Enabling Scalability and Flexibility into Network Routing Protocol using Behavior Tree

Jiaorui Huang, Chungang Yang, Tao Feng, Lujia Dong, Alagan Anpalagan, Qiang Ni, Mohsen Guizani

***Abstract*—Current network routing protocol design is faced with novel challenges due to evolving network scale, various network service demands, and dynamic network states. However, the conventional finite state machine models lack both scalability and flexibility for the description of network routing protocol states. In this article, we enable scalability and flexibility into network routing protocol by exploring and exploiting behavior trees, where behavior trees can reformulate the network routing protocol by characterizing state transformation as action nodes. We first present a generic routing protocol architecture with a comparative analysis of the behavior tree, finite state machine, etc. Then, we propose an implementable functional scheme, which provides a foundation for extending the functionality and enabling flexible configurations towards the network routing protocol. Finally, we design two use cases to verify that behavior trees can effectively replace finite state machines and the excellent scalability of behavior trees in terms of routing protocols.**

***Index Terms*—Behavior Tree, Control Plane Decoupling, Network Routing Protocol.**

I. Introduction

The diversity of service demands, the increasing scale of networks, and the dynamics of network situations involve novel scalability and flexibility challenges to network routing protocols [1]. On the one hand, most network routing protocols leverage finite state machine (FSM) to manage the state transitions in the network routing process [2], [3]. For instance, the border gateway protocol (BGP), as described in [2], the protocol operates with six states and experiences nineteen state transitions. The open shortest path first (OSPF), as discussed in [3], utilizes ten states to control packet flow. On the other hand, the FSM describes the work of a system by defining a finite number of states and transitions between states processes, which usually consist of state, transfer and event [4]. Adopting the FSM in routing protocols presents several challenges as networks scale, dynamics evolve, and service requirements diversify, which are given as follows.

- **Limited modularity and scalability**: The FSM is inherently linear and finite, requiring a redesign of the state transition when new states or behaviors are added, which makes scalability difficult.
- **Poor expression of complex behaviors**: The FSM is suitable for simple, sequential task processing, but it tends to become bloated and state explosion when dealing with complex behaviors that are difficult to manage.
- **Lack of parallel processing**: The FSM executes tasks in sequence order and is unable to handle multitasking or prioritize tasks simultaneously.
- **Limited fault handling and recovery**: The FSM requires the explicit design of additional error-handling states or fallback mechanisms when a state fails to complete a task, which increases design complexity.

Taking the BGP protocol as an example, it serves as a standardized protocol for exchanging routing information between the Internet and autonomous systems. This is accomplished by establishing adjacencies, advertising network prefixes, and dynamically adjusting the optimal path based on various policies [5]. Its FSM includes six states that manage the peer sessions, including *Idle*, *Connect*, *OpenSent*, *OpenConfirm*, *Established*, and *Active*, as shown in Fig. 1. However, as network scale and dynamics increase, the BGP exhibits critical vulnerabilities in route announcement verification and peer authentication [6]. To ensure secure communication between peers, a *Security* state can be introduced into the FSM. Before a peer transitions to the *Connect* state, it must first enter the *Security* state for identity authentication and certificate verification; only upon successful validation can a connection be established. Additionally, before the protocol moves to the *OpenConfirm* state, the security and integrity of the received Open message must be verified in the *Security* state. If any security issues arise in the *Established* state after the session has been established, it becomes necessary to access the *Security* state to refresh the security parameters. This analysis illustrates that introducing a single *Security* state requires seven additional state transitions. As the number of states increases, the complexity of state transition relationships escalates significantly, leading to state explosion. Moreover, in the BGP state transition process, subsequent steps are triggered only when the execution result of the preceding step satisfies the specified condition, preventing any synchronization of operations.

Behavior tree (BT) offers a modular and flexible alternative to the FSM [7]–[10]. The BT is a graphical mathematical

This work was supported in part by the National Key Research and Development Program of China (2020YFB1807700), the Specific Technology Program (JZX6Y202207010351), the Innovation Capability Support Program of Shaanxi (2024RS-CXTD-01), and the National Key Laboratory of Multi-domain Data Collaborative Processing and Control (MDPC20240401). Corresponding author Chungang Yang is with National Key Laboratory of Multi-domain Data Collaborative Processing and Control, Xian, 710068, China, and also with Hangzhou Institute of Technology of Xidian University, Hangzhou, 311231, China.

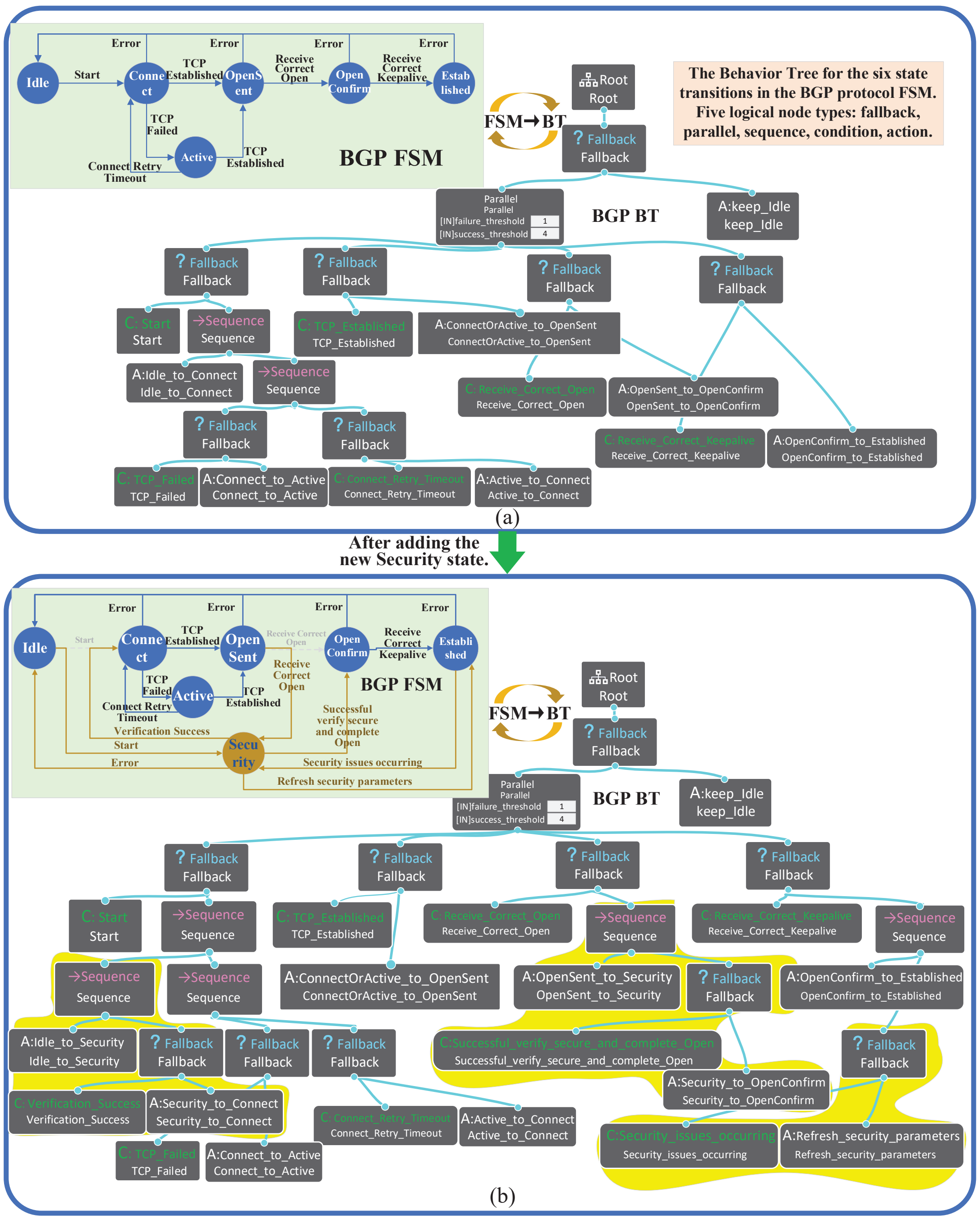
J. Huang, C. Yang and L. Dong are with the State Key Laboratory on Integrated Services Networks, Xidian University, Xi'an, 710071 China (emails: hjr6286@163.com, guideyang2050@163.com, jiatofight@163.com).

T. Feng is with Institute of System Engineering AMS PLA (email: feng09@163.com).

Q. Ni is with the School of Computing and Communications, Lancaster University, Lancaster LA1 4WA, U.K. (e-mail: q.ni@lancaster.ac.uk).

A. Anpalagan is with the Department of Electrical, Computer and Biomedical Engineering, Toronto Metropolitan University, Toronto, ON M5B 2K3, Canada (e-mail: alagan@ee.torontomu.ca).

M. Guizani is with the Machine Learning Department, Mohamed Bin Zayed University of Artificial Intelligence, Abu Dhabi, UAE (e-mail: mguizani@ieee.org).



BGP FSM
Idle
Start
Connect
TCP Established
OpenSent
Receive Correct Open
Open Confirm
Receive Correct Keepalive
Established
Error
TCP Failed
Active
TCP Established
Connect Retry Timeout
FSM→BT
Root
Fallback
Parallel
[IN]failure_threshold 1
[IN]success_threshold 4
BGP BT
A:keep_Idle
The Behavior Tree for the six state transitions in the BGP protocol FSM. Five logical node types: fallback, parallel, sequence, condition, action.
C: Start
Sequence
A:Idle_to_Connect
C: TCP_Established
A:ConnectOrActive_to_OpenSent
C: Receive_Correct_Open
A:OpenSent_to_OpenConfirm
C: Receive_Correct_Keepalive
A:OpenConfirm_to_Established
C: TCP_Failed
A:Connect_to_Active
C: Connect_Retry_Timeout
A:Active_to_Connect
(a)
After adding the new Security state.
Security
Verification Success
Successful verify secure and complete Open
Security issues occurring
Refresh security parameters
A:Idle_to_Security
C: Verification_Success
A:Security_to_Connect
A:OpenSent_to_Security
C:Successful_verify_secure_and_complete_Open
A:Security_to_OpenConfirm
C:Security_issues_occurring
A:Refresh_security_parameters
(b)

Fig. 1: Scalable and flexible network routing protocol using behavior tree for BGP protocol. (a) Transition between BGP internal finite state machine and behavior tree. (b) New finite state machine and behavior tree with added security state.

model designed to support reactive and fault-tolerant operations. It consists of five essential logical nodes: action, condition, fallback, sequence, and parallel [4]. The BT effectively addresses limitations in the FSM. First, the BT modularity results from its compositional structure, where each node functions as an independent, self-contained unit representing a distinct behavior or operation. This design allows nodes to be added, removed, or modified without disrupting the overall structure, making the BT highly adaptable and easy to extend or integrate. Second, the various logical nodes in the BT enable it to flexibly express complex logic without increasing system complexity, while parallel nodes support concurrent task execution. Finally, fallback nodes enhance fault tolerance by automatically selecting alternative paths when a task fails, providing greater flexibility in achieving objectives. The authors in [7] demonstrated that the BT was widely utilized in game development and robotics as a modular alternative to the FSM for managing complex decision-making processes while maintaining stability under high load. Drawing inspiration from this, we explore the potential of the BT in network systems, which often experience instability under heavy traffic. Accordingly, this article investigates the introduction of the BT into network routing protocols to address these challenges. Our previous study in [2] applied the BT to decouple processing logic from operational logic and action execution in routing protocols. However, that work was largely theoretical, lacking a practical implementation. To fill this gap, we propose a realizable functional scheme that establishes a foundation for extending functionality and enables flexible, dynamic configuration of network routing protocols. Additionally, the authors in [11] utilized the BT in the power systems domain to manage protocol switching, providing flexible control over the execution of operations. In contrast, our approach utilizes the BT to construct the process of the routing protocol itself, replacing the existing FSM mechanisms within the protocol's internal operations.

In this article, we explore and exploit the BT to achieve flexibility and scalability of network routing protocols with the following research contributions:

- **Generic network routing protocol architecture:** In this article, we replace the FSM in routing protocols with the BT and design a generic architecture based on the BT. Then, we compare and analyze this architecture in eight aspects with the FSM, hierarchical finite state machine (HFSM), and decision tree (DT).
- **Implementable functional scheme:** To bridge the gap between theory and implementation, this article proposes an implementable functional scheme based on the BT to decouple the control plane, enhancing the flexibility of protocol functionality extensions.
- **Proof-of-concept implementations:** This article presents two use cases. The first demonstrates the feasibility and advantages of replacing the FSM in a routing protocol with the BT. The second highlights the enhanced capabilities for extending the protocol through leveraging the BT.

The rest of this article is organized as follows. In Section II, we provide an overview of the concept of the BT and compare it with existing control architectures. In Section III, we focus on the application of the BT in routing protocols and design a generic network routing protocol architecture based on the BT. In Section IV, we present the feasibility analysis of leveraging the BT as an alternative for the FSM, along with a scalability analysis based on the BT through two use cases. Finally, conclusion is drawn in Section V.

II. Basics and Analysis of Behavior Trees for Network Routing Protocol Challenges

This section provides a brief overview of the BT and five types of logical nodes, leveraging the BGP protocol to illustrate the application of the BT in network routing protocols. Subsequently, we conduct a comparative analysis of the relationship between the BT and existing control architectures.

A. Basics and Logic Nodes of Behavior Tree

The BT is a widely applied graphical model for reactive and fault-tolerant task execution in gaming and robotics [4]. However, the application of the BT in routing protocols has not been fully explored. The modular structure of the BT enhances clarity and reduces design errors, with its strengths rooted in modularity, scalability, and reusability. The BT operates through five types of logical nodes: sequence, fallback, parallel, condition, and action. Among these, the sequence, fallback, and parallel nodes are composite nodes, each producing one of three possible outcomes: success, failure, or running. Child nodes that can be attached to a composite node include action, condition, and other composite nodes.

The five types of logical nodes as follows.

- **Fallback nodes**, also known as selectors, execute from left to right. If any child node succeeds, the following node stops execution and returns success. If all child nodes fail, the fallback node returns failure.
- **Sequence nodes** execute from left to right, returning success only if all child nodes succeed. If any node fails, the sequence node returns failure, and the remaining nodes are not executed.
- **Parallel nodes** determine their return status based on a predefined strategy, allowing threshold values to be set for the number of successes and failures. Specifically, if a parallel node has N child nodes, success and failure thresholds can be set at Q and P, respectively, where $P \leq Q \leq N$. The node returns success if at least Q child nodes succeed, and failure if at least P child nodes fail.
- **Action nodes** perform the necessary functions for the current network. They return success when the operation completes successfully, failure if it fails, or running when the operation is still in progress.
- **Condition nodes** serve the purpose of decision-making and return success if a given condition is met; otherwise, they return failure. These nodes do not influence the internal state of BT and are primarily intended to enhance its readability.

We utilize five types of logical nodes to implement the FSM to the BT conversion for the BGP protocol. The BT begins at the root node and initially enters a fallback node. When no event is triggered, the action keeps *Idle*. If an event is triggered, the process enters a parallel node consisting of four fallback nodes. These four fallback nodes are as follows. For the first fallback node, if a start event is triggered and then enters the sequence node. The first action in the sequence node is the transition to *Connect*. Subsequently, a sequence node with two fallback nodes is executed. Following a left-to-right sequence, the first fallback node within this sequence checks if the transmission control protocol (TCP) connection is successful. The action will transition to *Active* if the connection fails. The second fallback node within this sequence assesses if the retry connection has timed out. If it has, the action is to transition back to *Connect*. For the second fallback node, the system determines if the TCP connection has been established. If it has, the action is to transition to *OpenSent*. For the third fallback node, the system waits to receive an open message from a peer node after sending an open message in the current node. The action is to transition to *OpenConfirm* if received. For the fourth fallback node, the system waits for a keepalive message from a peer node. If received, the action is to transition to *Established*, where update messages are exchanged to learn route entries. Based on the threshold settings of the parallel nodes, if any of the four fallback nodes return failure, the entire parallel node immediately returns a failure to the connected fallback node, taking the action to

TABLE I: Comparative analysis of finite state machine, hierarchical finite state machine, decision tree and behavior tree.

Method / Attributes	FSM	HFSM	DT	BT
Modularity	Unit of state.	Unit of sub-state machine, low coupling.	Unit of decision result.	Unit of function subtree.
Hierarchy	Event-driven, one-way transfer	Event-driven, hierarchical execution, one-way transfer.	Recursive division, hierarchical executioon, tree visualization.	Tick-driven, hierarchical execution, bidirectional transfer
Readability	Simple state: Good Complex state: Fair	Fair	Good	Excellent
Reusability	Poor	Poor	Average	Excellent
Portability	Logic is easy, coding is complex.	Logic is easy, coding is complex.	Logic is easy, coding is moderate.	Logic is easy, coding is easy.
Modifiability	Manually control conversions, difficult to modify	Manually control conversions, difficult to modify	Manually division based on node characteristics.	Automatic update of state, action space and operation content.
Testability	SMC, Unity, Godot, Matlab,....	SMC, Unity, Godot, Matlab ,....	Visual Paradigm, WEKA,.....	BehaviorTree.CPP, behaviac, CocosCreator, Behavior Designer .
Robustness	1. Low parallelism; 2. State explosion occurs for multiple states;	1.Reduce sub-state coupling; 2.Parallel is easy to conflict; 3.Cooperative is easy to clock inconsistency.	1. Highly affected by changes in data; 2. Difficult to learn abstract concepts;	1.Modular easy decoupling, high readability and scalability; 2.Logic node control action execution; 3.Visualization manual design operation interface.

transition to *Idle*. Only when all four fallback nodes execute success does the root node ultimately receive success. In Fig. 1, the BT replaces the complex task-transition relation of the FSM by adding only three branches based on the template.

The transition from the FSM to the BT introduces several challenges. A primary concern is compatibility, as integrating the BT into existing routing protocols may lead to conflicts with the FSM. To address this issue, we adopt a phased approach, implementing incremental replacements rather than undertaking a complete overhaul, thereby ensuring a smooth transition and preserving system compatibility. Regarding debugging verification, while the BT offers two-way feedback, it lacks the capability to identify which specific function mounted under the logical node is malfunctioning. Consequently, this article proposes a feedback mechanism utilizing sockets within the mounted functions, facilitating more precise fault localization.

B. Comparative Analysis of Finite State Machine, Hierarchical Finite State Machine, Decision Tree and Behavior Tree

We compare the BT with other similar control architectures, including the FSM, HFSM, and DT, across eight aspects: modularity, hierarchy, readability, reusability, portability, modifiability, testability, and robustness, as presented in TABLE I. The definitions of these aspects are as follows: *Modularity* refers to the decomposition of a large program or system into smaller, more manageable modules or components. *Hierarchy* is a principle in software architecture that structures a system into various levels or layers. *Reusability* indicates that code or modules can be utilized across multiple projects or systems with minimal or no modifications. *Portability* signifies that the system can operate across different environments with little or no changes. *Robustness* denotes a program's resilience and reliability when encountering erroneous inputs or unexpected situations.

The comparisons in TABLE I are analyzed as follows:

1) **Modularity**: The smallest control unit differs across the four methods. Both the BT and the DT exhibit strong modularity, the HFSM has partial modularity, while the FSM lacks modularity entirely. The BT combines the structural aspects of the HFSM with the decision-making capabilities of the DT within a tree-like structure, offering a flexible and modular approach for representing complex behaviors [12].

2) **Hierarchy**: Both the FSM and the HFSM are event-driven with unidirectional transitions. The DT employs recursive classification to form a hierarchical tree structure, while the BT features a tick-driven, bidirectional hierarchical tree structure [13].

3) **Readability**: According to [14], the BT offers more excellent readability than the FSM and the HFSM, mainly for handling complex state problems. In this article, the FSM of BGP extends a *Security* state by adding seven state transition relations, as shown in Fig. 1, and the readability of the FSM is worse than that of the BT. Its tree structure and modularity enhance clarity. The DT also provides an intuitive structure, making it easy to understand [4].

4) **Reusability**: The BT demonstrates the highest reusability due to its strong modularity. The FSM is not modular and has the worst reusability. The HFSM offers reusability due to its hierarchical design, while DT's remains uncertain.

5) **Portability**: The logic of all four methods is simple, but in terms of code development, the BT is the easiest to develop based on templates; the DT is one of the most popular algorithms in machine learning, though its implementation complexity may vary. The FSM and HFSM are more challenging to code for complex scenarios, as they require managing multiple states and transitions.

6) **Modifiability**: The FSM, HFSM, and DT require manual modification and compilation by the developer for updates. In contrast, the BT supports automatic updates that can be triggered dynamically in response to different scenarios.

7) **Testability**: The four methods are tested with appropriate test tools.

8) **Robustness**: The FSM and HFSM struggle with complex states and parallel operations, while the DT's sensitivity to data changes leads to lower robustness. In contrast, the BT's modular design and decoupled structure provide enhanced flexibility and robustness, making it more suitable for intricate environments.

III. Behavior Trees in Network Routing Protocol

In this section, we first present a generic network routing protocol architecture. Then, we propose an implementable functional scheme.

A. Generic Network Routing Protocol Architecture of Behavior Tree

The generic routing protocol architecture proposed in this article aligns with the distributed structure of the Internet. However, when a new protocol is introduced to a network element, all other elements within the management domain require structural updates, as noted in [15]. This necessity for widespread adjustments presents practical challenges, often resulting in lengthy implementation processes. To address these issues, we propose a new routing protocol architecture with a three-plane structure: the management plane, the control plane, and the data plane, as shown in Fig. 2.

In the management plane, the *intent analysis* module primarily translates user intent into executable operations for network devices. First, user inputs in various forms are converted into natural language, with keyword extraction and intent interpretation performed through natural language processing. Complex intents are broken down into subtasks, which are then transformed into executable operations based on current network data and available network capabilities. The *BT visualization* module enables the construction of a BT framework using an editor, forming a basic operational template guided by the output of the *intent analysis* module. These BT templates are dynamically adjusted and reorganized during the subsequent decision-making process according to feedback from the *resource awareness* module.

In the control plane, network policies are generated by combining data from the *routing policy repository* with resource information on the calculation, memory, security, and forward capabilities sensed from the network. The control plane leverages the BT to separate control policies from control executions. The *decision module* applies decision rules through logical nodes within the BT, sending control signals to the execution module to carry out specific operational tasks. The *execution module* then assigns the generated routing table entries to *Zebra*, which interacts with the kernel to forward packets to the data plane. This process uses the open-source routing suite *FRRouting*, which includes multiple standard routing protocols. *Zebra*, the core daemon, manages routing tables and facilitates communication between the kernel and user space.

The data plane is designed to prioritize network topology and end-user deployment. Each network node integrates a BT,

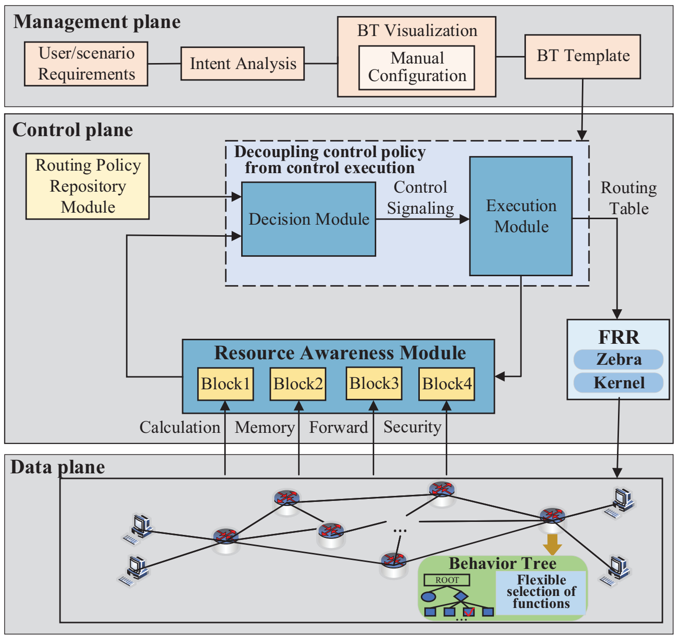


Fig. 2: Generic network routing protocol architecture using behavior tree.

which can be configured or activated to meet specific user requirements.

B. Implementable Functional Scheme of Behavior Tree

Building on our prior research [2], we reconstruct complex control logic for routing protocols utilizing the BT from a practical implementation perspective. Taking the intermediate system to intermediate system (IS-IS) routing protocol as an example, we model the entire process, from link establishment to final packet forwarding, leveraging the BT framework, as illustrated in Fig. 3.

During execution, control logic templates are constructed flexibly by editing control rules within the BehaviorTree.CPP editor and selecting logic nodes through the graphical user interface provided by the Groot plugin. This component primarily executes the source code of the network routing protocol during logical control operations. Furthermore, sockets are employed to link control rules to the action nodes of the BT, facilitating real-time reflection of the protocol's current status. For instance, at time *T_1*, four actions are defined: EXTABLISH_CONNECT, SEND_HELLO_PDU, CIRCUIT_UP, and FORWARD, which are executed sequentially as the protocol operates. To introduce a New_Function between CIRCUIT_UP and FORWARD at time *T_n*, an action node is inserted in Groot. It is important to note that newly added modules necessitate the prior preparation of the BT-based control logic templates. This approach significantly simplifies the extension of routing protocols with new functions, enhancing both the flexibility and scalability of protocol design and development.

IV. Proof-of-Concept Implementation of Behavior Tree for Routing Protocol

In this section, we design two use cases. Use case 1 establishes an equivalence analysis between the BT and the

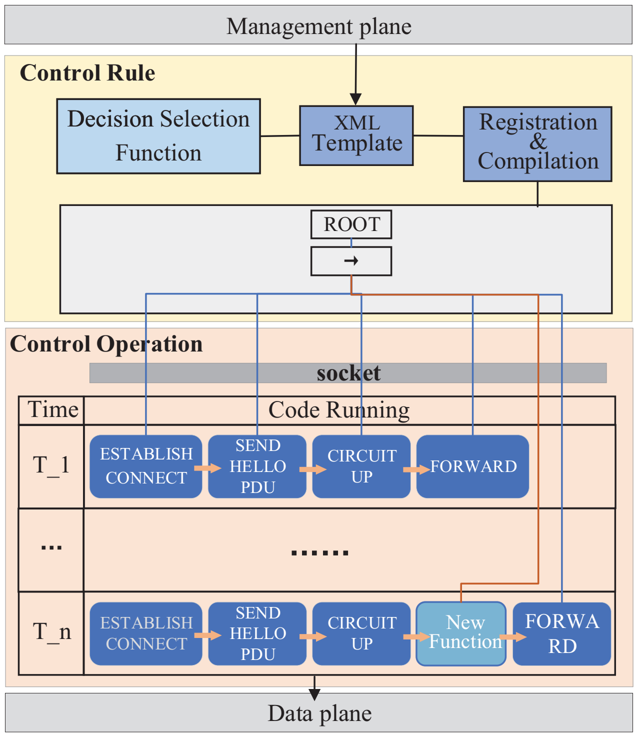


Fig. 3: Implementable control plane decoupling architecture of flexible network routing protocol utilizing behavior tree.

FSM, while use case 2 illustrates the scalability of the BT for extending routing functionalities.

A. *Use Case 1: An Equivalence Verification between Behavior Tree and Finite State Machine*

In this use case, we demonstrate the equivalence between the BT and the FSM. We establish a network topology utilizing the IS-IS protocol from the open-source IP routing suite FRRouting. Developed in C, this suite encompasses various standard routing protocols, including BGP, OSPF, and IS-IS, which contributes to its widespread adoption among organizations and institutions.

In the simulation, we configure network topologies consisting of three and five nodes, respectively. Focusing on the five-node configuration, which includes *R1*, *R2*, *R3*, *R4*, and *R5*, each node is configured with the IS-IS protocol and connected sequentially. During the conversion of the FSM to the BT mechanism within the IS-IS protocol, developers face significant challenges, primarily involving a comprehensive understanding of the state transition logic and interface relationships within the FSM code of the existing protocol. To address these challenges, developers need to accurately identify the relevant modules in the implementation code and systematically organize the FSM transition rules. We restructure the 149 lines of code in the *isis_csm.c* file according to the execution logic of the BT, leading to a total of 181 lines. This modification includes redesigning both state transitions and functional modules. Although these adjustments require code modifications, the challenges during development are effectively managed through proper modularization.

We employ five virtual machines to emulate network nodes, each operating on Ubuntu 20.04. The experiment utilizes the *ping* tool to simulate packet transmission. We execute the command *"ping 192.168.137.128"* to assess link connectivity from *R1* to *R5*. The evaluation method involves manually disrupting the link between *R3* and *R4* for 40 seconds during data transfer before restoring the connection. We measure the reconvergence time for two mechanisms: the FSM-based protocol and the BT-based protocol. Convergence is defined in terms of packet reachability, indicating that packet transmission within the network is restored following a link failure or update.

In the results, we present the distribution of ten data sets for each method, as illustrated in Fig. 4. The analysis reveals that the average convergence time for the BT-based protocol is 29.9 seconds, which is 0.8 seconds faster than the 30.7 seconds recorded for the FSM-based protocol. Similarly, in a three-node network, the average convergence time for the BT is 27.9 seconds, demonstrating a 0.4 seconds advantage over the FSM, which has an average convergence time of 28.3 seconds. These results indicate that the BT can effectively substitute the FSM in protocol implementations.

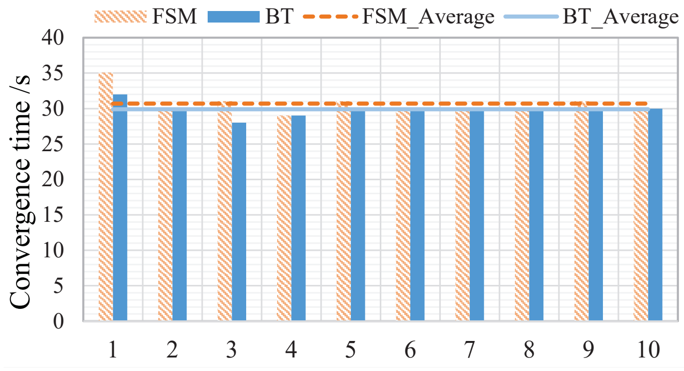


Fig. 4: Verification of the equivalence between finite state machines and behavior trees, with a comparative analysis of routing convergence time after replacing the circuit state machine component in the IS-IS routing protocol with the behavior tree.

B. *Use Case 2: An Scalability Analysis for Behavior Tree*

In this use case, we focus on demonstrating the scalability of the BT in terms of improved routing functionality.

Network failures and dynamic topology changes are primary factors affecting data transmission reliability. Conventional routing protocols often require time to converge following link state changes, which can lead to packet loss. Our research aims to implement a more rapid response mechanism through the BT to improve routing performance and mitigate the risk of data loss.

The new routing enhancement function is triggered when a packet transmission route is disrupted in the initial routing protocol. Specifically, when a port on a network node fails, the system promptly computes all available routes to the destination node. A new routing table format, based on the IPv6 routing table, is developed to accommodate these newly computed paths, which are subsequently inserted into the appropriate table entries. This updated table is then transmitted to the data forwarding plane to resume packet transmission. This design effectively reduces link state convergence time

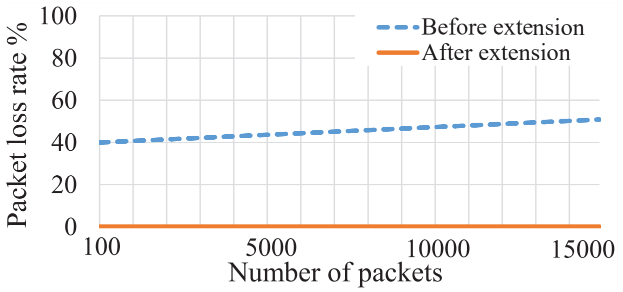


Fig. 5: Scalability of routing protocol functionalities utilizing the behavior tree to compare the impact on packet loss rates before and after the extension.

compared to the initial routing protocol. However, every approach has its drawbacks, which can result in additional overhead due to the need to carry network connection status information.

In the simulation, the network is configured such that *R1*, *R2*, and *R3* are interconnected, *R4* connects to *R1* and *h1*, *R5* connects to *R2* and *h2*, and *R6* connects to *R3* and *h3*. We transmit varying numbers of packets from *h2* to *h3*, subsequently interrupting the link between *R2* and *R3* to monitor the packet loss rate. Throughout the experiment, we record the packet loss before and after the extension of the new routing enhancement functionality BT for a comparative analysis. As illustrated in Fig. 5, following the enhancement, the packet loss rate for all packets significantly decreases, while the packet loss rate prior to the enhancement ranged from approximately 40% to 50%. These results demonstrate that the BT exhibits strong scalability in dynamically adjusting routes, particularly in environments characterized by frequent changes in link states.

V. Conclusion

In this article, we proposed a generic architecture for network routing protocols that leveraged the behavior tree to improve scalability and flexibility. Furthermore, we presented a functional scheme that enabled the effective deployment of the routing protocol. Finally, we conducted a proof-of-concept leveraging two use cases, revealing that: 1) the behavior tree served as an effective alternative to the finite state machine, achieving superior average routing convergence speeds, and 2) the behavior tree demonstrated scalability in adapting to dynamic routing adjustments. In future work, we plan to validate our method in larger and more complex network topologies, evaluate performance metrics such as CPU, memory overhead, and latency, and further clarify integration details. Additionally, we aim to explore incremental behavior tree adoption for conventional systems to enhance flexibility and demonstrate practical applications in platforms such as software-defined wide area networks, 5G, and the Internet of Things.

References

[1] M. Xu, H. Du, D. Niyato, J. Kang, Z. Xiong, S. Mao, Z. Han, A. Jamalipour, D. I. Kim, X. Shen, V. C. M. Leung, and H. V. Poor, "Unleashing the power of edge-cloud generative AI in mobile networks: A survey of AIGC services," *IEEE Commun. Surveys Tuts.*, vol. 26, no. 2, pp. 1127-1170, 2nd Quart., 2024.

[2] X. Gao, T. Feng, J. Du, and S. Jiang, "An programmable control plane framework by using behavior tree," in *Proc. 2021 4th Int. Conf. Hot Inf.Centric Network. (HotICN)*, Nanjing, China, Nov. 2021, pp. 74-80.

[3] A. Zengin and M. M. Ozturk, "Formal verification and validation with DEVS-Suite: OSPF Case study," *Simul. Model. Pract. Theory*, vol. 29, pp. 193-206, Dec. 2012.

[4] M. Colledanchise and P. Ögren, *Behavior Trees in Robotics and AI: An Introduction*. Boca Raton, FL, USA: CRC Press, 2018.

[5] B. Al-Musawi, P. Branch, and G. Armitage, "BGP anomaly detection techniques: A survey," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 1, pp. 377-396, 1st Quart., 2017.

[6] P. Spadaccino, S. Bruzzese, F. Cuomo, and F. Luciani, "Analysis and emulation of bgp hijacking events," in *Proc. IEEE/IFIP Network Operations and Management Symposium (NOMS)*, Miami, FL, USA, May. 2023.

[7] E. Coronado, F. Mastrogiovanni, and G. Venture, "Development of intelligent behaviors for social robots via user-friendly and modular programming tools," in *Proc. IEEE Workshop Adv. Robot. Social Impacts (ARSO)*, Genova, Italy, Sep. 2018, pp. 62-68.

[8] M. Iovino, E. Scukins, J. Styrud, P. Ögren, and C. Smith, "A survey of behavior trees in robotics and AI," *Robot. Auton. Syst.*, vol. 154, pp. 104096, Aug. 2022.

[9] M. Colledanchise and P. Ögren, "How behavior trees modularize hybrid control systems and generalize sequential behavior compositions, the subsumption architecture, and decision trees," *IEEE Trans. Robot.*, vol. 33, no. 2, pp. 372-389, Apr. 2017.

[10] M. Colledanchise and L. Natale, "On the implementation of behavior trees in robotics," *IEEE Trans. Robot. Autom.*, vol. 6, no. 3, pp. 5929-5936, Jul. 2021.

[11] B. Christalin, M. Colledanchise, P. Ögren, and R. M. Murray, "Synthesis of reactive control protocols for switch electrical power systems for commercial application with safety specifications," in *Proc. Symp. Computational Intell.*, Athens, Greece, Feb. 2016.

[12] A. v. Perger, P. Gamper, and R. Witzmann, "Behavior trees for smart grid control," *IFAC-PapersOnLine*, vol. 55, no. 9, pp. 122-127, 2022.

[13] M. Colledanchise, "Behavior trees in robotics," Ph.D. dissertation, KTH Royal Institute of Technology, 2017.

[14] M. Iovino, J. Förster, P. Falco, J. J. Chung, R. Siegwart, and C. Smith, "Comparison between behavior trees and finite state machines," *arXiv preprint*, arXiv: 2405.16137, 2024.

[15] H. Luo, S. Zhang, Z. Wang, and Q. Meng, "Understanding and thinking about the innovation on internet architecture," *Acta Electronica Sinica*, vol. 52, no. 4, pp. 1411-1420, 2024.

VI. Biography

Jiaorui Huang is currently pursuing her Ph.D. degree at Xidian University. Her research interests are artificial intelligence in intent-driven network and network routing protocols.

Chungang Yang is a full professor at Xidian University. His research interests are artificial intelligent 6G wireless mobile networks, intent-driven networks (IDN), space-terrestrial networks (STN), and game theory for emerging communication networks.

Tao Feng received Ph.D. degree in computer science from Tsinghua University, Beijing, China. His research interests are AI for networking, software-defined networks, and network management.

Lujia Dong is currently pursuing her masters degree at Xidian University. Her research interests are intent-driven network and network routing protocols.

Alagan Anpalagan received the B.A.Sc., M.A.Sc., and Ph.D. degrees in electrical engineering, in 2001 from the University of Toronto, Toronto, ON, Canada. He is currently a Professor in the Department of Electrical and Computer Engineering at Ryerson University, where he leads research on radio resource management and networking in the WINCORE Lab. He is a Registered Professional Engineer in the province of Ontario, Canada.

Qiang Ni received the Ph.D. degree in engineering from Huazhong University of Science and Technology, Wuhan, China, in 1999. He is currently a Full Professor and the Head of the Communication Systems Research Group, School of Computing and Communications, Lancaster University, Lancaster, U.K. His research interests include future generation communications and networking systems.

Mohsen Guizani received the B.S. (with distinction), M.S., and Ph.D. degrees in electrical and computer engineering from Syracuse University, Syracuse, NY, USA in 1985, 1987, and 1990, respectively. He is currently a Professor and the Associate Provost with Mohamed Bin Zayed University of Artificial Intelligence, Abu Dhabi, UAE. His research interests include applied machine learning, smart city, wireless communications/networking, cloud computing, and security. Prof. Guizani is currently serving on the editorial boards of many IEEE transactions and magazines.