Mathematical Biosciences and Engineering

MBE, 20(9): 16528–16550.
DOI: 10.3934/mbe.2023737
Received: 29 May 2023
Revised: 07 August 2023
Accepted: 10 August 2023
Published: 17 August 2023

http://www.aimspress.com/journal/MBE

Research article

FastCAT: A framework for fast routing table calculation incorporating multiple protocols

Jianfei Cai[1,2], Guozheng Yang[1,2,*], Jingju Liu[1,2] and Yi Xie[1,2]

[1] College of Electronic Engineering, National University of Defense Technology, Hefei 230037, China
[2] Anhui Province Key Laboratory of Cyberspace Security Situation Awareness and Evaluation, China

* **Correspondence:** Email: yangguoz0218@163.com.

Abstract: Currently, most network outages occur because of manual configuration errors. Therefore, it is essential to verify the correctness of network configurations before deployment. Computing the network control plane is a key technology for network configuration verification. We can verify the correctness of network configurations for fault tolerance by generating routing tables, as well as connectivity. However, existing routing table calculation tools have disadvantages such as lack of user-friendliness, limited expressiveness, and slower speed of routing table generation. In this paper, we present FastCAT, a framework for computing routing tables incorporating multiple protocols. FastCAT can simulate the interaction of multiple routing protocols and quickly generate routing tables based on configuration files and topology information. The key to FastCAT's performance is that FastCAT focuses only on the final stable state of the OSPF and IS-IS protocols, disregarding the transient states during protocol convergence. For RIPv2 and BGP, FastCAT computes the current protocol routing tables based on the protocol's previous state, retaining only the most recent protocol routing tables in the latest state. Experimental evaluations have shown that FastCAT generates routing tables more quickly and accurately than the state-of-the-art routing simulation tool, in a general network of around 200 routers.

Keywords: network configuration; verification; routing protocol; routing table; message passing

1. Introduction

Because network configuration involves multiple types of routing protocols and complex network topologies and the configuration languages are abstract and not easily understood, manually configuring a network is often error-prone. For example, on 8 July 2022, Rogers Communications Group experienced a nationwide network outage that lasted for approximately 19 hours due to a router failure caused by an error by a network operator in updating the network configurations, resulting in compensation payments to customers of US$28−70 million. Therefore, preventing network outages by misconfigurations is crucial to ensure the high reliability and availability of the network.

Network control plane simulation technology plays a crucial role in preventing network outages. It allows network operators to generate routing tables through the simulation of the control plane, enabling them to perform comprehensive checks based on these routing tables to prevent network outages.

In practice, before deploying network configurations into a production environment, network operators typically conduct control plane simulations. This involves generating routing tables using network configuration files and topology information, which are then utilized to perform offline verification of network properties. By analyzing the routing tables, operators can identify and rectify network configuration errors before deployment, thus mitigating the risk of network outages.

Furthermore, routing tables are invaluable tools for verifying specific network designs, such as load balancing. Network operators can use the routing tables to verify the intended configuration and ensure that it aligns with the desired network behavior.

Additionally, considering the potential vulnerabilities in device firmware, cross-checking the routing table in the real network with the one generated through simulation can help uncover unexpected firmware issues during configuration file deployment. This process aids in detecting and addressing any firmware-related problems that may impact network stability and performance.

Batfish [1] and FastPlane [2] are two prominent routing table calculation tools that operate based on configuration files. Batfish offers support for a wide range of routing protocols and numerous vendors. However, it requires network operators to be proficient in the associated query language, making it less user-friendly. Additionally, the initial generation of routing tables using Batfish can be time-consuming. On the other hand, FastPlane excels in generating routing tables at a faster pace. However, it is specifically designed for data centers where BGP routing priorities exhibit a monotonically decreasing pattern.

To address the limitations of existing routing table calculation tools, including lacking user-friendliness, limited expressive capabilities, and slower speed of routing table generation, we propose a multi-protocol routing table calculation framework called FastCAT (Fast Configuration Analysis Tool). FastCAT leverages message passing to simulate crucial interactions among routing protocols in actual networks. With FastCAT, network operators can simply input the device configuration files and network topology, enabling the generation of routing tables for each device based on the current configurations. FastCAT offers support for commonly used routing protocols such as OSPF, IS-IS, RIPv2, and BGP.

In summary, our main contributions are listed as follows:

- To solve the problem that existing routing table calculation tools lack user-friendliness and have limited expressive capabilities, we design a routing table calculation framework called FastCAT which incorporates multiple protocols.
- To improve the speed of generating routing tables, we propose fast simulation algorithms based on message passing for routing protocols. The algorithms effectively capture the essential behaviors of various routing protocols, enabling the swift generation of routing tables.
- To validate the efficiency and accuracy of our algorithm and framework, extensive

experimentation is conducted using real network topology data. The results affirm the effectiveness and reliability of the proposed approach.

The remainder of this paper is organized as follows. Section 2 reviews related works. Section 3 presents the overall framework of FastCAT. Section 4 provides the simulation algorithms of routing protocols based on message passing. Section 5 provides an experimental evaluation of FastCAT for accuracy verification and efficiency comparison. Section 6 concludes the article.

2. Related works

This paper is closely related to the fields of data plane verification, control plane verification, and network configuration synthesis. These areas of research share a common purpose with the present study, as they all aim to improve network availability. The findings and advancements made in data plane verification, control plane verification, and network configuration synthesis are valuable contributions towards achieving a more reliable and robust network infrastructure.

2.1. Data plane verification

Data plane verification verifies whether the forwarding behavior of packets matches the network design intent based on data plane information such as forwarding tables. Early data plane verification tools were only capable of offline verification. Representative works include ConfigChecker [3], Anteater [4], and HSA [5]. To improve the speed of verification, many scholars have launched research on real-time verification. Examples include VeriFlow [6], Atomic Predicates Verifier [7], NetPlumber [8], and AP classifier [9]. The above tools mainly focus on small and medium networks such as campus networks. To improve the scalability of the verification tools, Zeng et al. proposed Libra [10] based on distributed computing techniques and Jayaraman et al. proposed RCDC [11] based on network structure features. As some networks have complex and diverse network policies, to verify these policies, Lopes et al. proposed NOD [12] based on the logic programming language Datalog, Yang and Lam proposed APT [13] based on equivalence classes, and Zhang et al. proposed APKeep [14]. Since data plane verification needs to be done only after the network configurations are deployed, it can only detect errors but not avoid them.

2.2. Control plane verification

The control plane refers to the network program that generates the data plane by combining the network environment information and network topology. Control plane verification verifies whether the packet forwarding behavior under the data plane generated by the control plane meets the network intent. Early control plane verification tools were protocol specific, e.g., RCC [15] and C-BGP [16] for the BGP protocol and Fireman [17] for firewalls. Batfish [1] is the first control plane verification tool that is non-protocol-specific. Batfish is slow to verify due to its time-consuming simulation of the control plane. To address this issue, ARC [18], Tiramisu [19], and ERA [20] have been introduced to enhance the verification speed. However, ARC and Tiramisu have limited expressiveness. ERA is mainly concerned with reachability verification. The expressiveness and scalability of ERA still do not meet the needs of real networks.

Similar to data plane verification, control plane verification also faces the same challenges as large networks. To this end, Beckett et al. have proposed Minesweeper [21] based on SMT solving techniques, Bonsai [22] based on symmetric compression, and ShapeShifter [23] based on abstract interpretation. These tools improve the ability of control plane verification tools to detect large networks. However, these tools still have some limitations. For example, Minesweeper suffers from the state space explosion problem, Bonsai is only applicable to symmetric networks and ShapeShifter can only verify reachability. Plankton [24] proposed by Prabhu et al. uses model detection techniques to model and verify the control plane behavior of equivalence classes, resulting in a significant improvement in scalability and verification speed over existing tools. FastPlane [2] proposed by Lopes et al. exploits the monotonically decreasing BGP routing priority during the announcement process and uses Dijkstra's algorithm to quickly generate BGP routing tables. However, FastPlane only supports monotonic networks.

In most cases, updated configurations are modified from previous configurations. Therefore, many scholars have proposed verification tools based on the idea of incremental computing. Examples include RealConfig [25], NUV [26], and DNA [27]. However, the above methods cannot verify all network attributes. For those that cannot be verified, network operators need further analysis based on routing tables.

2.3. Network configuration synthesis

Network configuration synthesis aims to automatically generate the correct network configurations according to the network operator's intentions. It is also intended to avoid human-induced configuration errors. The Propane [28] based on a graph algorithm automatically converts user-specified specifications into router-level BGP configurations using a valid intermediate representation. Propane/AT [29] is an improved version of Propane with a greater speed of implementation, but both only support BGP protocol. SyNET [30] uses a formal reasoning approach to transform configuration-solving problems into problems with Datalog synthetic input, but Datalog lacks the flexibility to change according to the changes in network topology and requirements. SyNET supports static routes, OSPF and BGP. Netcomplete [31] reduces the configuration generation problem to a constraint satisfaction problem solved with an SMT solver, adding a configuration sketch, BGP sketch, and a counter-example guided inductive synthesis algorithm to speed up the solution based on SMT constraint. Netcomplete only supports OSPF, BGP, and static routes, but it is easier to use than SyNET. Other comprehensive tools based on SAT/SMT include JINJING [32] and AED [33]. However, the above tools support fewer protocols and cannot satisfy complex routing strategies.

3. The overall framework of FastCAT

To solve the problems of the existing routing table calculation tools such as lack of user-friendliness and limited expressiveness, we propose a framework for computing routing tables incorporating multiple protocols named FastCAT, which is shown in Figure 1.

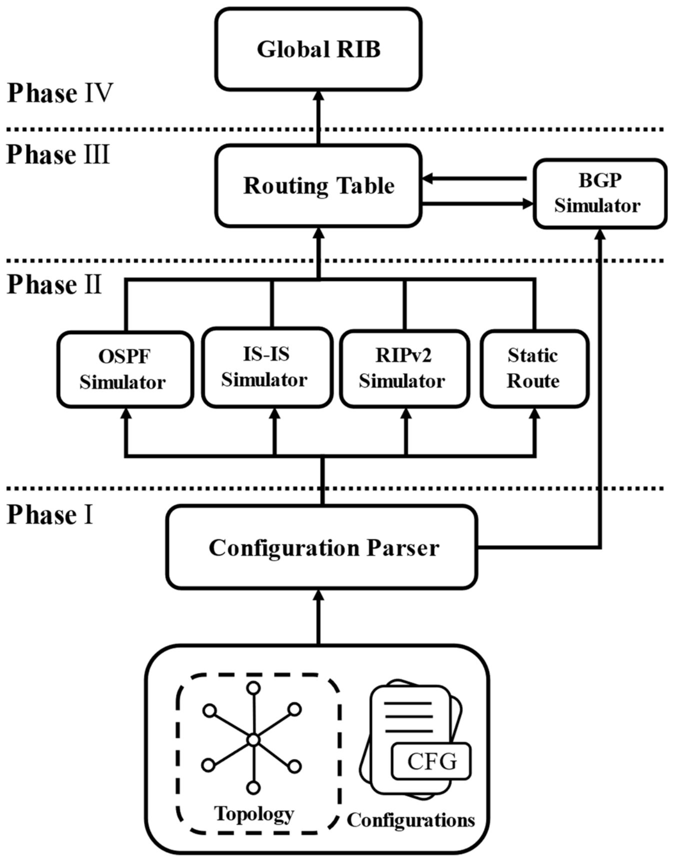

Figure 1. The framework of FastCAT.

FastCAT generates the routing table in four main phases:

1) Data acquisition. In phase 1 FastCAT reads the configuration files and topology information as intermediate data structures that are independent of device vendors. The network topology information enhances the analysis and simulation ability of FastCAT. Considering that some services cannot provide topology information, it is optional in this phase.

2) Internal gateway protocol simulation. FastCAT simulates the control plane of different routing protocols and generates the different protocol routing tables. Then, it loads the best routes and static routes from each protocol routing table into the global routing table. To ensure the accuracy of the routing table, the simulator of each protocol constructs a message propagation model based on the interaction mechanism of the protocol. The message propagation model simulates a complex set of network behaviors such as the propagation of link-state packets and route announcements in a real network by passing variables.

3) Border gateway protocol simulation. FastCAT simulates the control plane of the BGP protocol and generates the BGP protocol routing tables. According to the propagation of route advertisements and preference rules, the simulation module alternates messages among the simulated devices in a cyclic manner between BGP peers, until all simulated devices reach a stable state. Only valid BGP routes can participate in the preference and it means that their next hop is recursively reachable in the current global routing table. Therefore, the BGP simulator needs to interact with the current global routing tables when calculating the BGP routing tables.

4) Global routing table generation. FastCAT loads the best routes from the BGP protocol

routing tables into the final global routing tables. During the loading of best routes, if there are routes with the same destination prefix generated by different protocols, the routes generated by the lowest priority routing protocol are preferred. It should be stressed that the form of the routing table varies from the vendor, but regardless of the vendor, each table entry in the routing table has at least the destination prefix, the next hop, and the protocol that generated the route. If there are multiple best routes are generated for the same destination prefix, they are all loaded into the global routing table.

In summary, phases 2−4 constitute an iterative process of simulation, computation, and selection. Existing routing simulation tools use logical reasoning based on derived logic models or exploit the monotonicity of data center routing policy to calculate. FastCAT is more comprehensible and applicable than the above tools.

4. Fast simulation algorithms of routing protocols based on message passing

This section describes the algorithms for phases 2−4 in FastCAT, including the OSPF, IS-IS, RIPv2, and BGP simulation algorithms based on the message passing and the global routing table generation algorithm.

4.1. OSPF protocol simulation algorithm

OSPF is a typical link-state routing protocol. It is widely used in various medium and large networks because of fast convergence, dynamic sensing, and stable operation capabilities. After convergence of the network running the OSPF protocol, each router in regular areas of the network maintains an identical link state database (LSDB). Each participating router describes the topology of the autonomous system (AS) through the LSDB and calculates the routes using Dijkstra's algorithm based on LSDB.

To reduce the LSDB size in an area, OSPF usually divides the AS into a number of areas. Each individual area maintains a separate LSDB and communication between the routers in different areas is through the backbone area (Area0). In addition, a router, which is called AS boundary router (ASBR), locates at the borders of the autonomous systems (ASes) and connects to other non-OSPF networks. A router, which is called area border router (ABR), locates at the borders of backbone areas and connects to other non-backbone areas.

During the synchronization of LSDB, routers flood the link state update (LSU) packets in the network. The LSU packet usually carries multiple link state advertisements (LSAs). The contents of the LSAs contain the link state information that needs to exchange. In OSPF, many types of LSAs are defined according to different purposes and needs. It mainly includes Router LSA, Network LSA, Network Summary LSA, ASBR Summary LSA, AS External LSA, and Not-So-Stubby Area (NSSA) LSA [34]. The main contents of each LSA are listed as follows:

1) Router LSA. The Router LSA is generated by each router running the OSPF protocol. It describes the IP address and cost value of the router's interfaces.

2) Network LSA. The Network LSA is generated by the designated router (DR) in the multiple-access network. It describes the IP address and network mask of the multiple-access network. DR is a selection mechanism of OSPF, which is designed to save network bandwidth resources.

3) Network Summary LSA. The Network Summary LSA is generated by ABR. It describes the information of all Router LSAs and Network LSAs in the area. By flooding this LSA in each area,

ABR can solve the routing problem between different areas.

4) ASBR Summary LSA. The ASBR Summary LSA is generated by the ABR. By flooding this type of LSA, the ABR allows routers in the OSPF network to know the route to the ASBR.

5) AS External LSA. The AS External LSA is generated by ASBR. It describes the AS external routing information redistributed by ASBR. With the help of this LSA, routers in the OSPF network can calculate a route to a destination in another AS.

6) NSSA LSA. To reduce the size of the routing table, OSPF has designated several special areas, and one of them is the NSSA. NSSA prohibits AS External LSA from flooding in the area but allows the redistribution of external routes. The redistributed routes are flooded in the local area in the form of NSSA LSA and the ABR of the area converts them to AS External LSAs and floods in other areas.

In summary, computing the OSPF routing tables mainly involves two important stages. One is to obtain the LSDBs quickly and accurately and the other is to use Dijkstra's algorithm to calculate routes based on the LSDBs and the OSPF network topology. Algorithm 1 shows the pseudo-code that simulates the OSPF flooding LSAs.

Algorithm 1: Simulating algorithm for building OSPF LSDBs in a network

```
1:  DataModel ← InitOSPFData(InterFormatConfig)
2:  OSPFLSDB ← InitNodesLSDB(DataModel.Nodes)
3:  for each node v ∈ DataModel.OSPFNodes do
4:      GenerateAndFloodRtrNetLSA(vOSPFLinks, OSPFLSDB)
5:  for each node v ∈ DataModel.ABRs do
6:      GenerateAndFloodNetSummaryLSA(vRtrNetLSA, OSPFLSDB)
7:  for each node v ∈ DataModel.NSSANodes do
8:      GenerateAndFloodNSSALSA(vOSPFLinks, OSPFLSDB)
9:  for each node v ∈ DataModel.ASBRs do
10:     vImportRoutes ← DataModel.ImportRoutes[v]
11:     GenerateASExternalLSA(vImportRoutes, OSPFLSDB)
12:     if v ∈ DataModel.ABRs then
13:         ConvertNSSALSAToASExternalLSA(vNSSALSA, OSPFLSDB)
14: for each node v ∈ DataModel.ASBRs do
15:     FloodASExternalLSA(v, OSPFLSDB)
```

To save time as much as possible, the above algorithm omits some unnecessary operations and simplifies some processes in the construction of OSPF LSDB. For example, Algorithm 1 merges the information of Router LSA and Network LSA and only one round of generation and flooding is performed (lines 3–4). The algorithm also omits the generation and flooding of the ASBR Summary LSA because the actual OSPF routing table does not contain the route to ASBR.

Algorithm 2: Computing OSPF routing tables for all nodes

```
1:  Function: ComputingRIBs(Nodes)
2:    prefixes ← Ø, RIBs ← {}
3:    for each node v ∈ Nodes do
4:      prefixes ← prefixes ∪ GetPrefixs(LSDBInfo[v])
5:    Graph ← BuildOSPFGraph(Nodes, Connection, prefixes)
6:    for each node v ∈ Nodes do
7:      CostTables ← Dijkstra(v, Graph)
8:      for each prefix d ∈ prefixes do
9:        (ShortestPaths, TotalCost) ← DFS(CostTables, d)
10:       for each path p ∈ ShortestPaths do
11:         RouteEntry ← GetNextHop(p, TotalCost, Connection)
12:         AddRouteEntry(RIBs, RouteEntry)
13:   return RIBs
14:
15: IntraAndInterRIBs ← ComputingRIBs(OSPFNodes)
16: ASExternalRIBs ← ComputingRIBs(ASBRs)
17: NSSARIBs ← ComputingRIBs(NSSANodes)
18: OSPFRIBs ← IntraAndInterRIBs ∪ ASExternalRIBs ∪ NSSARIBs
```

After all necessary LSA generation and flooding finishes, except routers in special areas only contain network information in their area, each router in regular areas maintains the same LSDB. Algorithm 2 shows the pseudo-code for computing the OSPF routing tables based on LSDBs.

The LSDBs of all OSPF routers are not the same because of the special areas and the way to calculate the cost of redistributed routes is different from regular routes. Hence, the different types of routes are calculated separately. Line 15 in Algorithm 2 shows that intra-area and inter-area routes are calculated based on Router LSAs, Network LSAs, and Network Summary LSAs. Redistributed routes are calculated based on AS External LSAs (line 16), and NSSA routes are calculated based on NSSA LSAs (line 17). Each round of route calculation requires a separate calculation topology and the calculation topology is generated based on relationships between nodes and destination prefixes (lines 3−5). Finally, the final OSPF routing tables are obtained by merging the routing tables from each calculation.

Algorithm 1 performs only one round of generation and flooding for all LSAs, and its complexity is $O(n*m)$, where n is the number of all nodes and m is the average number of links. The most time-consuming part of computing the routing tables in Algorithm 2 is generating the calculation topology and computing the CostTables for all nodes, whose complexity is $O(v + v(v + l)^2)$, where v and l are the numbers of nodes and links among the calculation topology, respectively. When calculating inter-area routes, intra-area routes, and external routes, v and l are the numbers of all nodes and links, respectively. When calculating NSSA routes, v and l are the numbers of nodes and links in all NSSA areas, respectively. Since OSPF generates a separate calculation topology for different route types, the complexity of computing all OSPF routes is $O(k*n(n+e)^2 + k*n)$, where k is the number of route types and e is the number of all links. Combining Algorithm 1 and Algorithm 2 yields a total complexity of $O(k*n(n+e)^2 + k*n + n*m)$.

4.2. IS-IS protocol simulation algorithm

IS-IS is also a link-state routing protocol, which is widely used in large Internet service providers and data center networks. To support the management of large-scale networks, IS-IS usually divides the AS into several Level-1 areas and Level-2 areas. A Level-1 area consists of a series of contiguous Level-1 routers and Level-1-2 routers that belong to the same area and are responsible for intra-domain routing. A Level-2 area consists of a series of contiguous Level-2 routers and Level-1-2 routers that belong to the same area and are responsible for inter-domain routing. The neighbor relationships among these three types of routers are introduced in the following.

1) Level-1 router. A Level-1 router can establish a neighbor relationship with a Level-1 router or a Level-1-2 router in the same area and only maintains the LSDB of Level-1 in the area.

2) Level-2 router. Different from Level-1 routers, a Level-2 router can establish a neighbor relationship with a Level-2 router or a Level-1-2 router in any area and maintains the LSDB of Level-2.

3) Level-1-2 router. A Level-1-2 router can establish neighbor relationships with a Level-1 router in the same area or with a Level-2 router or a Level-1-2 router in any area. As a bridge between the Level-1 area and other areas, a Level-1-2 router needs to maintain the LSDB of Level-2 and the LSDB of the Level-1 area, which is similar to ABR in OSPF.

IS-IS uses the link state packet (LSP) to describe the link-state information. The LSP is divided into Level-1 LSP and Level-2 LSP, both of which have the same packet format. Level-1 LSP, which is generated by Level-1 routers, is only flooded in the Level-1 area. Level-2 LSP, which is generated by Level-2 routers, is only flooded in the Level-2 area. The Level-1-2 routers convert the Level-1 LSPs into Level-2 LSPs and flood them in Level-2 areas by default, but not vice versa. [35].

In summary, computing the IS-IS routing tables includes two key stages. First, construct the LSDBs of each node. Second, use Dijkstra's algorithm to calculate routes based on the LSDBs and the IS-IS network topology. Algorithm 3 shows the pseudo-code that simulates the IS-IS flooding LSPs.

Algorithm 3: Simulating algorithm for building IS-IS LSDBs in a network

```
1:   DataModel ← InitIS-ISData(InterFormatConfig)
2:   IS-ISLSDB ← InitNodesLSDB(DataModel.Nodes)
3:   for each node v ∈ DataModel.Level-1Nodes do
4:       GenerateAndFloodLevel-1LSP(vIS-ISLinks, IS-ISLSDB)
5:   for each node v ∈ DataModel.Level-2Nodes do
6:       GenerateLevel-2LSP(vIS-ISLinks, IS-ISLSDB)
7:   for each node v ∈ DataModel.Level-1-2Nodes do
8:       ConvertLevel-1LSPToLevel-2LSP(vLevel-1LSPs, IS-ISLSDB)
9:   Nodes ← DataModel.Level-1-2Nodes ∪ DataModel.Level-2Nodes
10:  for each node v ∈ Nodes do
11:      FloodLevel-2LSP(v, DataModel.Connection, IS-ISLSDB)
```

The protocol mechanism of IS-IS is simpler than OSPF. Lines 5 and 6 in the algorithm indicate that Level-2 LSPs are not flooded directly after they are generated by the Level-2 routers. The Level-2 LSPs are flooded uniformly after the Level-1-2 routers convert the corresponding Level-1 LSPs into Level-2 LSPs (lines 10−11). After the generation and flooding of LSPs, all IS-IS routers will reach a converged and stable state. The Level-2 and Level-1-2 routers will have intra-domain and inter-domain routing information of the network and the Level-1 routers will only have intra-domain routing information. Algorithm 4 shows the pseudo-code for computing the IS-IS routing tables based on LSDBs.

Algorithm 4: Computing IS-IS routing tables for all nodes

```
1:   Function: ComputingRIBs(Graph, Nodes, prefixes)
2:     RIBs ← {}
3:     for each node v ∈ Nodes do
4:       CostTables ← Dijkstra(v, Graph)
5:       for each prefix d ∈ prefixes do
6:         (ShortestPaths, TotalCost) ← DFS(CostTables, d)
7:         for each path p ∈ ShortestPaths do
8:           RouteEntry ← GetNextHop(p, TotalCost, Connection)
9:           AddRouteEntry(RIBs, RouteEntry)
10:    return RIBs
11:
12:  Graph ← BuildIS-ISGraph(IS-ISNodes, Connection, prefixes)
13:  Level-1RIBs ← ComputingRIBs(Graph, Level-1Nodes, Level-1Prefixes)
14:  Level-2RIBs ← ComputingRIBs(Graph, Level-2Nodes, Level-2Prefixes)
15:  IS-ISRIBs ← Level-1RIBs ∪ Level-2RIBs
```

Unlike the way OSPF calculates routes, only one calculation topology is generated in the process of calculating routes in IS-IS. The calculation topology is generated based on relationships between nodes and destination prefixes (line 12). Then, Level-1 routes and Level-2 routes are calculated based on the calculation topology separately (lines 13−14). Since Level-1-2 routers maintain LSDBs for both types of areas, the above route calculation processes include Level-1-2 routers separately. Finally, the final IS-IS routing tables are obtained by merging the routing tables from each calculation.

Algorithm 3 performs only one round of generation and flooding for all LSPs and its complexity is $O(n*m)$, where n is the number of all nodes and m is the average number of links. The most time-consuming part of calculating the routing tables in Algorithm 4 is to calculate the CostTables of all nodes, whose complexity is $O(v(v+l)^2)$, where v and l are the numbers of nodes and link connections involved in the calculation, respectively. When calculating Level-1 routes, v and l are the numbers of all Level-1 nodes (including Level-1-2 nodes) and links, respectively. When calculating Level-2 routes, v and l are the numbers of all Level-2 nodes (including Level-1-2 nodes) and links, respectively. The complexity of the calculation topology generation is $O(n+e)$, where e is the number of all links. Combining Algorithm 3 and Algorithm 4 yields a total complexity of $O(n(n+e)^2 + n*m + n+e)$.

4.3. RIPv2 protocol simulation algorithm

RIPv2 is a typical distance vector routing protocol. It is the earliest and most widely used interior gateway protocol (IGP) because of its simple implementation mechanism. RIP uses the hop count as the metric of routing and the hop count refers to the number of hops required to reach the destination network. A route that has more than 15 hops is considered an invalid route, RIP prefers the route which has fewer the number of hops. The RIP router only exchanges update packets with its own directly connected routers.

In the initial phase, each router's routing table only contains connected routes. Once a router is enabled RIP protocol, it sends routing update packets through the corresponding interface. When receiving the update packets, the RIP router learns the routes that are better and not in its routing table. After the learning, the router will announce the routing updates. With the continuous repetition of the above process, each RIP router's routing table will become more complete. After a number of cycles, each router will have the complete routing information of the network.

RIP takes split horizon, poisoned reverse, and triggered updates to prevent loops from occurring. The split horizon means that the updated route will not be announced back to the router that it learns from. The poisoned reverse is the opposite of split horizon, allowing the updated route to be announced back to the router that it learns from, but the hop count of the route is set to 16. The triggered updates mean that when the network topology or the cost of the route changes, the router will announce the updated route immediately [36].

Algorithm 5: Computing RIP routing tables for all nodes

```
1:  DataModel ← InitRIPData(InterFormatConfig)
2:  for each node v ∈ DataModel.RIPNodes do
3:      vRIPLinks ← DataModel.RIPLinks[v]
4:      AnnounceOwnRoutingInfo(vRIPLinks)
5:  while not converged do
6:      for each node v ∈ DataModel.RIPNodes do
7:          ReceiveAnnouncements()
8:          flag ← False
9:          flag ← UpdateRoutingTable()
10:         if flag is True then
11:             AnnounceUpdateRoutingUsingSplitHorizon()
```

As shown in Algorithm 5, each router first announces the IP address information of interfaces (lines 2−4). Then, a loop of receiving updated packets, updating the routing table, and announcing updated routes starts (lines 5−11). The loop program does not stop until the RIP network converges. There is no updated packet at that time. Considering that the routing tables are under deterministic control planes, we omit the triggered update and use the split horizon to prevent forwarding loops.

The total complexity of Algorithm 5 depends on the number of each node's neighbors and the number of route propagation rounds. It can be initially qualitatively estimated as $O(n*m*r*t)$, where n is the number of all nodes, m is the average number of links, r is the average number of routes propagated by all nodes and t is the average number of route propagation rounds.

4.4. BGP protocol simulation algorithm

BGP is a standard external gateway protocol. It is widely used in large Internet service providers for exchanging reachability information among ASes. BGP Routers exchange routing information by establishing sessions, each of which is set up by a pair of routers over a TCP connection. Session types include the external border gateway protocol (EBGP) session and the internal border gateway protocol (IBGP) session. The border routers in different ASes use EBGP sessions to announce reachability information. The routers maintain IBGP sessions to exchange routing information with internal routers in the same AS.

The routes learned from an IBGP session cannot be re-announced in other IBGP sessions to prevent forwarding loops. In addition, BGP uses the AS_Path attribute to record the AS numbers of the EBGP sessions that the routes pass through. BGP routers drop the EBGP announcements that contain the local AS number in the AS_Path attribute to prevent forwarding loops.

When receiving multiple routing announcements to the same prefix, the BGP router will take a series of comparison rules to select the best route. Table 1 shows the detailed routing preference rules. The comparison starts from the first rule and if the BGP router can select the best route based on the current rule, it will not execute the next rules. Otherwise, it will continue to execute the next rules until the best route is selected. Note that some attributes in the table belong to Huawei's private attributes. If the current attribute does not exist, the BGP router will automatically skip to the next attribute for comparison [37]. In addition, the rules in Table 1 can be potentially applicable as a module in the consensus control [38,39].

Table 1. BGP routing preference rules.

Seq	Attribute	Rule
1	Preferred_Value	Prefer the route with the largest value of this attribute.
2	Local_Preference	Prefer the route with the largest value of this attribute.
3	Prefer local announced routes, followed by routes learned from BGP sessions.	
4	AS_Path	Prefer the route with the shortest value of this attribute.
5	Origin	The order of preference for this attribute value is IGP > EGP > Incomplete.
6	MED	Prefer the route with the smallest value of this attribute.
7	Prefer an updated route learned from the EBGP session.	
8	IGP cost	Prefer the route with the lowest IGP cost to the next hop.
9	Cluster_list	Prefer the route with the shortest value of this attribute.
10	Router-ID	Prefer the route with the smallest value of this attribute.
11	Peer-ID	Prefer the route with the smallest value of this attribute.

In summary, the focus of BGP is not on calculating routes, but on filtering and controlling them. In terms of route preference rules, we can consider BGP as an enhanced distance vector protocol. Therefore, there are many similarities between the BGP simulation algorithm and the RIP simulation algorithm.

As shown in Algorithm 6, before computing the BGP routing table, a global routing table in the current state needs to be generated based on the existing IGP routing tables (see lines 9−16 in Algorithm 7 for the implementation of the pseudo-code in line 2). Each BGP router announces its routing announcements (lines 3−5). Then, a loop of receiving route updates, determining route validity, selecting the best route, updating the routing table, and announcing the best route starts (lines 6−14). If the next hop of the learned route can be queried in the global routing table, the BGP router regards the route as a valid route. Otherwise, it will be set as an invalid route (line 9). Then, each router selects the best announcement according to rules 1−11 of Table 1 (line 11) and adds the best route to its BGP routing table (line 13). If the best route is learned from an EBGP session, the router will redistribute it to other EBGP and IBGP sessions. Otherwise, it is only redistributed to other EBGP sessions (line 14).

Algorithm 6: Computing BGP routing tables for all nodes

```
1:  DataModel ← InitBGPData(InterFormatConfig)
2:  GlobalRIBs ← LoadBestRoutes()
3:  for each node v ∈ DataModel.BGPNodes do
4:      vAnnouncements ← DataModel.BGPAnnouncements[v]
5:      Announce(vAnnouncements)
6:  while not converged do
7:      for each node v ∈ DataModel.BGPNodes do
8:          announcements ← ReceiveAnnouncements()
9:          ValidAnnouncements ← DetermineRouteValidity(GlobalRIBs[v],announcements)
10:         flag ← False
11:         flag ← SelectBestAnnouncement(ValidAnnouncements)
12:         if flag is True then
13:             UpdateRoutingTable()
14:             RedistributeBestAnnouncement()
```

Similar to Algorithm 5, the total complexity of Algorithm 6 depends on the number of EBGP and IBGP sessions and the number of route propagation rounds. It can be initially qualitatively estimated as *O(n*r*s*t)*, where *n* is the number of all nodes, *r* is the average number of routes propagated by all nodes, *s* is the average number of all BGP sessions and *t* is the average number of all route propagation rounds.

4.5. Global routing table generation algorithm

The forwarding table is the basis for the router forwarding packets. It is converted from the global routing table. The forwarding table and routing table are nearly identical in content, but the global routing table is more intuitive and easier to read than the forwarding table. Hence, network operators often use the global routing table to analyze the network. The global routing table consists of each routing protocol's best routes, as well as connected routes and static routes. The specific generation algorithm of the global routing table is as follows.

As shown in Algorithm 7, in the initial state, the global routing table of each router only contains connected routes (lines 7). Each router adds the best routes of enabled routing protocols to its global routing table (lines 9−18). If the prefix of the added route already exists in the global routing table, the router will compare the preferences of two protocols that produced the routes and prefer the route generated by the lower preference protocol. Otherwise, it will be added directly (lines 3−5).

The total complexity of Algorithm 7 is *O(n*p)*, where *n* is the number of all nodes and *p* is the maximum number of protocols running on each router.

FastCAT can accurately support BGP, commonly used IGPs, static routes, and route redistribution between connected routes and OSPF. Thus, FastCAT is generally universal, but it has some limitations. For instance, FastCAT cannot support MPLS, VPN-related protocols, and route redistribution between other protocols. Even so, FastCAT can meet the requirements of most scenarios.

Algorithm 7: Computing global routing tables in a network partition V

```
1:  Function: AddRouteEntry(BestRoutes, RIB)
2:     for each route r ∈ BestRoutes do
3:        if route is not in RIB then
4:           Add(route, RIB)
5:        else AddAfterComparison(route, RIB)
6:
7:  GlobalRIBs ← InitGlobalRIBs()
8:  for each node v ∈ V do
9:     if v is enabled OSPF then
10:       AddRouteEntry(GetBestRoutes(OSPFRIBs[v]), GlobalRIBs[v])
11:    if v is enabled IS-IS then
12:       AddRouteEntry(GetBestRoutes(IS-ISRIBs[v]), GlobalRIBs[v])
13:    if v is enabled RIPv2 then
14:       AddRouteEntry(GetBestRoutes(RIPv2RIBs[v]), GlobalRIBs[v])
15:    if v is enabled Static then
16:       AddRouteEntry(StaticRIBs[v], GlobalRIBs[v])
17:    if v is enabled BGP then
18:       AddRouteEntry(GetBestRoutes(BGPRIBs[v]), GlobalRIBs[v])
```

5. Experiments

5.1. Experimental design

We implemented the functional modules of FastCAT in Python. The configuration file analysis module currently supports the configuration languages of Cisco IOS and Huawei IOS, which are the most commonly used. The control plane simulation module currently supports OSPF, IS-IS, RIPv2, BGP, and static routes. Except for OSPF, which supports connected route redistribution, other protocols do not support route redistribution. The existing routing table calculation tools cannot support all route redistribution scenarios. In addition, FastCAT is designed in modular form. For the protocols that are not supported, they can be added flexibly in the future on demand without any modification to the overall framework.

We measure two indicators to evaluate FastCAT: the routing table generation accuracy and the efficiency in different routing protocol usage scenarios. To this end, we designed two evaluation experiments, including accuracy verification and efficiency comparison.

The accuracy verification framework is shown in Figure 2. The experiment mainly includes three stages. In the first stage, an experienced network operator carries out network planning according to the existing topology, and then builds and configures the simulation environment of Cisco and Huawei based on GNS3 [40] and eNSP [41] respectively. In the second stage, the configuration files in the simulation environment are exported and input into FastCAT, and the routing tables of all devices in the Cisco and Huawei environment are calculated by FastCAT. In the third stage, the verification program first downloads the real routing tables in GNS3 and eNSP through Telnet. Then, the program compares the real routing table with the routing table calculated by FastCAT and generates difference reports. Finally, the network operator makes further analysis according to the reports.

The accuracy verification experiment was executed on a desktop computer running the Windows 10 operating system and equipped with an 8-core, 3.0-GHz Intel i7 processor and 24 G memory.

The efficiency comparison experiment includes single routing protocol efficiency comparison and multiple routing protocol efficiency comparison. Batfish is currently the only open-source routing table computation tool that supports general networks. It has higher accuracy and comprehensiveness than other frameworks. Therefore, in the single routing protocol efficiency comparison experiment, FastCAT is only compared with Batfish revision d1e1274 from 8/Sep/2022. This experiment focuses on the speed of generating routing tables based on different routing protocols at different network sizes.

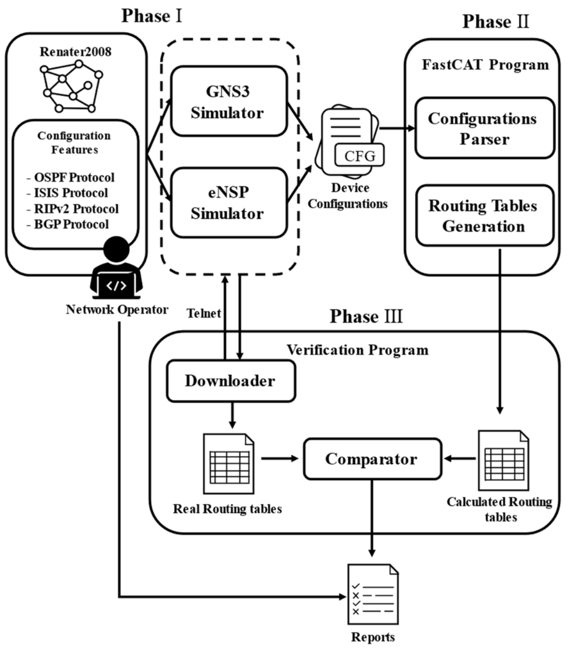

Figure 2. The framework of accuracy verification.

In the efficiency comparison experiment of multiple routing protocols, we further analyze the efficiency of FastCAT by comparing the speed of generating routing tables under different routing protocol usage scenarios.

In the efficiency comparison experiment, FastCAT and Batfish were both run in a virtual machine running the Ubuntu 22.10 operating system and equipped with an 8-core, 3.0-GHz Intel i7 processor and 16 G memory.

5.2. Datasets

To effectively evaluate FastCAT, we conduct the accuracy verification experiment using a small network named Renater2008 in the Topology Zoo [42]. We use the method of expert configuration to manually configure OSPF, IS-IS, RIPv2, BGP, and static routes based on GNS3 and eNSP, respectively. Table 2 shows the topology properties and routing features used by the network.

Table 2. Topology properties and routing features.

Topology	Routers	Links	Routing Features OSPF	IS-IS	RIPv2	BGP
Renater2008	33	43	Multi-areas including NSSA	Multi-areas	Version 2	EBGP

In the single routing protocol efficiency comparison experiment, we use the Ring networks (ring50−ring200) from the benchmark network configurations [43] as the comparison data for the BGP protocol. Since there are few public network configurations including IGPs and the existing network configuration synthesis tool NetComplete [31] only supports OSPF, BGP, and static routes. We select 10 network topologies (small, medium, and large) from the Topology Zoo that we classify according to their size. We use automatic synthesis to generate single IGP configurations based on these topologies. Table 3 shows the number of devices and links in each topology.

Table 3. The properties of selected topologies.

Name / Attrs	Abilene	Belnet2009	Canerie	Cernet	Cesnet 201006	Sinet	Deltacom	TataNld	UsCarrier	Cogentco
Routers	11	21	32	41	52	74	113	145	158	197
Links	14	31	41	59	63	76	183	194	189	245

Table 4. The combination mode and percentage settings of different routing protocols.

Protocol num	Mode	Ratio $(p\_a, p\_b, \ldots, p\_d) \mid (x\%, y\%, \ldots, z\%) \rightarrow p\_a{:}\ x\%, p\_b{:}\ y\%, \ldots, p\_d{:}\ z\%$				
2	combination	(0, 100), (10, 90), (20, 80), …, (100, 0)				
3	permutation	(34,33,33)	(50,10,40)	(50,20,30)	(50,30,20)	(50,40,10)
		(60,10,30)	(60,30,10)	(80,10,10)		
4	permutation	(25,25,25,25)	(50,10,10,30)	(50,10,30,10)	(50,30,10,10)	(60,10,10,20)
		(60,10,20,10)	(60,20,10,10)	(80,5,5,10)	(80,5,10,5)	(80,10,5,5)

(The above values are in percentage form and the percentage sign is omitted.)

In the efficiency comparison experiment of multiple routing protocols, we set up different routing protocol combinations and the percentage of different routing protocols under each combination based on the Cogentco topology. We use automatic synthesis to generate the corresponding configurations. Table 4 shows the specific routing protocol combination and percentage settings.

On the one hand, RIP is only widely used in small and medium networks. On the other hand, Batfish prompts a “REQUEUEFAILURE” exception message when analyzing more than 113 RIP configurations. Therefore, in the single routing protocol efficiency comparison experiment, we only use the configurations of small and medium networks to compare the routing table generation speed of the RIPv2 protocol. In the efficiency comparison experiment of multiple routing protocols, the RIPv2 is not set to a large percentage in the permutation-based combination modes.

5.3. Accuracy verification

As shown in Table 5, FastCAT refers to the format of the Huawei routing table and generates additional broadcast routes and IP address routes for each interface. Therefore, the route entries calculated based on the Cisco configurations are more than the real route entries in the GNS3. Moreover, because Cisco and Huawei have different preference settings for routing protocols, the nodes running the same protocols will load different routing tables based on different vendors' preference criteria. For instance, router R20 is running OSPF, IS-IS, and BGP. For Cisco devices, EBGP's preference is 20 and IS-IS's preference is 115. For multiple routes to the same destination prefix, R20 will prefer the routes learned by EBGP. For Huawei devices, IS-IS has a lower preference than EBGP. Hence, R20 will prefer the routes learned by IS-IS. Due to the load balance of IS-IS routes on the R20, the route entries that are calculated based on the Huawei configurations are more than those based on the Cisco configurations.

Table 5. The number of routing entries on each node.

RIBs	R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
RealRIBs (GNS3)	34	31	7	31	30	11	37	29	11	28	37
RealRIBs (eNSP)	44	39	15	39	38	19	47	39	21	34	45
CalRIBs (FastCAT)	**44**	**39**	16	**39**	**38**	**19**	**47**	**39**	**21**	**34**	**45**
RIBs	R11	R12	R13	R14	R15	R16	R17	R18	R19	R20	R21
RealRIBs (GNS3)	28	28	40	28	28	29	30	29	11	47	47
RealRIBs (eNSP)	34	34	50	34	34	41	38	37	19	71	73
CalRIBs (FastCAT)	**34**	**34**	**50**	**34**	**34**	**41**	**38**	**37**	**19**	**69/71**	74
RIBs	R22	R23	R24	R25	R26	R27	R28	R29	R30	R31	R32
RealRIBs (GNS3)	28	8	44	7	7	10	29	29	9	9	8
RealRIBs (eNSP)	34	16	58	15	15	18	37	37	16	17	16
CalRIBs (FastCAT)	**34**	17	58/59	**15**	**15**	**18**	**37**	**37**	17	**17**	17

(The representation of CalRIBs value includes A and A/B. A means that the value of the route entries calculated based on the Cisco and Huawei configurations is A. A/B means that the value of the route entries calculated based on the Cisco configurations is A and the value of the route entries calculated based on the Huawei configurations is B.)

The network operator finds that the calculated routing tables in bolded nodes are consistent with the real routing tables in both simulators by analyzing the comparison reports. We also use Batfish to generate the routing tables based on the Cisco configurations of the above experimental topology. The network operator finds that the cost of each RIP route generated by Batfish is always one more than the cost of the real RIP route and a few OSPF inter-area routes have the wrong next hop.

The main problem of FastCAT discovered from the differences on non-bolded nodes is: FastCAT generates an additional load-balance route for each router in a loop with an odd number of RIP routers. There are two main reasons for the above problem. One is the anti-loop mechanism of RIP. One is that the network device sends packets randomly. The control plane of RIP under the loop has multiple converged data planes for the above reasons. The final steady-state data plane is uncertain. However, FastCAT can only obtain one data plane of these. This problem also exists in Batfish.

In summary, FastCAT is very easy for average network operators to use. The network operators can provide FastCAT with network configurations to obtain relatively accurate routing tables without executing any query statements.

5.4. Efficiency comparison

We conduct these comparison experiments using four groups dataset that incorporates OSPF, IS-IS, RIPv2, and BGP. Each group dataset is grouped into six buckets, according to the network size. For each bucket, we show the CPU time taken to compute the routing tables by Batfish and FastCAT in that bucket.

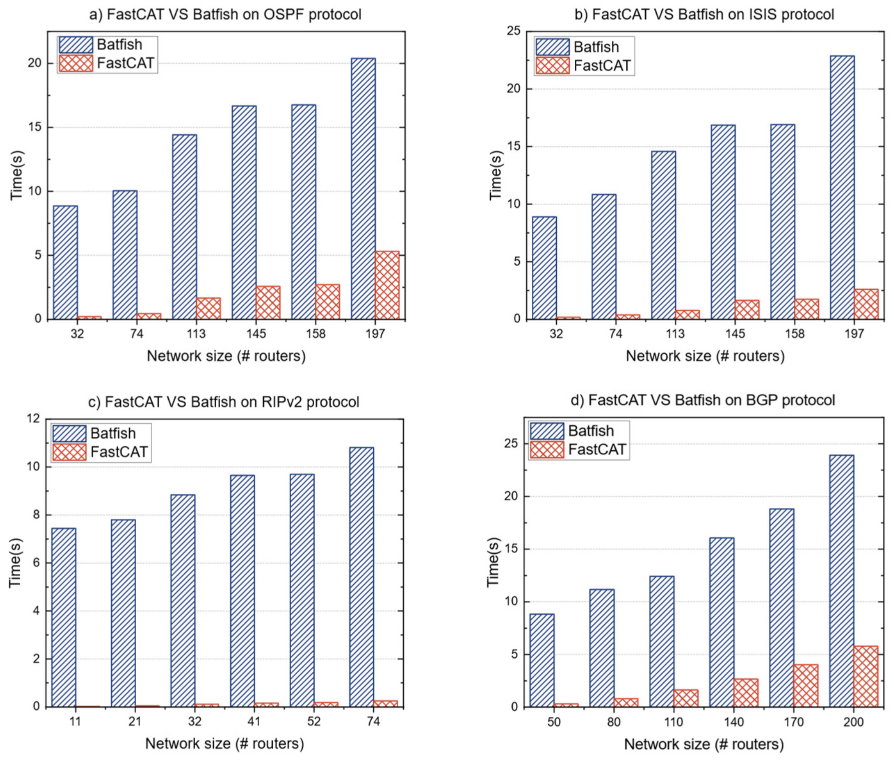

Figure 3. CPU time to compute the routing tables by Batfish and FastCAT.

Figure 3 shows that FastCAT is significantly faster than Batfish, regardless of IGPs or BGP. The main reason why FastCAT is faster than Batfish is as follows.

FastCAT focuses only on the final stable state of the OSPF and IS-IS protocols, disregarding the transient states during protocol convergence. For RIPv2 and BGP, FastCAT computes the current protocol routing tables based on the protocol's previous state, retaining only the most recent protocol routing tables in the latest state. In this experiment, we compared FastCAT with the latest version of Batfish, which employs a specific domain algorithm instead of data plane generation based on Datalog, resulting in significant efficiency improvements compared to its previous versions. However, during the routing table calculation process, Batfish utilizes the previous routing table of the node and its neighbors' routing tables to compute the current routing table of the node. Consequently, Batfish stores two routing tables for each node at a time: the previous state routing table and the current state routing table [2].

Next, we analyze the degree of impact of different routing protocols on the efficiency of FastCAT. We conduct these experiments using diverse combinations of routing protocols that are shown in Table 4.

Based on the previous theoretical introduction, we know that the above four protocols can be roughly divided into two categories. One is the link-state routing protocol which includes OSPF and IS-IS. One is the distance vector routing protocol which includes RIPv2 and BGP. From Figure 4(a), we can see that the trend first falls and then rises in the case of two link-state routing protocol combinations. The trend is upward all the time in other cases of two routing protocol combinations. The main reasons are as follows.

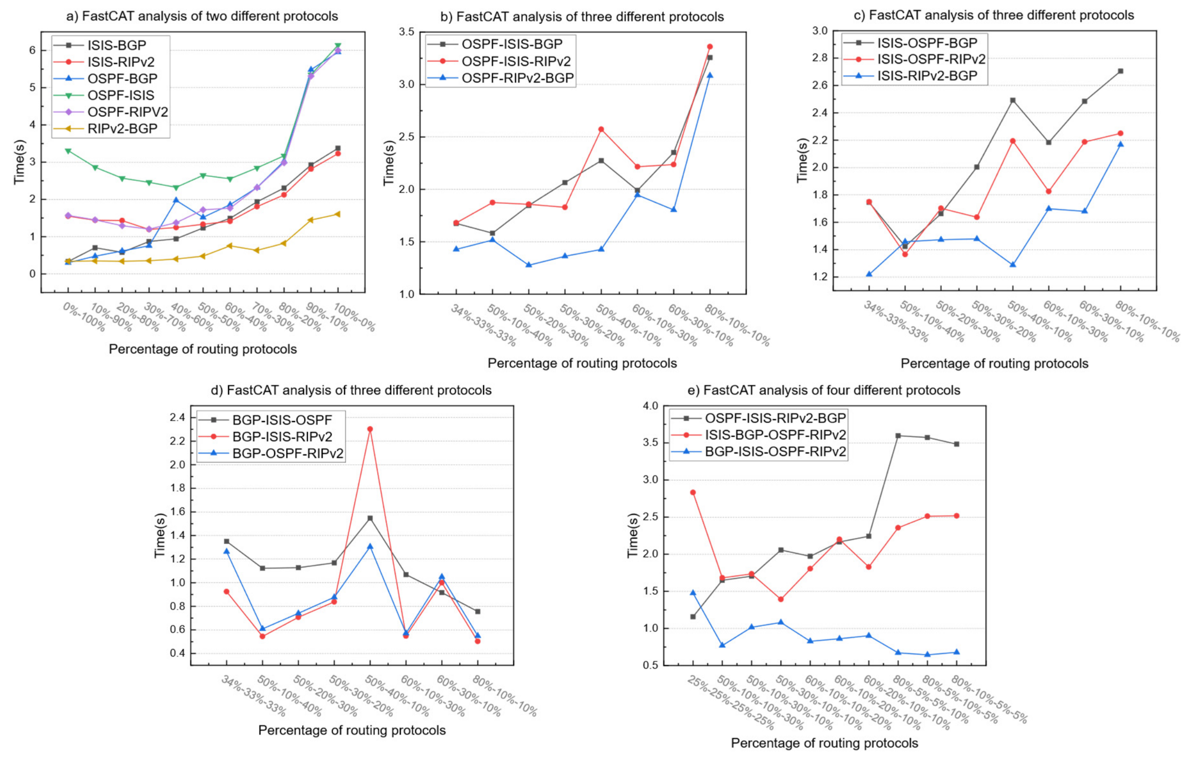

Figure 4. Comparison of FastCAT efficiency in multiple routing protocol combination scenarios.

Based on the previous analysis of algorithms' complexity, we can know that the time to calculate the routing tables for link-state routing protocols depends on the size of the LSDBs. The time to calculate the routing tables for distance vector routing protocols relies on the number of neighbors and route propagation rounds. The routes of different routing protocols are isolated from each other. Therefore, for the combinations of OSPF and IS-IS, when the ratio of OSPF gradually increases, the

LSDBs of IS-IS will gradually decrease. Then, the overall time to compute will decrease. Once the combination ratio of both exceeds the critical value of 40%−60%, the overall time will increase.

In addition, the configurations of RIPv2 and BGP randomly generated based on the Cogentco have simple neighbor relationships and fewer route propagation rounds. Hence, for the combinations of IS-IS and BGP, IS-IS and RIPv2, OSPF and BGP, and OSPF and RIPv2, the overall time always increases. Compared with RIPv2, BGP only announces the loopback address routes, the number of which is fewer than the interconnection interfaces. So, for BGP, there are fewer routing announcements to be announced. This is the reason why the curve of RIPv2-BGP has an increasing trend.

Next, with Figure 4(b)−(e), we can see that the overall time generally tends to increase when OSPF or IS-IS is the main component of combinations. The overall time generally tends to decrease when BGP is the main component of combinations. Finally, by further analyzing the local trends of the curves and the percentages of the remaining protocols, we conclude that the efficiency of FastCAT is affected by each routing protocol in descending order of OSPF, IS-IS, RIPv2, and BGP.

Note that the speed of routing table generation is not only related to the network size but is also affected by the complexity of the network topology and the number of destination prefixes. The network topologies of the experiments in this section represent general networks [40]. Hence, for the sake of rigor, all the above conclusions need to be limited to a general network with about 200 routers.

6. Conclusions

In this paper, we propose a routing table calculation framework called FastCAT which incorporates multiple protocols and fast simulation algorithms for routing protocols based on message passing, to address the shortcomings of current routing table calculation tools. FastCAT can simulate routing protocol interaction behaviors based on configuration files and topology information and generate routing tables quickly. Evaluations show that FastCAT can generate routing tables faster and more accurately than leading-edge methods in a general network with about 200 routers.

In addition to the routing protocols mentioned in this paper, actual networks use other complex and diverse protocols, such as MPLS, VRRP, etc. The existing routing table calculation tools including FastCAT do not support these protocols. In addition, route redistribution is an important means for network operators to solve the problem of route interaction. However, FastCAT currently only supports connected route redistribution in the OSPF protocol and does not support route redistribution between other types of protocols. Nonetheless, FastCAT currently supports commonly used routing protocols and can meet most requirements of network operators.

FastCAT is designed in modular form. For the protocols that are not supported, they can be added flexibly in the future on demand without any modification to the overall framework. Further research is needed to extend FastCAT to support additional protocol features.

Use of AI tools declaration

The authors declare they have not used Artificial Intelligence (AI) tools in the creation of this article.

Conflict of interest

The authors declare there is no conflict of interest.

References

1. A. Fogel, S. Fung, L. Pedrosa, M. Walraed-Sullivan, R. Govindan, R. Mahajan, et al., A general approach to network configuration analysis, in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, (2015), 469−483. Available from: http://web.cs.ucla.edu/~todd/research/nsdi15_batfish.pdf.
2. N. P. Lopes, A. Rybalchenko, Fast BGP simulation of large datacenters, in *Verification, Model Checking, and Abstract Interpretation*, **11388** (2019), 386−408. https://doi.org/10.1007/978-3-030-11245-5_18
3. E. Al-Shaer, W. Marrero, A. El-Atawy, K. Elbadawi, Network configuration in a box: towards end-to-end verification of network reachability and security, in *2009 17th IEEE International Conference on Network Protocols*, (2009), 123−132. https://doi.org/10.1109/ICNP.2009.5339690
4. H. Mai, A. Khurshid, R. Agarwal, M. Caesar, P. B. Godfrey, S. T. King, Debugging the data plane with anteater, *ACM SIGCOMM Comput. Commun. Rev.*, **41** (2011), 290−301. https://doi.org/10.1145/2043164.2018470
5. P. Kazemian, G., Varghese, N. McKeown, Header space analysis: static checking for networks, in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, (2012), 113−126.
6. A. Khurshid, W. Zhou, M. Caesar, P. B. Godfrey, Veriflow: verifying network-wide invariants in real time, in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, (2012), 49−54. https://doi.org/10.1145/2342441.2342452
7. H. Yang, S. S. Lam, Real-time verification of network properties using atomic predicates, *IEEE/ACM Trans. Networking*, **24** (2015), 887−900. https://doi.org/10.1109/TNET.2015.2398197
8. P. Kazemian, M. Chang, H. Zeng, G. Varghese, N. McKeown, S. Whyte, Real time network policy checking using header space analysis, in *Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation*, (2013), 99−112.
9. H. Wang, C. Qian, Y. Yu, H. Yang, S. S. Lam, Practical network-wide packet behavior identification by AP classifier, in *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, (2015), 1−13. https://doi.org/10.1145/2716281.2836095
10. H. Zeng, S. Zhang, F. Ye, V. Jeyakumar, M. Ju, J. Liu, et al., Libra: divide and conquer to verify forwarding tables in huge networks, in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, (2014), 87−99. Available from: https://www.usenix.org/system/files/conference/nsdi14/nsdi14-paper-zeng.pdf.
11. K. Jayaraman, N. Bjørner, J. Padhye, A. Agrawal, A. Bhargava, P. A. C. Bissonnette, et al., Validating datacenters at scale, in *Proceedings of the ACM Special Interest Group on Data Communication*, (2019), 200−213. https://doi.org/10.1145/3341302.3342094
12. N. P. Lopes, N. Bjørner, P. Godefroid, K. Jayaraman, G. Varghese, Checking beliefs in dynamic networks, in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, (2015), 499−512.
13. H. Yang, S. S. Lam, Scalable verification of networks with packet transformers using atomic predicates, *IEEE/ACM Trans. Networking*, **25** (2017), 2900−2915. https://doi.org/10.1109/TNET.2017.2720172

14. P. Zhang, X. Liu, H. Yang, N. Kang, Z. Gu, H. Li, APKeep: realtime Verification for Real Networks, in *Proceedings of the 17th Usenix Conference on Networked Systems Design and Implementation*, (2020), 241−255. Available from: https://www.usenix.org/system/files/nsdi20-paper-zhang-peng.pdf.
15. N. Feamster, H. Balakrishnan, Detecting BGP configuration faults with static analysis, in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, **2** (2005), 43−56. https://dl.acm.org/doi/10.5555/1251203.1251207
16. B. Quoitin, S. Uhlig, Modeling the routing of an autonomous system with C-BGP, *IEEE Network*, **19** (2005), 12−19. https://doi.org/10.1109/MNET.2005.1541716
17. L. Yuan, H. Chen, J. Mai, C. N. Chuah, Z. Su, P. Mohapatra, Fireman: a toolkit for firewall modeling and analysis, in *2006 IEEE Symposium on Security and Privacy (S&P'06)*, 2006. https://doi.org/10.1109/SP.2006.16
18. A. Gember-Jacobson, R. Viswanathan, A. Akella, R. Mahajan, Fast control plane analysis using an abstract representation, in *Proceedings of the 2016 ACM SIGCOMM Conference*, (2016), 300−313. https://doi.org/10.1145/2934872.2934876
19. A. Abhashkumar, A. Gember-Jacobson, A. Akella, Tiramisu: Fast multilayer network verification, in *17th USENIX Symposium on Networked Systems Design and Implementation*, (2020), 201−219. Available from: https://www.usenix.org/system/files/nsdi20-paper-abhashkumar.pdf.
20. S. K. Fayaz, T. Sharma, A. Fogel, R. Mahajan, T. D. Millstein, V. Sekar, et al., Efficient network reachability analysis using a succinct control plane representation, in *Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation*, **16** (2016), 217−232.
21. R. Beckett, A. Gupta, R. Mahajan, D. Walker, A general approach to network configuration verification, in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, (2017), 155−168. https://doi.org/10.1145/3098822.3098834
22. R. Beckett, A. Gupta, R. Mahajan, D. Walker, Control plane compression, in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, (2018), 476−489. https://doi.org/10.1145/3230543.3230583
23. R. Beckett, A. Gupta, R. Mahajan, D. Walker, Abstract interpretation of distributed network control planes, in *Proceedings of the ACM on Programming Languages*, **4** (2019), 1−27. https://doi.org/10.1145/3371110
24. S. Prabhu, K. Y. Chou, A. Kheradmand, B. Godfrey, M. Caesar, Plankton: scalable network configuration verification through model checking, in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, (2020), 953−967.
25. P. Zhang, Y. Huang, A. Gember-Jacobson, W. Shi, X. Liu, H. Yang, et al., Incremental network configuration verification, in *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*, (2020), 81−87. https://doi.org/10.1145/3422604.3425936
26. Y. Li, Z. Wang, X. Yin, X. Shi, J. Wu, F. Ye, et al., Assisting reachability verification of network configurations updates with NUV, *Comput. Networks*, **177** (2020), 107326. https://doi.org/10.1016/j.comnet.2020.107326
27. P. Zhang, A. Gember-Jacobson, Y. Zuo, Y. Huang, X. Liu, H. Li, Differential network analysis, in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022. Available from: https://www.usenix.org/conference/nsdi22/presentation/zhang-peng.

28. R. Beckett, R. Mahajan, T. Millstein, J. Padhye, D. Walker, Don't mind the gap: bridging network-wide objectives and device-level configurations, in *Proceedings of the 2016 ACM SIGCOMM Conference*, (2016), 328–341. https://doi.org/10.1145/2934872.2934909
29. R. Beckett, R. Mahajan, T. Millstein, J. Padhye, D. Walker, Network configuration synthesis with abstract topologies, in *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, (2017), 437–451. https://doi.org/10.1145/3062341.3062367
30. A. El-Hassany, P. Tsankov, L. Vanbever, M. Vechev, Network-wide configuration synthesis, in *Computer Aided Verification*, CAV 2017, Heidelberg, Germany, **10427** (2017), 261–281. https://doi.org/10.1007/978-3-319-63390-9_14
31. A. El-Hassany, P. Tsankov, L. Vanbever, M. Vechev, Netcomplete: practical network-wide configuration synthesis with autocompletion, in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, (2018), 579–594.
32. B. Tian, X. Zhang, E. Zhai, H. H. Liu, Q. Ye, C. Wang, et al., Safely and automatically updating in-network ACL configurations with intent language, in *Proceedings of the ACM Special Interest Group on Data Communication*, (2019), 214–226. https://doi.org/10.1145/3341302.3342088
33. A. Abhashkumar, A. Gember-Jacobson, A. Akella, Aed: incrementally synthesizing policy-compliant and manageable configurations, in *Proceedings of the 16th International Conference on Emerging Networking Experiments and Technologies*, (2020), 482–495. https://doi.org/10.1145/3386367.3431304
34. J. Moy, *RFC2328: OSPF Version 2*, 1998. https://doi.org/10.17487/rfc2328
35. R. Callon, *Use of OSI IS-IS for Routing in TCP/IP and Dual Environments (No. rfc1195)*, 1990. https://doi.org/10.17487/rfc1195
36. G. Malkin, *RIP Version 2 (No. rfc2453)*, 1998. https://doi.org/10.17487/rfc2453
37. Y. Rekhter, T. Li, S. Hares, *A Border Gateway Protocol 4 (BGP-4) (No. rfc4271)*, 2006. https://doi.org/10.17487/rfc4271
38. W. Abbas, A. Laszka, X. Koutsoukos, Improving network connectivity and robustness using trusted nodes with application to resilient consensus, *IEEE Trans. Control Network Syst.*, **5** (2018), 2036–2048. https://doi.org/10.1109/TCNS.2017.2782486
39. Y. Shang, Resilient tracking consensus over dynamic random graphs: a linear system approach, *Eur. J. Appl. Math.*, **34** (2023), 408–423. https://doi.org/10.1017/S0956792522000225
40. Graphical network simulator-3 (GNS3). Available from: https://www.gns3.com/.
41. *Enterprise Network Simulation Platform (eNSP)*. Available from: https://forum.huawei.com/enterprise/en/huawei-ensp.
42. S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, M. Roughan, The internet topology zoo, *IEEE J. Sel. Areas Commun.*, **29** (2011), 1765–1775. https://doi.org/10.1109/JSAC.2011.111002
43. Batfish. Available from: https://github.com/batfish/batfish.