Вычислительные технологии, 2022, том 27, № 4, с. 98–107. © ФИЦ ИВТ, 2022 ISSN 1560-7534
Computational Technologies, 2022, vol. 27, no. 4, pp. 98–107. © FRC ICT, 2022 eISSN 2313-691X

DOI:10.25743/ICT.2022.27.4.008

Simulation of distributed sequence number routing algorithm

H. Khayou*, M. A. Orlova, L. I. Abrosimov
National Research University "MPEI", 111250, Moscow, Russia
*Corresponding author: Khayou Hussein, e-mail: hussein.khayou@gmail.com
Received December 07, 2021, revised January 13, 2022, accepted April 06, 2022.

Scaling is a growing challenge in networks, especially in data centers. There is a need for simple scalable routing protocols to ease the automation and management of constantly growing networks. Distributed sequence number (DSN) protocol was proposed by the authors to address, the challenges of routing in large networks. DSN uses a hybrid distance vector link state algorithm to achieve scalability and simplicity. In this work, the simulation of DSN algorithm is presented.

Keywords: loop free routing, link state, distance vector, sequence number.

Citation: Khayou H., Orlova M.A., Abrosimov L.I. Simulation of distributed sequence number routing algorithm. Computational Technologies. 2022; 27(4):98–107. DOI:10.25743/ICT.2022.27.4.008.

Introduction

Data centers and the services they provide have changed significantly over the past decades. They are constantly growing in scale and the volume of traffic they support. Traffic patterns and requirements have also changed dramatically. Modern data centers include more traffic exchanged between servers, which is called east-west traffic. On the contrary, north-south traffic is traffic from local networks to external networks and vice versa [1]. Thus, data centers' requirements in terms of the topology and the routing protocol have also evolved. The routing protocol should be simple with ease of implementation, in terms of programming code complexity and ease of operational support. Also, the set of features and protocols is limited so that they are supported by several vendors. The period and radius of failure of the protocol or equipment are contained. Traffic engineering can be used with the routing protocol [2]. This is why, especially in terms of scalability, large enterprises have resorted to using BGP (Border Gateway Protocol) for routing in ultra-large data centers [3].

In [4] a simple hybrid distance vector link-state protocol was proposed, which has the scalability and the simplicity needed for big data centers. The distance vector algorithm is preserved and loop freedom operations are passed on to metric computation by adding a sequence number (SN) to the metric. The distance vector algorithm is used to calculate the best paths between the nodes. Starvation (when a node has no feasible successor) is resolved by using the distributed sequence number algorithm, which distributes the request to increase the sequence number (RISN) in the updates until it reaches the route's owner. After the RISN reaches the route's owner, the route's owner issues an update with an increased SN. This acts as a reset for calculation for this route. Full details of the algorithm could be found in [4]. The link-state algorithm is used to distribute the networks (prefixes) within the

same area and between the different areas instead of the links themselves. The combination of a distance vector and link-state helps to reduce the amount of link-state information in the database which is needed to be maintained and distributed by each node since prefixes instead of the links are advertised using the link-state algorithm. Prefixes can also be further summarized using route summarization. The routing domain can be divided into areas as in link-state protocols to increase the scalability of the protocol, which was not available in pure distance vector protocols. The distance vector algorithm is used to compute the routes to the nodes instead of the networks connected to each node, thus, no dependency on IP connectivity is required. This also limits the number of updates after topology changes. Unequal load balancing can be supported by the protocol as in EIGRP (Enhanced Interior Gateway Routing Protocol) because each node maintains a list of feasible successors. Protocol correctness can be inferred from the concept of monotone routing in [5]. In monotone routing, if the routing matrix "decreases" in one iteration, it will continue decreasing to convergence.

In this paper, we present an implementation for the routing algorithm used in the DSN protocol proposed in [1]. The paper is organized as follows: next section scans related work. Section 2 gives details about the protocol implementation. Section 3 presents the results and statistics about the exchanged updates in the algorithm.

1. Related work

EIGRP is a Cisco proprietary loop-free protocol. It was released in 2013 and published with informational status [6]. EIGRP was previously advertised as a hybrid routing protocol, however, Cisco now classifies it as an advanced distance vector protocol [6, 7]. EIGRP is a loop-free routing protocol that employs the DUAL algorithm (Diffusing Update Algorithm). In DUAL a node can only change its successor if the feasibility condition is met [8]. The feasible condition guarantees that when a node selects a feasible successor to a certain destination, it will not form a loop in routing to that destination. EIGRP has the advantage of fast convergence and scalability in small and medium networks. EIGRP also supports unequal-cost multipath routing using the concept of feasible successors. However, EIGRP was not adopted to our knowledge by other vendors. In addition, EIGRP does not provide a mechanism to support the division of the routing domain into areas or subdomains.

Babel is a distance vector routing protocol [9], where a non-decreasing sequence number is added to the metric. The cost now is in the form (s, d) of where s is the sequence number and d is the distance. When a node starves i. e., it cannot find any feasible successor to a certain destination, then it will send a unicast request to the destination to increase its sequence number. The request is sent over the non-feasible network. Babel has the advantage of simplicity and small size. It is suitable for routing in highly dynamic wireless mesh networks. However, it is not well suited for large and relatively stable networks, as it depends on periodic updates [9]. Furthermore, in Babel, the network during the reconfiguration phase can be unstable, because upstream nodes which did not receive updates that their current routes are invalid will continue to forward traffic over the invalid routes until they receive the new routing information with the increased sequence number.

2. Algorithm implementation

To implement the algorithm, we have created a custom event-driven simulator. In the simulator, routers are represented as nodes on a graph, where the edges of the graph represent

links between routers. We have built a random graph generator to create the network topology. Routers exchange routing updates among themselves on the edges of the graph. Figure 1 shows a class diagram for the various messages exchanged by routers in the system. Figure 2 shows the class diagram of the routing logic.

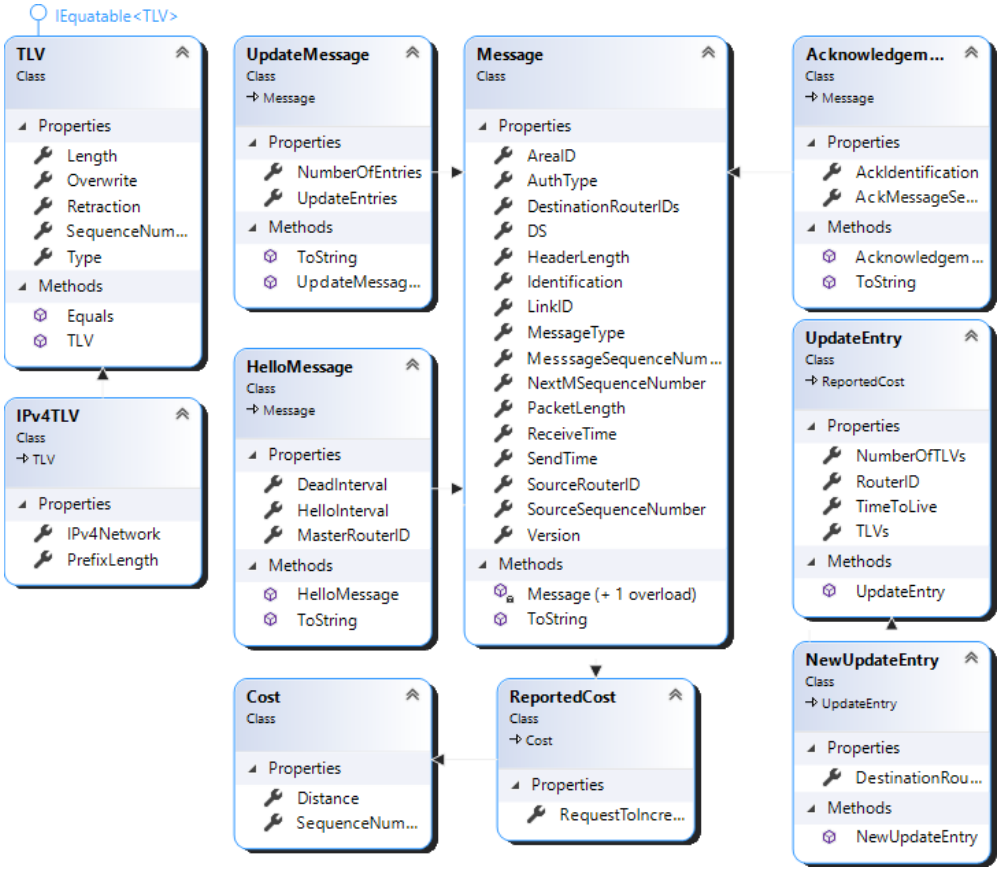

Fig. 1. Messages class diagram

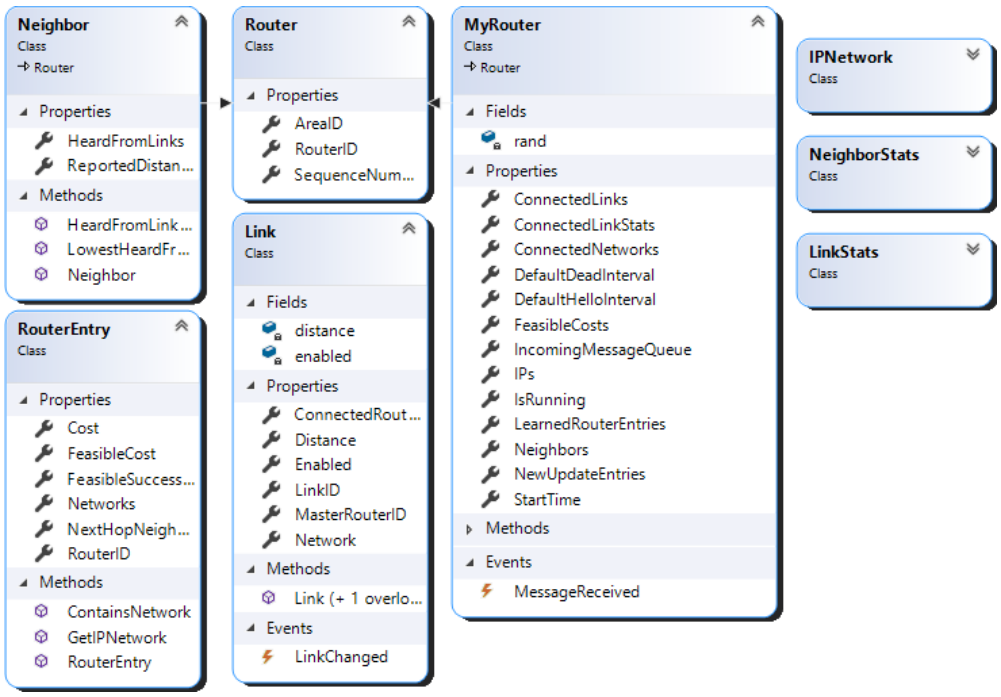

Fig. 2. Routers model class diagram

Each message consists of a common header and a body. The common header is comprised of the following fields: Protocol Version, Differentiated Services, Packet Length (Total length of the message), Identification (Each message is identified by the message identification, Message sequence number and the source router ID for reliable transfer), Message Sequence Number, Source Router ID, Sequence Number of the source router, Area ID, Message Type (Hello, Acknowledgment or Update), Header Length, Number of router IDs in the list of destination routers' ID, List of destination router IDs, Checksum, Authentication Type, Authentication(not implemented).

Message receipt should be acknowledged except for hello messages. Message identification and Message Sequence number are echoed in the Ack Identification and Ack Message sequence number fields in the Acknowledgment message.

Hello messages should be sent each hello interval to keep the link between the routers alive. The neighbour is declared dead if the dead interval expires and no message is received from the neighbour. Another function is to elect a master router for non-point-to-point links. The master will distribute the link state information (networks) for new routers that join the link. The rest of neighbours also should send their distances and sequence numbers to other routers in the area. Capabilities can include router physical and logical addresses and protocol extensions.

Update messages contains a list of update entries. Each update entry consists of router-id and sequence number of the entry's owner. The distance from the source router of the update message to the owner. RISN flag should be set when there is a request to the owner to increase the sequence number because of starvation (non-existence of feasible successors to the owner). Each update entry is comprised of a list of TLVs (Type — Length — Value). This list contains the connected networks to the owner (which are distributed by the link-state algorithm). For example, IPv4 addresses and prefixes are distributed using the IPv4TLV. A retraction flag should be set when the owner has lost connection to a network and wants to remove it from the database. Each TLV is accompanied by a separate sequence number generated and maintained by the owner. The sequence number is incremented when a change occurs to the TLV. Retracted TLVs should not be immediately deleted from the database. Retracted TLVs will be marked with Retracted-flag and can be removed only after the expiration of a configurable timeout value. This is to avoid confusion when a node receives an obsolete advertisement. The advertisement then will be ignored because it will contain a smaller sequence number for the TLV.

The exchanged messages are stored in a database and a log file to collect statistics about the performance of the protocol.

Every router maintains the following state tables: 1) a table of connected links to store information about the directly connected links; 2) a neighbours table to store information about the adjacent neighbours of the router; 3) a learned router entries table, to maintain information about the learned routes; 4) a new update entry table which contains the new update entries that need to be sent to the neighbours. For every neighbour in the neighbour table, we have a table for the links the neighbour is heard from, and a table for the reported distances from the neighbour. In every router entry, we maintain the cost and feasible cost for the route. In addition, we maintain tables for every route to store feasible successors, next hops (or simply successors), and networks (link-state information).

Handling update messages algorithm is explained in algorithm 1. *UpdateNeighborReportedDistance* updates (or adds if not existing) the reported distance of the neighbour for the router id specified in the update entry. *UpdateLinkState* updates (Add, Overwrite or

Algorithm 1. Handling update messages algorithm

```
1: procedure UpdateMessageReceived(updateMessage)
2:     neighbor ← Neighbors[updateMessage.SourceRouterID]                ▷ Exit if failed
3:     for all updateEntry in updateMessage.UpdateEntries do
4:         newDistance ← updateEntry.Distance + GetLinkByID(updateMessage.LinkID)
5:         routerEntry ← LearnedRouterEntries[updateEntry.RouterID]
6:         if routerEntry = null then
7:             routerEntry ← CreateNewRouterEntry(updateEntry, newDistance)
8:             newUpdateEntry ← CreateNewUpdateEntry(updateEntry,newDistance)
9:             routerEntry.NextHopNeighbors.Add(neighbor)
10:            routerEntry.FeasibleSuccessors.Add(neighbor)
11:            UpdateNeighborReportedDistance(neighbor, updateEntry)
12:            UpdateLinkState(updateEntry, routerEntry, newUpdateEntry)
13:            AddDestinationsRouters(newUpdateEntry)
14:            AddOrUpdateNewUpdateEntry(newUpdateEntry)
15:            LearnedRouterEntries.Add(routerEntry)
16:        else
17:            if (
    updateEntry.RouterID = self.RouterID and updateEntry.RISN = true
    and updateEntry.SequenceNumber = self.SequenceNumber
               )) then
18:                IncreaseSelfRouterSequenceNumber(routerEntry)
19:                newUpdateEntry ← CreateSelfNewUpdateEntry()
20:                AddDestinationsRouters(newUpdateEntry)
21:                AddOrUpdateNewUpdateEntry(newUpdateEntry)
22:            else if  (
    updateEntry.SequenceNumber > routerEntry.FeasibleCost.SequenceNumber
    or (updateEntry.SequenceNumber = routerEntry.FeasibleCost.SequenceNumber)
    and (newDistance < routerEntry.FeasibleCost.Distance)
               ) then
23:                oldSeqNo ← routerEntry.FeasibleCost.SequenceNumber
24:                routerEntry.FeasibleCost.Distance ← newDistance
25:                routerEntry.FeasibleCost.SequenceNumber ← updateEntry.SequenceNumber
26:                routerEntry.Cost.Distance ← newDistance
27:                routerEntry.Cost.SequenceNumber ← updateEntry.SequenceNumber
28:                routerEntry.NextHopNeighbors.Clear()
29:                routerEntry.NextHopNeighbors.Add(neighbor)
30:                newUpdateEntry ← CreateNewUpdateEntry(updateEntry, newDistance)
31:                routerEntry.FeasibleSuccessors.Add(neighbor)
32:                UpdateNeighborReportedDistance(neighbor, updateEntry)
33:                UpdateFeasibleSuccessors(routerEntry) ▷ Some of the old feasible successor have
    to be removed from the feasible successors table if they do not satisfy the feasibility condition
34:                UpdateLinkState(updateEntry, routerEntry, newUpdateEntry)
35:                if (updateEntry.SequenceNumber > oldSeqNo) then
36:                    routerEntry.Cost.RISN ← updateEntry.RISN
37:                    newUpdateEntry.RISN ← updateEntry.RISN
38:                else
39:                    routerEntry.Cost.RISN ← routerEntry.Cost.RISN or updateEntry.RISN
40:                    newUpdateEntry.RISN ← newUpdateEntry.RISN or updateEntry.RISN
```

Sequel of algorithm 1

```
41:                 end if
42:                 AddDestinationsRouters(newUpdateEntry)
43:                 AddOrUpdateNewUpdateEntry(newUpdateEntry)
44:             else if (
    updateEntry.SequenceNumber = routerEntry.FeasibleCost.SequenceNumber
            ) then
45:                 UpdateNeighborReportedDistance(neighbor, updateEntry)
46:                 if Reported Distance changed then
47:                     if (updateEntry.Distance < routerEntry.FeasibleCost.Distance) then
48:                         routerEntry.FeasibleSuccessors.Add(neighbor)
49:                     else
50:                         routerEntry.FeasibleSuccessors.Remove(neighbor)
51:                     end if
52:                     UpdateDistance(routerEntry)
53:                 end if
54:                 routerEntry.Cost.RISN ← routerEntry.Cost.RISN or updateEntry.RISN
55:                 UpdateLinkState(updateEntry, routerEntry, newUpdateEntry)
56:                 if There is a change in cost or RISN or link state then
57:                     newUpdateEntry ← CreateNewUpdateEntryFromRouterEntry(routerEntry)
58:                     AddDestinationsRouters(newUpdateEntry)
59:                     AddOrUpdateNewUpdateEntry(newUpdateEntry)
60:                 end if
61:             end if
62:         end if
63:     end for
64: end procedure
```

Retract) the networks components of the router entry that corresponds to the router id specified in the update entry, and it conveys these changes to the new update entry (to be sent to neighbours). *AddDestinationsRouters* adds the appropriate destination router-ids to the new update entry respecting the rules of incremental triggered updates with split horizon. *AddOrUpdateNewUpdateEntry* update the new update entries table. Corresponding update messages are then created by the scheduler for these entries. *UpdateDistance* updates the current distance for the router-id (specified in the update entry) by choosing the lowest cost using only the feasible successors costs. It also updates the next hops for this router-id. If there is no feasible successor, the distance will be infinity and RISN is set.

3. Results

We first present the exchanged message in the implemented algorithm for a simple topology depicted in Fig. 3.

Figure 4 shows a subset of the first exchanged messages during the initial convergence phase where node "1" is the source or the destination of the message.

Initially, router "1" sends information about itself (sequence number and distance (0), and the directly connected networks to router "1") to routers "2", "3", and "4". When a router learns about another router, it sends this information to its neighbours with its distance to reach the new learned router.

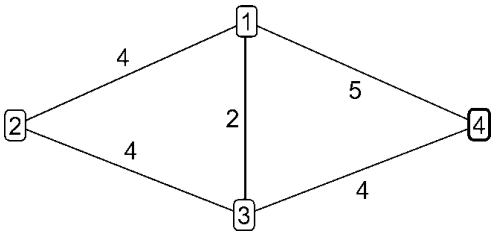

Fig. 3. Simple network topology

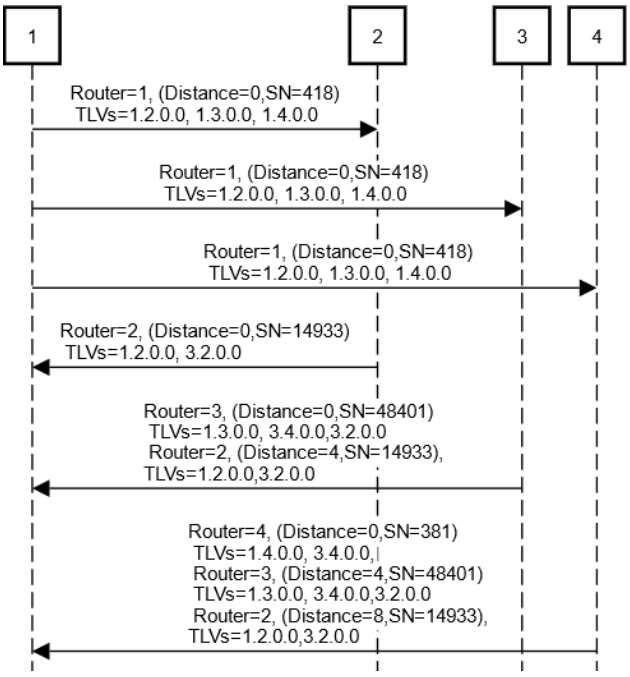

Fig. 4. Network initializing phase

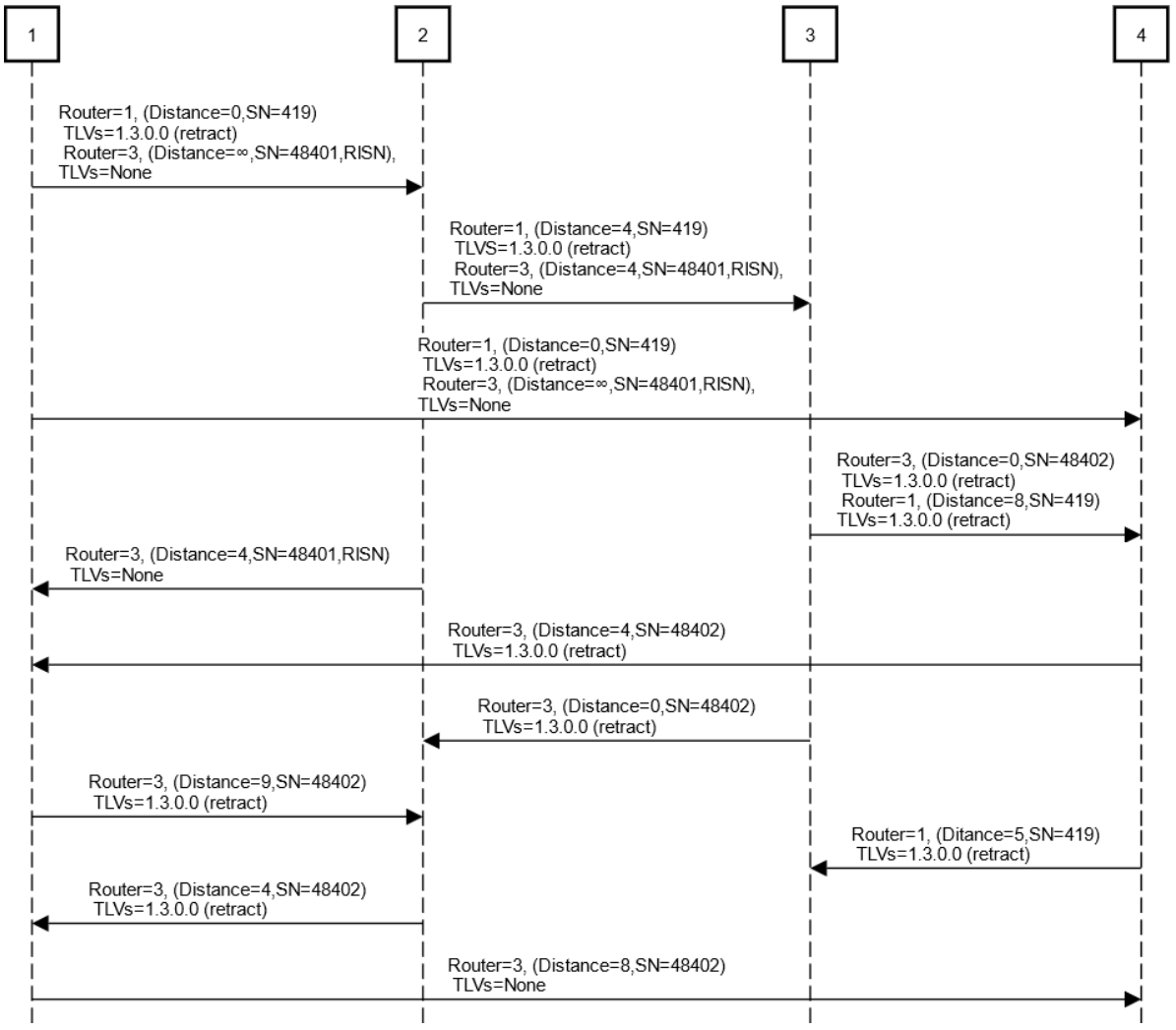

Fig. 5. Exchanged messages for reconvergence after the failure of link 1–3

Figure 5 shows the exchanged messages for reconvergence after the link between “1” and “3” is disconnected.

Router “1” retracts the network for link “1–3” (the prefix “1.3.0.0”) and sends a request to increase the sequence number for router 3 (with infinite distance), because router “1” has no feasible successor for router “3”. The request is sent to routers “2” and “4”. After the request to increase the sequence reaches router “3” it sends updates with the increased sequence number and also retracts the prefix “1.3.0.0” as it is also no longer reachable from “3”. The updates are propagated in the network respecting the rules of incremental triggered updates with split horizon.

Next, we provide statistics about the exchanged update messages in the randomly generated topologies with a variant number of routers and links. In the scenario, one of the links is disconnected when the simulation time equals $S/3$, where S is the duration time. The link is reconnected when the simulation time equals $2S/3$. In all the figures: Series 1 is the measured value in the first third (before the link is disconnected), while series 2 is in the second third (after the link is disconnected) and series 3 in the last third (after the link is restored). In Figures 6–9, *a*, the horizontal axis represents the number of links in

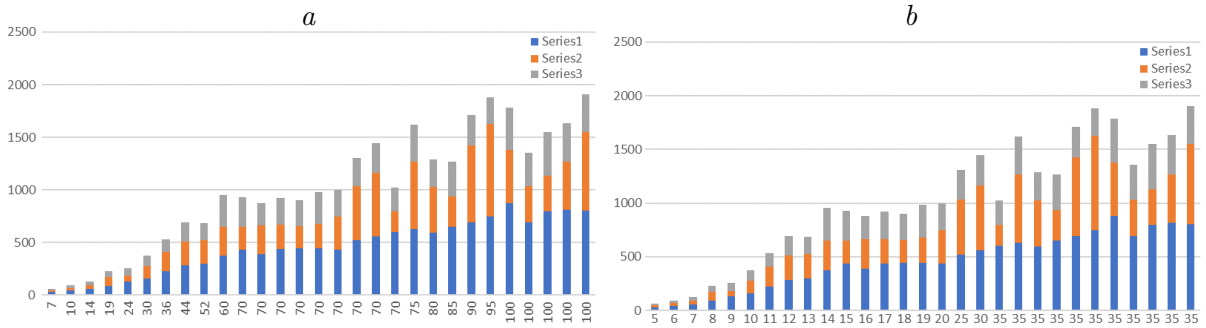

Fig. 6. Number of update messages by the number of links (*a*) and nodes (*b*)

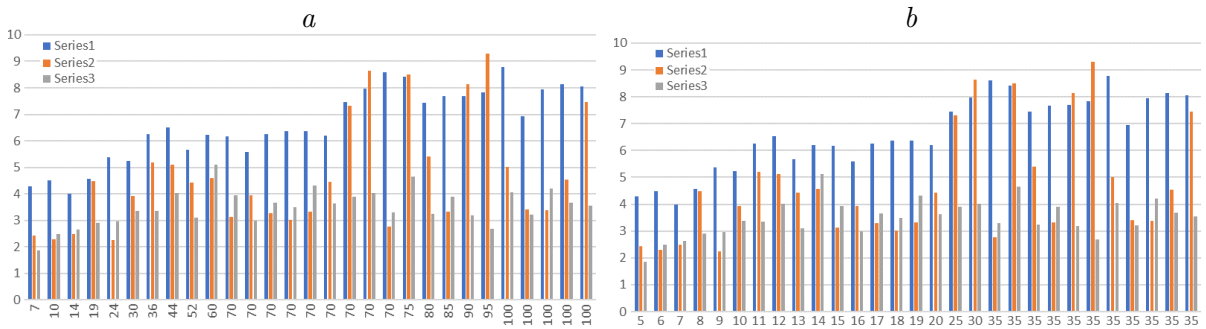

Fig. 7. Average number of update messages per link by the number of links (*a*) and nodes (*b*)

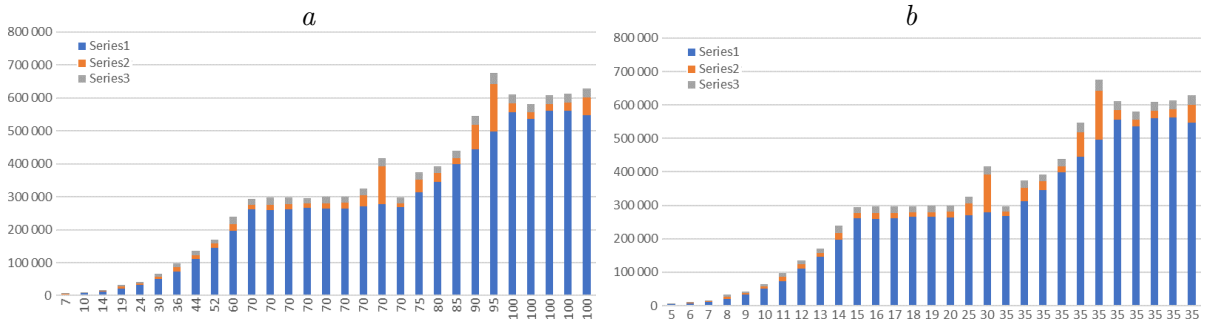

Fig. 8. Total size of exchanged update messages (in bytes) by the number of links (*a*) and nodes (*b*)

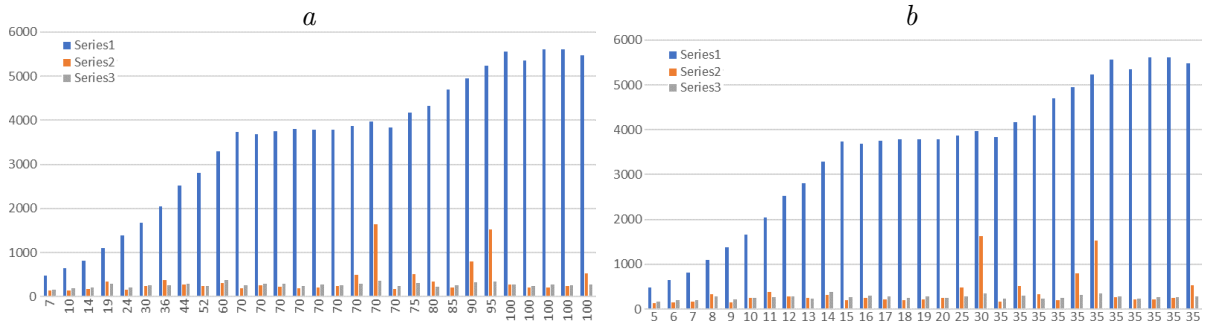

Fig. 9. Average total size of updates per link by the number of links (a) and nodes (b)

the experiment. Whereas in Figures 6–9, b, the horizontal axis represents the number of routers. Figure 6 show the the number of update messages. Figure 7 show the average number of updates per link for varying topologies by the number of links and the number of nodes in the topology respectively. Figure 8 show the total size of updates exchanged in the experiment. Figures 9 show the average total size per link. The protocol is showing fast convergence speed with minimal size of information exchanged to reach convergence. It is also clear that the exchanged information for changes in the topology after convergence is marginal compared to the initial size of information to reach convergence.

Conclusion

The peculiarity of the presented algorithm is the decoupling of forwarding from routing. Routing computes the cost of the path to the destination node, while forwarding computes the cost of the path to some network connected to the remote router. Therefore, we divided the task into two sub-tasks. The first task is computing the path to the infrastructure nodes, which is achieved using the modified distance vector algorithm with sequence numbers. The second task is distributing the mapping between the networks and the infrastructure nodes, which is achieved using the link state algorithm.

This way, the protocol will have a much smaller database to maintain than other link state protocols. This is because we only advertise the networks of the connected links rather than the links themselves, and networks can also be further summarized using route summarization. The protocol is also a distance vector only for infrastructure nodes. Hence, the number of updates after topology changes will be limited. Large topologies can be divided into areas, which was not possible in existing distance vector protocols. When multiple areas are used, all routers (including edge routers) belong to only one area. Thus, edge routers will be less loaded because they only participate in distance vector updates for their areas. Networks are distributed using link state algorithm, thus route attributes can be added for policy based routing.

The implementation of the algorithm showed that the size of the exchanged information between nodes for reconvergence is marginal compared to the one at the initial phase of convergence. Future work is to implement and simulate the neighbour discovery protocol, reliable transfer protocol, and multiarea routing algorithm with policy based routing decision.

References

[1] **Dutt D.G.** BGP in the data center. O'Reilly Media. 2017. Available at: `https://www.oreilly.com/library/view/bgp-in-the/9781491983416`.

[2] **Lapukhov P., Premji A., Mitchell J.** Use of BGP for routing in large-scale data centers. Technical Report. 2016. Available at: https://doi.org/10.17487/rfc7938.

[3] **Deepankar M., Ramasamy K.** Network routing: algorithms, protocols, and architectures. 2nd edition. Elsevier, Morgan Kaufmann Publishers; 2017. Available at: https://www.elsevier.com/books/network-routing/medhi/978-0-12-800737-2.

[4] **Khayou H., Orlova M.A., Abrosimov L.I.** A hybrid distance vector link state algorithm: distributed sequence number. International Journal of Computer Networks and Applications. 2021; 8(3):203. DOI:10.22247/ijcna/2021/209188.

[5] **Khayou H., Rudenkova M.A., Abrosimov L.I.** On the algebraic theory of loop free routing. International Conference on Distributed Computer and Communication Networks. 2020: 161–175. Available at: https://link.springer.com/chapter/10.1007/978-3-030-66471-8_14?error=cookies_not_supported&code=437f9e3e-cba1-46e9-ab29-5d0bd096b913.

[6] **Savage D., Ng J., Moore S., Slice D., Paluch P., White R.** Cisco's enhanced interior gateway routing protocol (EIGRP). Technical Report. 2016. DOI:10.17487/rfc7868.

[7] **Bruno A.** CCIE routing and switching exam certification guide. Cisco Press; 2002: 741. Available at: https://books.google.ru/books?id=NzYb1pPZTB0C.

[8] **Garcia-Lunes-Aceves J.J.** Loop-free routing using diffusing computations. IEEE/ACM Transactions on Networking. 1993; 1(1):130–141. DOI:10.1109/90.222913.

[9] **Chroboczek J., Schinazi D.** The babel routing protocol. Technical Report. 2021. DOI:10.17487/RFC8966.

Вычислительные технологии, 2022, том 27, № 4, с. 98–107. © ФИЦ ИВТ, 2022 ISSN 1560-7534
Computational Technologies, 2022, vol. 27, no. 4, pp. 98–107. © FRC ICT, 2022 eISSN 2313-691X

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

DOI:10.25743/ICT.2022.27.4.008

Моделирование алгоритма маршрутизации распределенных порядковых номеров

Х. Хаю*, М. А. Орлова, Л. И. Абросимов

Национальный исследовательский университет "Московский энергетический институт", 111250, Москва, Россия

*Контактный автор: Хаю Хуссейн, e-mail: hussein.khayou@gmail.com

Поступила 07 декабря 2021 г., доработана 13 января 2022 г., принята в печать 06 апреля 2022 г.

Аннотация

Масштабирование является растущей проблемой в сетях, особенно в центрах обработки данных. Существует потребность в простых масштабируемых протоколах маршрутизации для упрощения автоматизации и управления постоянно растущими сетями. DSN (Протокол распределенного порядкового номера) был предложен авторами для решения проблем маршрутизации в больших сетях. DSN использует гибридный алгоритм состояния каналов и дистанционно-векторный для достижения масштабируемости и простоты. В этой работе представлено моделирование алгоритма DSN.

Ключевые слова: бесцикловая маршрутизация, состояние каналов, дистанционно-векторная маршрутизация.

Цитирование: Хаю Х., Орлова М.А., Абросимов Л.И. Моделирование алгоритма маршрутизации распределенных порядковых номеров. Вычислительные технологии. 2022; 27(4):98–107. DOI:10.25743/ICT.2022.27.4.008. (на английском)