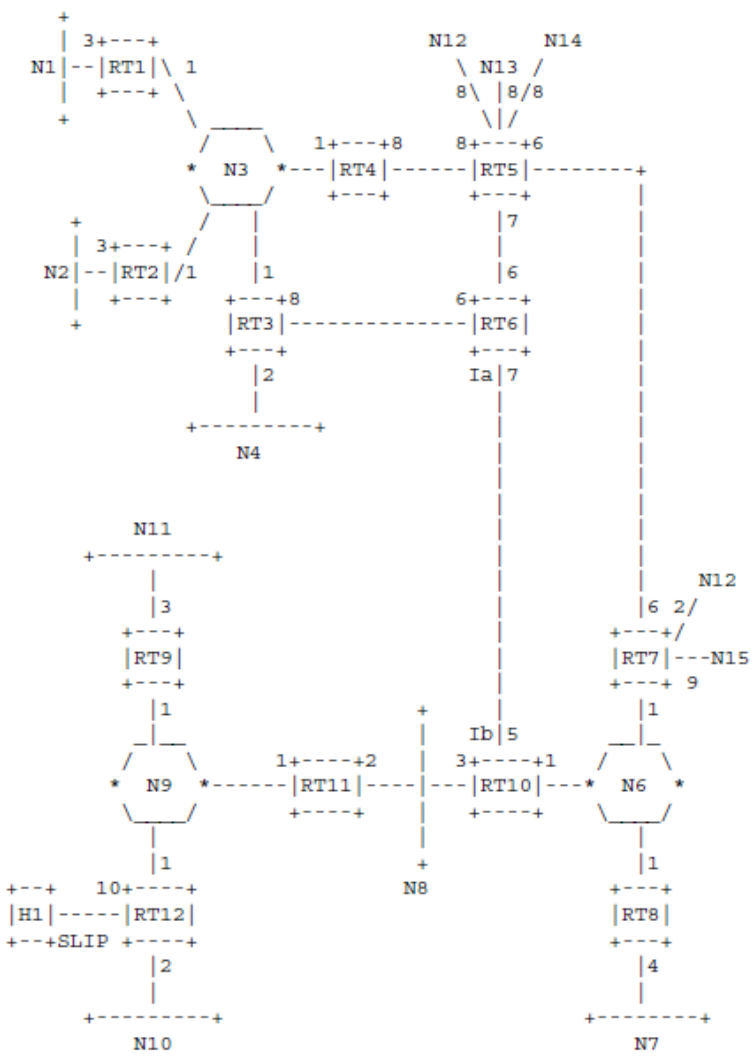


Figure: Sample Autonous System With Multiple Areas

If routing table is generated for router RT6 in above network, it will be as follows:

Type	Dest	Area	Path Type	Cost	Next Hop(s)	Adv. Router(s)
N	N1	0	intra-area	10	RT3	*
N	N2	0	intra-area	10	RT3	*
N	N3	0	intra-area	7	RT3	*
N	N4	0	intra-area	8	RT3	*
N	Ib	0	intra-area	7	*	*
N	Ia	0	intra-area	12	RT10	*
N	N6	0	intra-area	8	RT10	*
N	N7	0	intra-area	12	RT10	*
N	N8	0	intra-area	10	RT10	*
N	N9	0	intra-area	11	RT10	*
N	N10	0	intra-area	13	RT10	*
N	N11	0	intra-area	14	RT10	*
N	H1	0	intra-area	21	RT10	*
R	RT5	0	intra-area	6	RT5	*
R	RT7	0	intra-area	8	RT10	*
N	N12	*	type 1 ext.	10	RT10	RT7
N	N13	*	type 1 ext.	14	RT5	RT5
N	N14	*	type 1 ext.	14	RT5	RT5
N	N15	*	type 1 ext.	17	RT10	RT7

Figure: The routing table for Router RT6 mentioned above

Originating LSAs:

Into any given OSPF area, a router will originate several LSAs. Each router originates a router-LSA. If the router is also the Designated Router for any of the area's networks, it will originate network-LSAs for those networks. Area border routers originate a single summary-LSA for each known inter-area destination. AS boundary routers originate a single AS-external-LSA for each known AS external destination. Destinations are advertised one at a time so that the change in any single route can be flooded without reflooding the entire collection of routes. During the flooding procedure, many LSAs can be carried by a single Link State Update packet.

As an example, consider Router RT4 in above figure. It is an area border router, having a connection to Area 1 and the backbone. Router RT4 originates 5 distinct LSAs into the backbone (one router-LSA, and one summary-LSA for each of the networks N1-N4). Router RT4 will also originate 8 distinct LSAs into Area 1 (one router-LSA and seven summary-LSAs). If RT4 has been selected as Designated Router for Network N3, it will also originate a network-LSA for N3 into Area 1.

Whenever a new instance of an LSA is originated, its LS sequence number is incremented, its LS age is set to 0, its LS checksum is calculated, and the LSA is added to the link state database and flooded out the appropriate interfaces.

The ten events that can cause a new instance of an LSA to be originated are:

(1) The LS age field of one of the router's self-originated LSAs reaches the value LSRefreshTime. In this case, a new instance of the LSA is originated, even though the contents of the LSA (apart from the LSA header) will be the same.

(2) An interface's state changes. This may mean that it is necessary to produce a new instance of the router-LSA.

(3) An attached network's Designated Router changes.

(4) One of the neighboring routers changes to/from the FULL state.

The next four events concern area border routers only:

(5) An intra-area route has been added/deleted/modified in the routing table. This may cause a new instance of a summary-LSA (for this route) to be originated in each attached area (possibly including the backbone).

(6) An inter-area route has been added/deleted/modified in the routing table. This may cause a new instance of a summary-LSA (for this route) to be originated in each attached area (but NEVER for the backbone).

(7) The router becomes newly attached to an area. The router must then originate summary-LSAs into the newly attached area for all pertinent intra-area and inter-area routes in the router's routing table.

(8) When the state of one of the router's configured virtual links changes, it may be necessary to originate a new router-LSA into the virtual link's Transit area.

The last two events concern AS boundary routers (and former AS boundary routers) only:

(9) An external route gained through direct experience with an external routing protocol (like BGP) changes. This will cause an AS boundary router to originate a new instance of an AS-external-LSA.

(10) A router ceases to be an AS boundary router, perhaps after restarting. In this situation the router should flush all AS-external-LSAs that it had previously originated.

The Flooding Procedure:

Link State Update packets provide the mechanism for flooding LSAs. A Link State Update packet may contain several distinct LSAs, and floods each LSA one hop further from its point of origination. To make the flooding procedure reliable, each LSA must be acknowledged separately. Acknowledgments are

transmitted in Link State Acknowledgment packets.

Many separate acknowledgments can also be grouped together into a single packet.

Determining which LSA is newer
When a router encounters two instances of an LSA, it must determine which is more recent. For two instances of the same LSA, the LS sequence number, LS age, and LS checksum fields are used to determine which instance is more recent. Installing LSAs in the database Installing a new LSA in the database, either as the result of flooding or a newly self-originated LSA, may cause the OSPF routing table structure to be recalculated. The contents of the new LSA should be compared to the old instance, if present. If there is no difference, there is no need to recalculate the routing table. If the contents are different, the following pieces of the routing table must be recalculated, depending on the new LSA's LS type field.

Apart from above, the detailed logic for following can be read from RFC:
Installing LSAs in the database
Next step in the flooding procedure
Receiving self-originated LSAs

Sending Link State Acknowledgment packets

Each newly received LSA must be acknowledged. This is usually done by sending Link State Acknowledgment packets. However, acknowledgments can also be accomplished implicitly
by sending Link State Update packets. Many acknowledgments may be grouped together into a single Link State Acknowledgment packet. Such a packet is sent back out the interface which received the LSAs. The packet can be sent in one of two ways: delayed and sent on an interval timer, or sent directly to a particular neighbor. The particular acknowledgment strategy used depends on the circumstances surrounding the receipt of the LSA.

Retransmitting LSAs
LSAs flooded out an adjacency are placed on the adjacency's Link state retransmission list. In order to ensure that flooding is reliable, these LSAs are retransmitted until they are acknowledged. The length of time between retransmissions is a configurable per-interface value, RxmtInterval.

Aging The Link State Database:

Each LSA has an LS age field. The LS age is expressed in seconds. An LSA's LS age field is incremented while it is contained in a router's database. Also, when copied into a Link State Update Packet for flooding out a particular interface, the LSA's LS age is incremented by InfTransDelay. An LSA's LS age is never incremented past the value MaxAge. LSAs having age MaxAge are not used in the routing table calculation. As a router ages its link state database, an LSA's LS age may reach MaxAge.At this time, the router must attempt to flush the LSA from the routing domain. This is done simply by reflooding the MaxAge LSA just as if it was a newly originated LSA.

Premature aging of LSAs

An LSA can be flushed from the routing domain by setting its LS age to MaxAge, while leaving its LS sequence number alone, and then reflooding the LSA. This procedure follows the same course as flushing an LSA whose LS age has naturally reached the value MaxAge.

Virtual Links:

The single backbone area (Area ID = 0.0.0.0) cannot be disconnected, or some areas of the Autonomous System will become unreachable. To establish/maintain connectivity of the backbone, virtual links can be configured through non-backbone areas. Virtual links serve to connect physically separate components of the backbone. The two endpoints of a virtual link are area border routers. The virtual link must be configured in both routers.

Calculation of the Routing Table:

This process can be broken into the following steps:
(1) The present routing table is invalidated. The routing table is built again from scratch. The old routing table is saved so that changes in routing table entries can be identified.

(2) The intra-area routes are calculated by building the shortestpath tree for each attached area. In particular, all routing table entries whose Destination Type is "area border router" are calculated in this step.

(3) The inter-area routes are calculated, through examination of summary-LSAs. If the router is attached to multiple areas (i.e., it is an area border router), only backbone summary-LSAs are examined.

(4) In area border routers connecting to one or more transit areas (i.e, non-backbone areas whose TransitCapability is found to be TRUE), the transit areas' summary-LSAs are examined to see whether better paths exist using the transit areas than were found in Steps 2-3 above.

(5) Routes to external destinations are calculated, through examination of AS-external-LSAs. The locations of the AS boundary routers (which originate the AS-external-LSAs) have been determined in steps 2-4.

Examples of router-LSAs from above mentioned network:

; RT3's router-LSA for Area 1

LS age = 0 ;always true on origination
Options = (E-bit) ;
LS type = 1 ;indicates router-LSA
Link State ID = 192.1.1.3 ;RT3's Router ID
Advertising Router = 192.1.1.3 ;RT3's Router ID
bit E = 0 ;not an AS boundary router

bit B = 1 ;area border router
#links = 2

Link ID = 192.1.1.4 ;IP address of Desig. Rtr.
Link Data = 192.1.1.3 ;RT3's IP interface to net
Type = 2 ;connects to transit network
# TOS metrics = 0
metric = 1

Link ID = 192.1.4.0 ;IP Network number
Link Data = 0xffffff00 ;Network mask
Type = 3 ;connects to stub network
# TOS metrics = 0
metric = 2

; RT3's router-LSA for the backbone

LS age = 0 ;always true on origination
Options = (E-bit) ;
LS type = 1 ;indicates router-LSA
Link State ID = 192.1.1.3 ;RT3's router ID
Advertising Router = 192.1.1.3 ;RT3's router ID
bit E = 0 ;not an AS boundary router
bit B = 1 ;area border router
#links = 1

Link ID = 18.10.0.6 ;Neighbor's Router ID
Link Data = 0.0.0.3 ;MIB-II ifIndex of P-P link
Type = 1 ;connects to router
# TOS metrics = 0
metric = 8

Examples of network-LSAs from above mentioned network:

A network-LSA is generated for every transit broadcast or NBMA network. (A transit network is a network having two or more attached routers). The network-LSA describes all the routers that are attached to the network.

; Network-LSA for Network N3

LS age = 0 ;always true on origination
Options = (E-bit) ;
LS type = 2 ;indicates network-LSA
Link State ID = 192.1.1.4 ;IP address of Desig. Rtr.
Advertising Router = 192.1.1.4 ;RT4's Router ID
Network Mask = 0xffffff00

Attached Router = 192.1.1.4 ;Router ID
Attached Router = 192.1.1.1 ;Router ID
Attached Router = 192.1.1.2 ;Router ID
Attached Router = 192.1.1.3 ;Router ID

Examples of summary-LSAs from above mentioned network:

; Summary-LSA for Network N1,
; originated by Router RT4 into the backbone
LS age = 0 ;always true on origination
Options = (E-bit) ;
LS type = 3 ;Type 3 summary-LSA
Link State ID = 192.1.2.0 ;N1's IP network number
Advertising Router = 192.1.1.4 ;RT4's ID
metric = 4

OSPF Logical Code Flow of Various Functionalities and Examples from OPEN OSPF STACK

Global Configurations:

A 32 bit router ID is assigned to the router which is generally smallest IP address of one of it's interfaces. If router ID changes, OSPF software should be restarted. At this point, required no of areas are configured within global OSPF domain if any otherwise by default backbone area (which is called area 0) is created. For each area, area data structure is created and populated with specific data holders important being Area ID, List of address ranges, Associated Router Interfaces, LSAs list, Shortest-path tree etc. Some of the data holder may be filled later on.

Then within each area, required no of interfaces are created. For each interface, separate interface data structure is maintained inside specific area data structure. Important data holders of interface data structures are state, interface address, interface mask, Hello timer, list of neighboring routers etc. Interfaces can be of following types: point to point interface, broadcast interface, NBMA interface, point to multipoint interface, etc. List of active interfaces within area is either statically configured by administrator or pulled from kernel. New configured parameters of interfaces are filled. Once assigned ip address and mask, if it is operational, interface state machine is triggered with interface state as UP. Within the Interface State Machine, as said earlier, action IFA_START is triggered which will send HELLO to its neighbors. Also, if the interface is up, it joins multicast address "224.0.0.6" and "224.0.0.5" and router-LSA is recalculated for that routerID.

Hello Packet Flow:

Sending Hello Packets:

1) A particular interface is configured. The moment an interface is configured, hello timer is started for that interface which send hello pkt on that interface periodically to the neighbor.
2) If hello is already received on that interface, one neighbor data structure is created within the interface which has data holders for neighbor properties and it's state will be NBS_INIT. Hello pkt is filled with Associated network's mask, Interval between Hellos, Hello's options field which has 6 bits and can be read from RFC in detail, Sending router's priority, Designated Router (sender's view) and Backup DR (sender's view). If hello has been received from other side, nbr's router id is packed into the hello body. In case of broadcast network, there can be an array of routerids in that network.

3) Hello packet is sent out through that interface. The destination address of that hello pkt will be "224.0.0.5". Kernel will send out the packet from the physical interface corresponding to the ip address of the interface with destination as multicast address. Packet will be broadcasted on that interface and it will go only to till next hop.

Receiving Hello Packets:

1) Hello packet is received on a particular interface. Certain parameters of received hello packet is matched with existing parameters of the interface. If mismatch, packet is discarded. Find the neighbor structure within the interface. Validated and received details from neighbor is filled in neighbor structure. Neighbor state is DOWN. First time, neighbor state machine is triggered with event NBE_HELLO. This will trigger neighbor state machine (NbrFsm) to move to NBS_INIT state. Router dead interval timer is restarted. If Hello is not bidirectional, processing will stop at NBS_INIT state.
2) If bidirectional, NBE_2WAY is generated and NbrFsm moves to action NBA_EVAL1. Now, it will be decided whether current node should attempt adjacency with neighbor node. In a given ospf domain, there is a max limit on locally initiated and remotely initiated adjacencies. If the total adjacencies on the current node has crossed the max limit, put this adjacency into pending adjacency list which will be processed later. Otherwise NbrFsm state moves to NBS_EXST state (Negotiate Master/Slave) and adjacency is started with the neighbor.

   At this point capabilities with neighbor router has been validated and verified using Hello options field and both neighbor knows each other's property and existing link(point-to-point link, point-to-multipoint link, virtual link, NBMA link, broadcast link etc). Also master/slave negotiation has been done.

   Now sending Database Description (DD) pkt to neighbor will start.

DD Packets Flow:

Sending DD Packets:

1) DD packet is sent in following cases: To start adjacency, as a result of receiving DD packet, DD retransmission timer firing. DD packets are filled with all details along with DD options.
2) DD retransmit timer and DD progress timer is started.
3) During beginning of adjacency creation, DD summary list is prepared to be retransmitted again and again till acknowledged.
4) DD packet is filled with the "Database summary list" already prepared. DD packet is sent to the neighbor. If slave and last DD packet sent, NbrFsm moves to NBS_FULL or NBS_LOAD state depending upon whether LS req list is empty or not. If NBS_FULL, exit db exchange and router-LSA for the node is regenerated. Network-LSA is reoriginated if the current node is DR.

Receiving DD Packets:

1) Received DD pkt. Store DD options and Init/more/master/slave fields.
2) If neighbor state is NBS_EXST(Negotiate Master/Slave), NbrFsm moves with action NBE_NEGDONE and database exchange starts. Associated with neighbor, DD retransmit timer stopped and progress timer started.
3) Contents of Database Description packet is processed. Those LSAs that are more recent than database copy of the node are added to neighbor's link state request list.
4) Now it's time to send a link state request packet to the neighbor. Fill packet until either a) the end of the "link state request list" is reached or b) the packet size reached the MTU of the outgoing interface. Then , link state request packet is sent to the neighbor.
5) If no outstanding LS requests, send next DD packet in sequence. Nbr state at this point is NBS_EXCH. If no more DD packets to be exchanged, NbrFsm moves to NBS_FULL/NBS_LOAD. If NBS_LOAD, wait to LS request to complete and then moves it to NBS_FULL.
6) If more DD packets, repeat above steps.

LS Request Flow:

Sending LS Request Packets:

1) If "Link state request list" is empty for the neighbor of the current node, return.
2) Fill in packet from the "Link state request list" till either "link state request list" is reached or the packet size reached the MTU of the outgoing interface.
3) LS Req can be sent when the link with the neighbor is in NBS_EXCH state or NBS_LOAD state.

Receiving LS Request Packets:

1) Received a link state request packet.
2) For each LS request, look up the requested piece in the link state database. If the piece can't be found, NbrFsm triggers with action NBE_BADLSREQ (Bad LS request received). Neighbor adjacency is restarted which means neighbor's data structure is reinitialized in preparation for either redoing the adjacency process or deleting the neighbor. Neighbor state moves back to NBS_2WAY state.
3) If the piece is found, build a network-ready LSA from a parsed database copy.

4) Add an LSA to a pending update packet to be sent to a particular neighbor. Send the Link State Update if the pkt buffer is full.

LS Update Flow:

Sending LS Update Packets:

1) In the context of DD Exchange, sending LS update packet is in response to receiving LS request packet. As said earlier, when network-ready LSA is formed and added to update buffer to neighbor, once buffer is full, update is sent to that neighbor.
2) In the context of flooding, sending update to all neighbors has a different flow and will be detailed in the context of flooding.

Receiving LS Update Packets:

1) Received a link state update packet from a neighbor.
2) Go through all LSAs listed in the update pkt.
3) Find current database copy, if any.
   If not present, install LSA in the database and flood.
   If present, compare this LSA with the database instance LSA.
   If received LSA is more recent than the database, install LSA in the database and flood.
   If received LSA is same as the database, remove LSA from retransmission list of neighbor data structure and ack the neighbor.
   If database copy more recent and reply has not been sent for this LSA, add the content of the LSA to the update buffer of the neighbor to be sent back.
4) At the end, pending update buffer data if any , for flooding, is sent out over all interfaces within an area over multicast address.

LS Ack Flow:

Sending LS Ack Packets:

1) Sending LS Ack packet is in response to receiving LS update from the neighbor.
2) Once LS update has been successful accepted, LS hdr of the LSA is added to pkt buffer of immediate acks to be sent to neighbor.
3) When the ack buffer is full, it is sent to neighbor.
4) Most of the time, if possible, LS acks are piggybacked with LS updates.

Receiving LS Ack Packets:

1) Received a link state acknowledgment packet.
2) For each LSA listed, check is made to see whether a matching LSA (same instance) exists on the neighbor's link state retransmission list or not.
3) If it does, remove it from the retransmission list.

4) Otherwise log an error for NEWER_ACK or OLD_ACK and continue.

Finally, at this point NbrFsm is in NBS_EXCH state and exchange of DD packets has completed from neighbor. If there are no outstanding requests, NbrFsm transits to NBS_FULL state. But if LS requests are present, NbrFsm transits to NBS_LOAD state and wait for the requests to complete which is what I have described in LS Req, LS Update and LS Ack flow above. Once LS update for the LS request arrives from the neighbor, LS requests are removed from the neighbor's link state request queues. If this causes the neighbor request queue to become empty, then if the NbrFsm state is NBS_LOAD, NbrFsm is triggered with event NBE_LDONE which causes it to transition to state NBS_FULL.

Once link of the current node is in NBS_FULL state with neighbor, that link can now participate in reliable flooding for ospf control traffic form this point onward till it maintains the same state. Now will describe flow for reliable flooding.

DD Exchange can be best described from following flowchart from OPEN OSPF STACK:

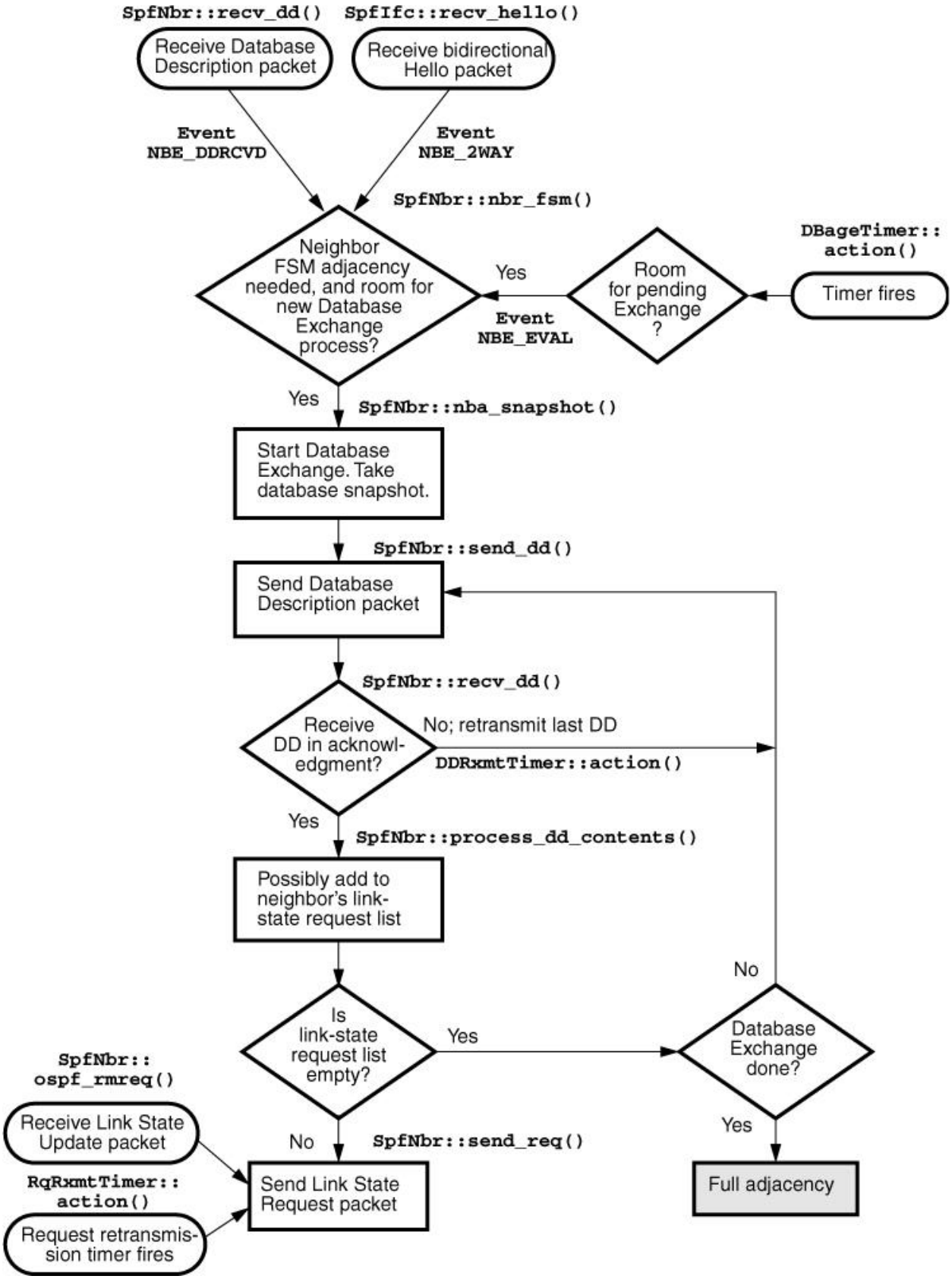


Figure: The Database Exchange Process

Before diving into reliable flooding, we will discuss flow for self router LSA generation and self network-LSA generation.

Self Router-LSA Origination:

Router-LSA is regenerated in following cases:

1) One of the Interface of the node goes up/down.
2) Current node gets elected to DR or vice versa.
3) Existing router-LSA is reoriginated periodically.
4) State of one of the conversations with a neighbor has changed.
5) While configuring one of the interface, it's cost or assigned IFIndex has changed.

LSA header will have following four important fields among others: LSA Age, LSA Type, Link State ID, Advertising Router ID, LS Seq No. For each area, one router will generate one router-LSA. Within an area, each router-LSA is uniquely identified by Link State ID within the LSA database. Typically, for each LS type, a separate LSA database is maintained per area. Most of the time, LSA database is represented as AVL tree. So when a router generates/refreshes it's own LSA, it is updated in it's own LSA database first before flooding to others.

Logical flow of the self router-LSA origination:

1) Within an area, all active interfaces are iterated and correct size of router-LSA is calculated.
2) Calculate the new LS sequence number that should be used for this LSA. Basically LS seq no is the new avatar of the LSA having same Link State ID. There is a separate algorithm for regeneration of LS seq no each time. Won't go into details of that as RFC covers that in details.
3) Build LSA header which has generic as well as router-LSA specific information.
4) Iterate through each interface and for each link corresponding to the interface, interface link state is checked. If state is DOWN, that link is bypassed.

   If the link type is point-to-point, fields are filled as follows:
   Link ID = Neighbors addr;
   Link Data = if addr assigned to interface is zero, assign MIB-II IfIndex (unnumbered) or assign interface ip addr;
   Link Type= Point-to-point router link;
   Metric = cost assigned to local interface.

   And other fields are filled similarly.

   No of links is incremented as and when iteration over interface happens.
5) Once LSA header and body prepared, LSA is ready for origination. As a part of origination, It is checked whether LSA having same link id is present in LSA database or not. If not present, LSA is added to database and flooded. If present, it's content is compared with that of present in database and if different, new LSA is added and flooded.
6) Intra-area route calculation is rescheduled within that area.

List of link types for router-LSA are as follows:

```
enum {                  // Values for link type
   LT_PP = 1,           // Point-to-point router link
   LT_TNET = 2,         // Link to transit network
   LT_STUB = 3,         // Link to stub network
   LT_VL = 4,           // Virtual link
};
```

Transit and stub network has been defined in above sections.
Among all LSAs, router-LSA will be largest in size.

Self Network-LSA Origination:

If the network is of broadcast in nature, one router is selected as Designated router(DR) and another one as backup designated router(BDR) during hello negotiation. Once done, any new node coming into that network will have to establish adjacency with DR before taking part into flooding. Every node in the network will generate router-LSA of link type "Link to transit network" . Only DR in the broadcast network will generate network-LSA for the whole network. An example of network LSA has been given above. For network-LSA, link state id will be ip address of the interface of the DR and advertising router id will be router id of DR. The network-LSA describes all the routers that are attached to the network. The purpose of this LSA is let others know within the area who are the router ids which are participating in a particular broadcast network.

Summary-LSA Origination from ABR(Address Border Router) :

Summary LSA is generated by area border router (ABR) to be advertised into another area. For point to point network, summary LSA details will have as follows:

Link State ID of LSA: Router ID of LSA for which summary is being advertised.
Cost: Cost to reach ABR from specified routerID.
Advertising Router: Router ID of the ABR.

For broadcast network, summary LSA details will have as follows:

Link State ID of LSA: IP network for which summary is being advertised.
Cost: Cost to reach ABR from specified network.
Advertising Router: Router ID of the ABR.

LSA reorigination can be best described from the following flowchart from OPEN OSPF STACK:

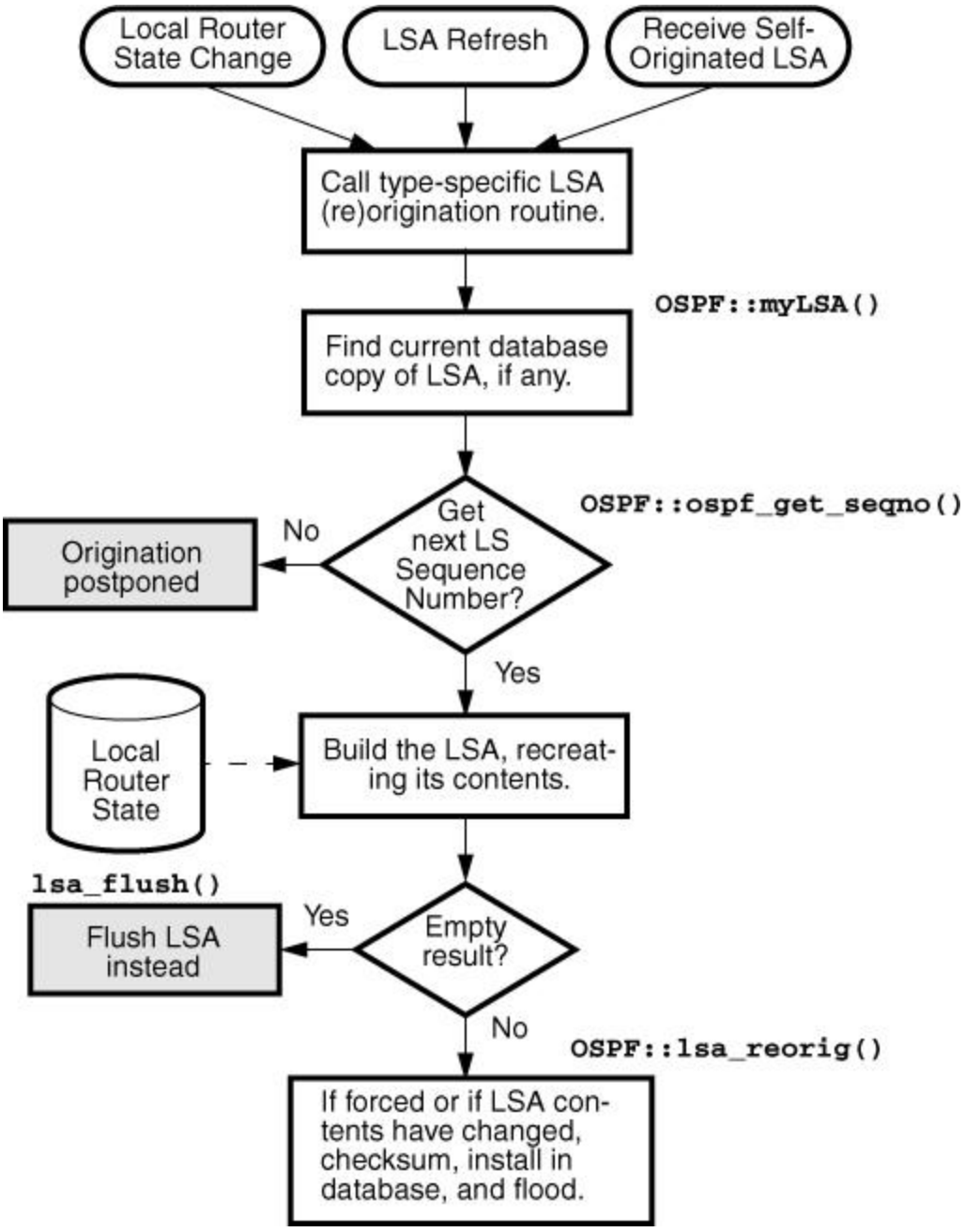


Figure: Steps in originating an LSA

Reliable Flooding:

This has two aspect: adding LSA into the database and flooding the LSA across all it's adjacent links. LSA can be self originated as discussed above or it can be received as a part of LS update for other node within the area. At the end, all nodes in the LSA should have same configuration of LSAs. This is achieved through reliable flooding. This is reliable because every update flooded is acked.

Logical flow of adding LSA to the database:

1) Up till this point, it s confirmed that new LSA is different than database copy. The logic for deciding which is recent between incoming LSA versus same LSA in LSA database has been detailed in previous section and can be found in much detail in RFC.
2) If the database copy is not in any list (DD retransmission list, LS request list, etc) means it is not being any where. So it is just replaced by the new copy in the database.
3) Otherwise, a new database copy is allocated, installed it in the database and parsed for ease of later routing calculations.
4) If opaque-LSA, hook is provisioned so that user's opaque-LSA can be added.
5) If change in LSA content, intra-area and inter-area(depends) routing calculation is scheduled.
   If new LSA changed is router-LSA or network-LSA, full routing calculation is rescheduled.
   If new LSA changed/added is summary-LSA becz of change of ABRs), no need to do full routing calculation. Only incremental routing calculation is needed.

Logical flow of flooding LSA within area:

1) For each neighbor in state NBS_EXCH or greater, add the LSA to the neighbor's link state retransmission list.
2) If it has been added to any retransmission lists, then flood out the interface, except
   if it was received on the interface.
3) Flooding has following three scopes: LocalScope, AreaScope and GlobalScope.
4) If LocalScope, LSA is flooded on the specific interface only. This is the case during DD exchange. This flood is specific to neighbor and achieved using unicast address of neighbor.
5) If AreaScope, LSA is flooded on all the interfaces within the area. This flood is not specific to neighbor and achieved using multicast address.
6) If GlobalScope, LSA is flooded on all the interfaces across areas within OSPF domain. This is typically used for AS-External-LSAs. Detail for AS-External-LSA can be found in RFC.

Reliable flooding can be described from following flowchart from OPEN OSPF STACK:

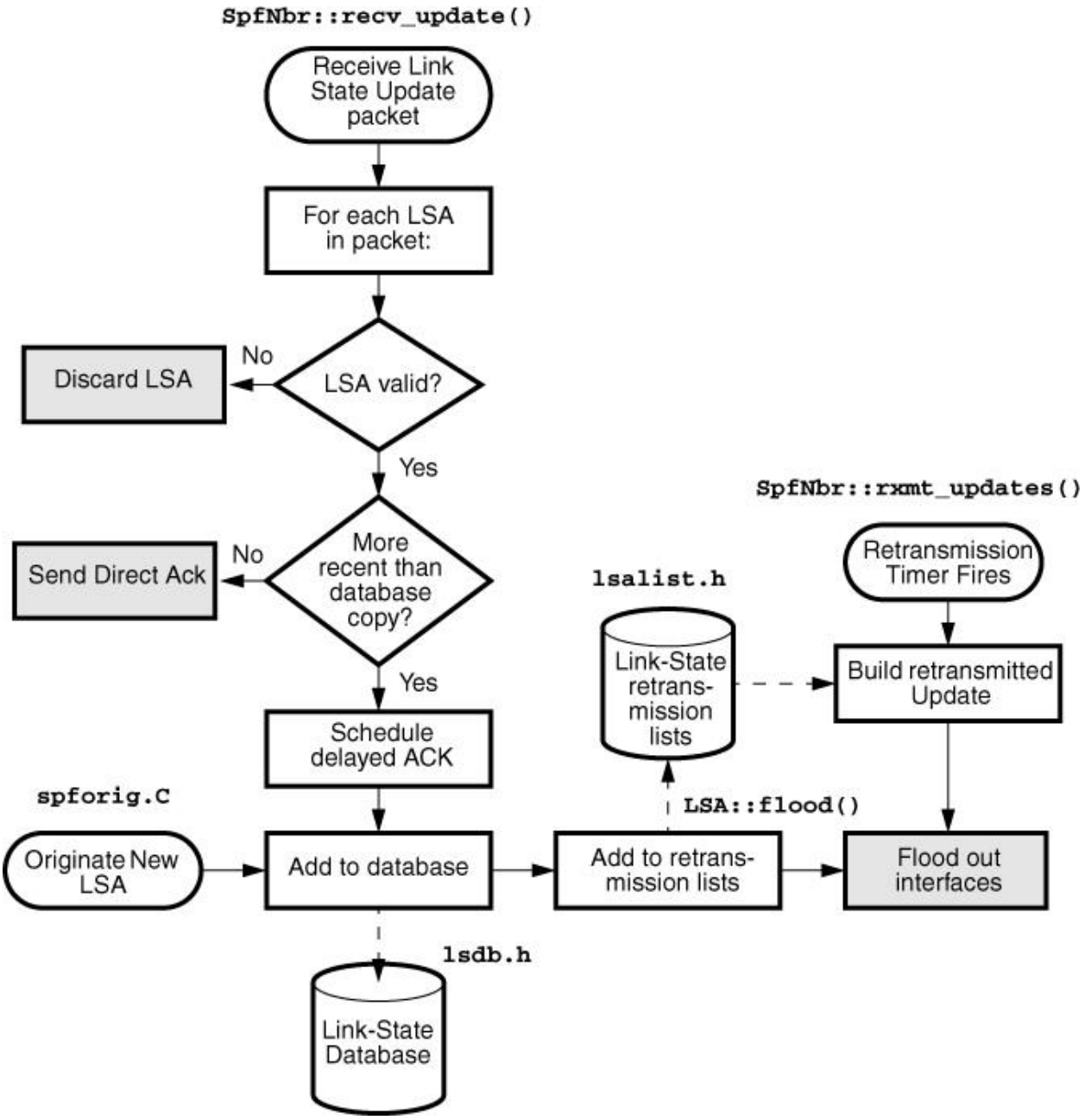


Figure: Data flow of flooding in a single router

Routing Calculation and Generation of Routing Table:

A background timer task runs in OSPF which apart from many other bookkeeping, checks if routing calculation is scheduled. If yes, full routing calculation is triggered over LSA database.
In an OSPF domain, if exists multiple areas, each area contains it's own LSA database of router-LSAs, network-LSAs, summary-LSAs, ASBR-LSAs, AS-External-LSAs and LSAs for multicast routing (generated by MOSPF). Within each area, all nodes will have similar set of router-LSAs and network-LSAs. To generate routing table, Dijkstra is ran per area. If Dijkstra involves only router-LSAs and network-LSAs, it will be called intra-area routing calculation whereas if it involves other LSAs also, then it is called inter-area routing calculation. For the sake of simplicity, will demonstrate summary of Dijkstra for intra-area routing calculation as follows:

1) Initialize the algorithm's data structures. Clear the list of candidate vertices. Initialize the shortest-path tree to only the root (which is the router doing the calculation).
2) Call the vertex just added to the tree vertex V. Examine the LSA associated with vertex V. This is a lookup in the Area A's link state database based on the Vertex ID. For each described link, (say it joins vertex V to vertex W):
    a) If this is a link to a stub network, examine the next link in V's LSA. Links to stub networks will be considered in the second stage of the shortest path calculation.
    b) Otherwise, W is a transit vertex (router or transit network). Look up the vertex W's LSA (router-LSA or network-LSA) in Area A's link state database. If the LSA does not exist, or its LS age is equal to MaxAge, or it does not have a link back to vertex V, examine the next link in V's LSA.
    c) If vertex W is already on the shortest-path tree, examine the next link in the LSA.
    d) Calculate the link state cost D of the resulting path from the root to vertex W. D is equal to the sum of the link state cost of the (already calculated) shortest path to vertex V and the advertised cost of the link between vertices V and W. If D is:
        - Greater than the value that already appears for vertex W on the candidate list, then examine the next link.
        - Equal to the value that appears for vertex W on the candidate list, calculate the set of next hops that result from using the advertised link. Input to this calculation is the destination (W), and its parent (V). This set of hops should be added to the next hop values that appear for W on the candidate list.
        - Less than the value that appears for vertex W on the candidate list, or if W does not yet appear on the candidate list, then set the entry for W on the candidate list to indicate a distance of D from the root. Also calculate the list of next hops that result from using the advertised link, setting the next hop values for W accordingly. it takes as input the destination (W) and its parent (V).

    e) If at this step the candidate list is empty, the shortestpath tree (of transit vertices) has been completely built and this stage of the procedure terminates. Otherwise, choose the vertex belonging to the candidate list that is closest to the root, and add it to the shortest-path tree (removing it from the candidate list in the process). Note that when there is a choice of vertices closest to the root, network vertices must be chosen before router vertices in order to necessarily find all equal-cost paths.
3) Iterate the algorithm by returning to Step 2.

Once Dijkstra is complete for an area, we will have shortest path from current node to all other nodes in the area. During Dijkstra run, as and when shortest path to a new node form current node is discovered, a routing table entry is entered/modified in the routing table for that area with next hop addresses (including multi next-hop for equal cost path) as well as cost to that node via this hop updated. Hence when Dijkstra run completes, we have complete routing table ready for that area including next hop database.

Triggering intra-area as well inter-area routing calculation can be best described from following flowchart from OPEN OSPF STACK:

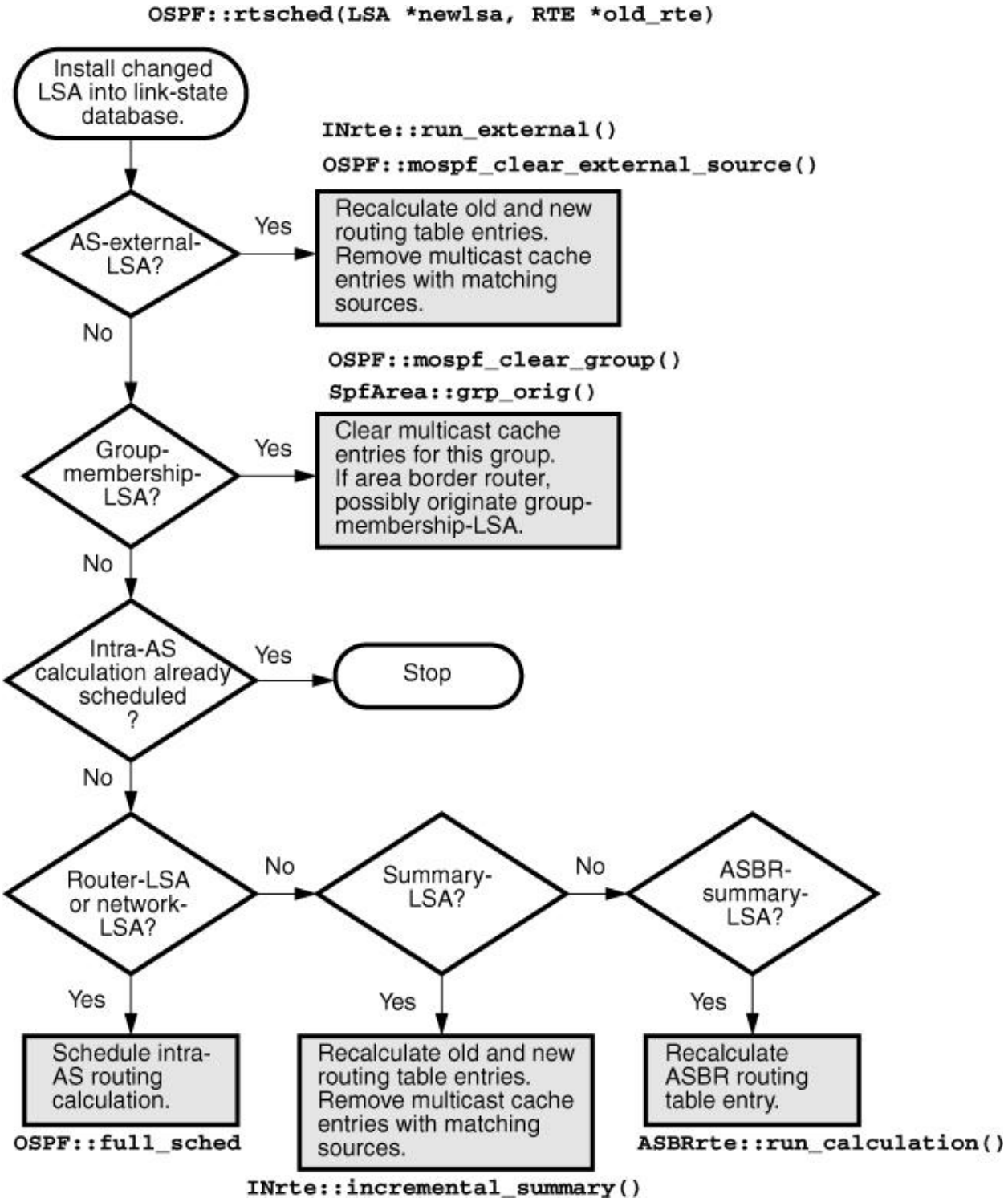


Figure: Triggering recalculation of routing table entries

Globals for OSPF:

In general, following globals are maintained in any OSPF implementation:

OSPF *ospf - This is the global data structure representing OSPF domain. LSA databases per area is contained within that. LSA database is mostly maintained as AVL tree.

PriQ timerq - Global timer queues. Normally timer queues are implemented as priority queue. Various timers like Hello timer, Wait timer, LS Age timer, etc are built on the top of this.

OspfSysCalls *sys - System call interface. This is the interface wrapper through which routing updates are sent to kernel routing table. In linux, this wrapper is built over NETLINK sockets whereas in BSD unix, this is built over ROUTE sockets.

INtbl *inrttbl - IP routing table. This represents RIB (Routing Information Base) base of OSPF. Usually maintained as AVL tree.

FWDtbl *fa_tbl- Forwarding address table. Mostly used with AS-External-LSAs.

INrte *default_route - The default routing entry (0/0). This entry is used within stub area because all nodes within stub area is blind to other areas.

ConfigItem *cfglist – This maintains global configurations pushed to ospf domain as per OSPF MIB.

PatTree MPath::nhdb - Next hop(s) database . Each routing entry in the table point to this database to get next hop details. Usually maintained as patricia tree.

SPFtime sys_etime - Time since ospf program start.