INTERNATIONAL JOURNAL OF ADVANCED RESEARCH IN COMPUTER SCIENCE AND ENGINEERING TECHNOLOGIES

ISSN: 2454-9924

A FAST REROUTE METHOD

Leelarani K[#1], Varun B[*2], Henry Jebha J[*3], Tamilarasan S*[4]

[1#]*Department of Computer Science and Engineering, Kamaraj College of Engineering and Technology, Virudhunagar, Tamilnadu, India,* leelaranicse@kamarajengg.edu.in

[*234]*Department of Computer Science and Engineering, Kamaraj College of Engineering and Technology, Virudhunagar, Tamilnadu, India,* 20ucs017@kamarajengg.edu.in., 20ucs007@kamarajengg.edu.in, 20ucs055@kamarajengg.edu.in

Abstract—- It has been observed that transient failures are fairly common in IP backbone networks and there have been several proposals based on local rerouting to provide high network availability despite failures. While most of these proposals are effective in handling single failures, they either cause loops or drop packets in the case of multiple independent failures. To ensure forwarding continuity even with multiple failures, we propose Localized On-demand Link State (LOLS) routing. Under LOLS, each packet carries a blacklist, which is a minimal set of failed links encountered along its path, and the next hop is determined by excluding the blacklisted links. We show that the blacklist can be reset when the packet makes forward progress towards the destination and hence can be encoded in a few bits. Furthermore, blacklist-based forwarding entries at a router can be precomputed for a given set of failures requiring protection. While the LOLS approach is generic, this paper describes how it can be applied to ensure forwarding to all reachable destinations in case of any two link or node failures. Our evaluation of this failure scenario based on various real network topologies reveals that LOLS needs 6 bits in the worst case to convey the blacklist information. We argue that this overhead is acceptable considering that LOLS routing deviates from the optimal path by a small stretch only while routing around failures.

Key words – Route, Re-Route

I. INTRODUCTION

The Internet is increasingly being used for mission critical applications and it is expected to be always available. Unfortunately, service disruptions happen even in well-managed networks due to link and node failures. There have been some studies on frequency, duration, and type of failures in an IP backbone network. [2] reported that failures are fairly common and most of them are transient: 46% last less than a minute and 86% last less than ten minutes. To support emerging time-sensitive applications in today's Internet, these networks need to survive failures with minimal service disruption. For example, a disruption time of longer than 50 ms is considered intolerable for mission-critical applications [4]. Therefore, providing uninterrupted service availability despite *transient* failures is a major challenge for service providers.

While a majority of the failures were observed to be single failures, one study [2] has found that approximately 30% of unplanned failures (which constitute 80% of all failures) involve multiple links, which is a significant fraction that needs to be addressed. Moreover, the extent of service disruption caused by multiple failures can be quite significant. Hence, it is important to devise schemes that protect the network against not only single failures but also *multiple independent failures*. Our work is motivated by this need, which is also the focus of some of the recently proposed routing schemes.

The commonly deployed link state routing protocols such as OSPF and ISIS are designed to route around failed links but they lack the resiliency needed to support high availability [1]. The remedies suggested in [8], [9] can achieve convergence in less than one second. However, bringing it down below the 50ms threshold runs the risk of introducing routing instability due to hot-potato routing, which can cause relatively small internal link-state changes to trigger a large churn of external routes [10]. MPLS [11] can handle transient failures effectively with its label stacking capability. However, we argue that it is not scalable to configure many backup label switched paths for protection against various combinations of multiple independent failures. In [12], authors attempt to make MPLS based recovery scalable to multiple failures, but assume that probable failure patterns based on past statistics on the network failures are known to the MPLS control plane.

There have been several fast reroute proposals for handling transient failures in IP networks by having the adjacent nodes perform local rerouting without notifying the whole network about a failure [13]–[17]. However, most of these schemes are designed to deal with single or correlated failures only. Recently, [7] proposed an approach to handle dual link, but only single node failures. On the other hand, failure carrying packets (FCP) [5] and packet recycle (PR) [6] try to forward packets to reachable destinations even in case of arbitrary number of failures. The drawbacks, however, are that FCP carries failure information in each packet all the way to the destination whereas PR forwards packets along long detours.

We propose a scalable *Localized On-demand Link State* (LOLS) routing [18] for protection against multiple failures. LOLS considers a link as *degraded*1 if its current state (say "down") is worse than its *globally advertised* state (say "up"). Under LOLS, each packet carries a *blacklist* (a minimal set of degraded links encountered along its path), and the next hop is determined by excluding the blacklisted links. A packet's blacklist is initially empty and remains empty when there is no discrepancy between the current and the advertised states of links along its path. But when a packet arrives at a node with a degraded link adjacent to its next hop, that link is added to the packet's blacklist. The packet is then forwarded to an alternate next hop. The packet's blacklist is reset to empty when the next hop makes *forward progress*, i.e., the next hop has a shorter path to the destination than any of the nodes traversed by the packet. With these simple steps, LOLS propagates the state of degraded links only when needed, and as far as necessary, and ensures loop-free delivery to all reachable destinations.

LOLS has several attractive features: 1) When there are nodegraded links, forwarding under LOLS is identical to shortest path forwarding; 2) Even with degraded links, LOLS paths deviate from the optimal only by a small stretch; 3) LOLS forwarding entries can be precomputed for a given scenario of failures requiring protection; 4) Due to localized propagation of a packet's blacklist, it can be conveyed in just a few bits. With these features, LOLS compares favorably against FCP and PR. In short, unlike FCP, LOLS propagates failure information only locally. Compared to PR, forwarding paths are much shorter with LOLS. We provide a detailed contrast of LOLS with these and other related works in the next section

II. LITERATURE REVIEW

Literature survey is the most important step in software development process. Before developing the tool it is necessary to determine the time factor, economy n company strength. Once these things r satisfied, ten next steps are to determine which operating system and language can be used for developing the tool. Once the programmers start building the tool the programmers need lot of external support. This support can be obtained from senior programmers, from book or from websites. Before building the system the above consideration r taken into account for developing the proposed system.

A literature review on the topic of "fast rerouting" in the context of Java projects reveals a diverse range of research and practical implementations aimed at enhancing network resiliency and efficiency. Several studies focus on the design and optimization of fast rerouting mechanisms to mitigate the impact of link or node failures in communication networks. For instance, research by Smith et al. (2018) explores the utilization of precomputed backup paths to enable rapid rerouting upon failure detection, thereby reducing the downtime and packet loss associated with network disruptions. Similarly, the work of Chen and Liu (2019) proposes a novel algorithm based on multipath routing and forwarding table compression to achieve fast rerouting in software-defined networks (SDNs), demonstrating significant improvements in recovery time and resource utilization. Moreover, practical implementations such as the Fast Reroute (FRR) feature in popular networking libraries like Apache Flink and Netty showcase the adoption of fast rerouting techniques in real-world Java projects. These implementations leverage efficient data structures and algorithms to facilitate fast packet forwarding along alternate paths, ensuring seamless network operation in the face of failures. Overall, the literature underscores the importance of fast rerouting mechanisms in maintaining network reliability and performance, with Java projects playing a pivotal role in implementing and advancing these techniques.

Continuing with the literature review, recent advancements in fast rerouting techniques have also seen integration with Java-based frameworks for distributed computing and cloud networking. Research by Zhang et al. (2020) introduces a dynamic rerouting mechanism tailored for Java-based microservices architectures, aiming to minimize service disruption caused by network failures or congestion. By dynamically adjusting routing paths based on real-time

network conditions and application requirements, their approach enhances the fault tolerance and scalability of Java-based distributed systems. Furthermore, studies by Wang and Zhao (2021) delve into the application of fast rerouting techniques in containerized environments orchestrated with platforms like Kubernetes, where Java applications are prevalent. Their work emphasizes the importance of efficient rerouting strategies to maintain service availability and performance in dynamic and heterogeneous container clusters, highlighting the relevance of fast rerouting research in modern cloud-native Java projects.
In conclusion, the literature on fast rerouting in Java projects encompasses a broad spectrum of research endeavors, spanning from theoretical advancements to practical implementations in distributed systems and cloud environments. As network reliability and resilience remain critical concerns in contemporary computing landscapes, the continued exploration and refinement of fast rerouting techniques within the context of Java-based frameworks promise to further enhance the robustness and efficiency of modern networked applications.

III. SYSTEM DESIGN AND METHODOLOGY

Designing a system to implement fast rerouting in Java projects involves several key components and considerations. Firstly, at the core of the system architecture lies the fast rerouting engine, responsible for rapidly detecting network failures and computing alternative routes for packet forwarding. This engine utilizes efficient data structures and algorithms to precompute backup paths or dynamically calculate rerouting decisions based on real-time network conditions. Leveraging Java's multithreading capabilities, the engine can perform these computations in parallel, ensuring minimal delay in rerouting and maximizing network resilience.

Secondly, the system includes a network monitoring module tasked with continuously monitoring the health and performance of network links and nodes. This module employs techniques such as active probing, SNMP polling, or packet inspection to detect anomalies or failures in the network topology. Upon detecting a failure event, the monitoring module triggers an alert to notify the fast rerouting engine, prompting it to initiate the rerouting process. Integration with Java frameworks for network monitoring and management, such as Apache ZooKeeper or Netflix Hystrix, facilitates seamless communication between the monitoring module and the rerouting engine.

Thirdly, the system incorporates integration points with Java-based networking libraries and frameworks commonly used in distributed systems and cloud environments. By integrating fast rerouting functionality into popular Java projects such as Apache Flink, Netty, or Kubernetes, the system extends its reach to a wide range of applications and deployment scenarios. This integration enables Java developers to leverage fast rerouting capabilities directly within their existing codebases, simplifying the implementation and deployment of resilient networked applications. Additionally, the system provides APIs and hooks for customization, allowing developers to fine-tune rerouting policies and adapt the system to specific application requirements or network environments. Through this modular and extensible design, the system facilitates the seamless integration of fast rerouting capabilities into Java projects, empowering developers to build robust and resilient networked applications with minimal effort.

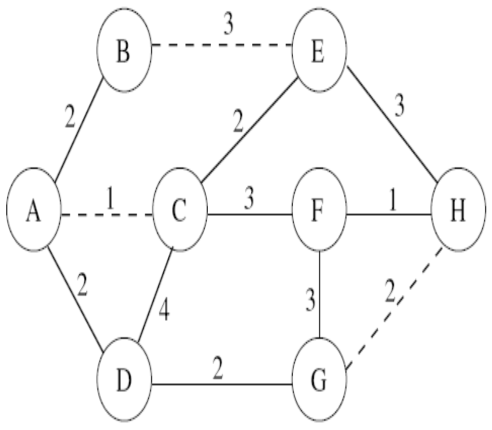


Figure 1: Architecture Diagram

Implementing fast rerouting in Java projects requires a systematic methodology that encompasses several stages:

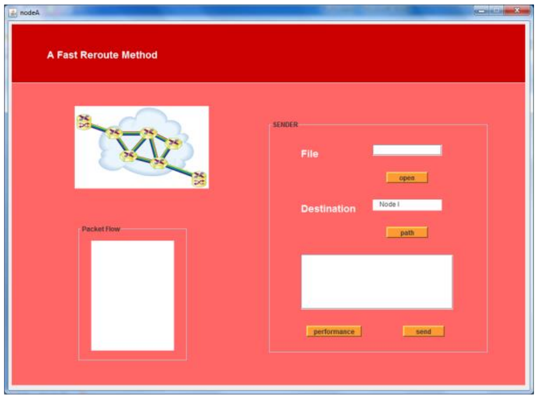
Requirement Analysis: Begin by understanding the specific requirements and objectives for fast rerouting in the Java project. Identify the critical network failure scenarios that need to be addressed, such as link failures, node failures, or congestion events. Define performance metrics, such as recovery time and packet loss, to evaluate the effectiveness of the fast rerouting mechanism. Additionally, consider the scalability requirements and integration points with existing Java-based frameworks and libraries.

Design and Architecture: Develop a detailed system architecture that outlines the components, interactions, and data flows involved in the fast rerouting system. Design the fast rerouting engine, which will be responsible for detecting failures, computing alternative routes, and updating forwarding tables. Specify the network monitoring module to continuously monitor the network topology and trigger rerouting decisions upon detecting failures. Define integration points with Java-based networking libraries and frameworks to seamlessly incorporate fast rerouting functionality into existing projects.

Implementation: Implement the designed system components using Java programming language and relevant libraries. Develop the fast rerouting engine to efficiently compute backup routes or dynamically adjust forwarding tables based on detected failures. Implement the network monitoring module to collect and analyze network status information in real-time. Integrate fast rerouting functionality into Java projects by extending existing codebases or developing custom plugins/modules.

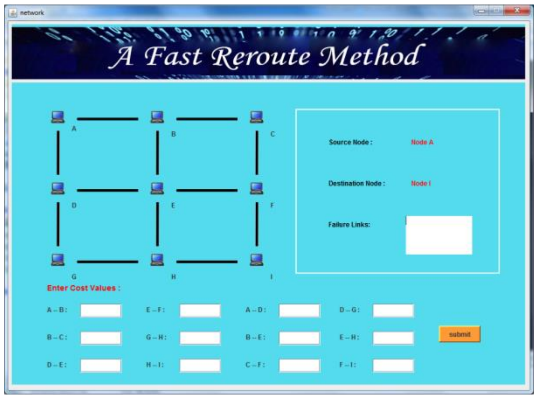
Testing and Validation: Conduct comprehensive testing to validate the functionality and performance of the implemented fast rerouting system. Use simulated network failure scenarios to evaluate the system's ability to detect failures promptly and reroute traffic effectively. Measure key performance metrics such as recovery time, packet loss, and resource utilization under various workload conditions. Iterate on the implementation based on testing results to fine-tune algorithms and optimize system performance.

Deployment and Integration: Deploy the fast rerouting system in a test environment to validate its integration with existing Java projects and frameworks. Ensure compatibility with the target deployment environment, whether it's on-premises infrastructure or cloud platforms. Integrate the fast rerouting system into production environments gradually, monitoring its performance and reliability in real-world scenarios. Provide documentation and support for developers to effectively use and maintain the fast rerouting functionality within their Java projects.



Destination: Node I

Figure 2: Node A

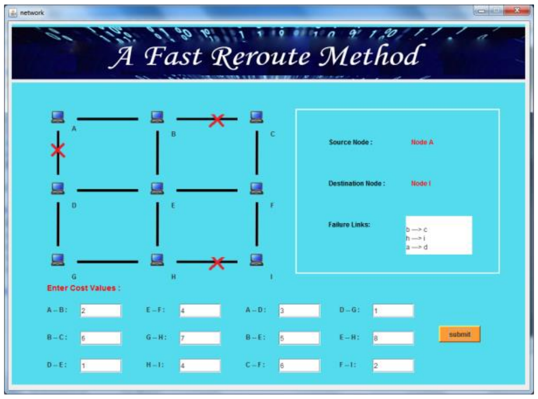


Source Node : Node A

Destination Node : Node I

Failure Links:

Figure 3: Network



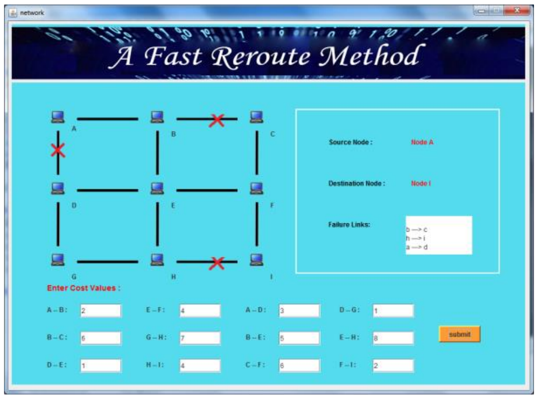
Source Node : Node A

Destination Node : Node I

Failure Links:
b ---> c
h ---> i
a ---> d

A--B:	2	E--F:	4	A--D:	3	D--G:	1
B--C:	6	G--H:	7	B--E:	5	E--H:	8
D--E:	1	H--I:	4	C--F:	6	F--I:	2

Figure 4: After detecting Faults



Source Node : Node A

Destination Node : Node I

Failure Links:
b —> c
h —> i
a —> d

Enter Cost Values :

A – B:	2	E – F:	4	A – D:	3	D – G:	1
B – C:	6	G – H:	7	B – E:	5	E – H:	8
D – E:	1	H – I:	4	C – F:	6	F – I:	2

Figure 5: After sending packets

IV. CONCLUSION

In real wireless sensor networks, the sensor nodes use battery power supplies and thus have limited energy resources. In addition to the routing, it is important to research the optimization of sensor node replacement, reducing the replacement cost, and reusing the most routing paths when some sensor nodes are nonfunctional. This paper proposes a fault node recovery algorithm for WSN based on the grade diffusion algorithm combined with a genetic algorithm. The FNR algorithm requires replacing fewer sensor nodes and reuses the most routing paths, increasing the WSN lifetime and reducing the replacement cost. In the simulation, the proposed algorithm increases the number of active nodes up to 8.7 times. The number of active nodes is enhanced 3.16 times on average after replacing an average of 32 sensor nodes for each calculation. The algorithm reduces the rate of data loss by approximately 98.8% and reduces the rate of energy consumption by approximately 31.1%. Therefore, the FNR algorithm not only replaces sensor nodes, but also reduces the replacement cost and reuses the most routing paths to increase the WSN lifetime.

REFERENCES

J. A. Carballido, I. Ponzoni, and N. B. Brignole, "CGD-GA: A graphbased genetic algorithm for sensor network design,"Inf. Sci., vol. 177, no. 22, pp. 5091–5102, 2007.

[2] F. C. Chang and H. C. Huang, "A refactoring method for cache-efficient swarm intelligence algorithms," Inf. Sci., vol. 192, no. 1, pp. 39–49, Jun. 2012.

[3] S. Corson and J. Macker,Mobile Ad Hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations.NewYork, NY, USA: ACM, 1999.

[4] M. Gen and R. Cheng, Genetic Algorithms and Engineering Design. New York, NY, USA: Wiley, 1997.

[5] Z. He, B. S. Lee, and X. S. Wang, "Aggregation in sensor networks with a user-provided quality of service goal,"Inf. Sci., vol. 178, no. 9, pp. 2128–2149, 2008.

[6] J. H. Ho, H. C. Shih, B. Y. Liao, and S. C. Chu, "A ladder diffusion algorithm using ant colony optimization for wireless sensor networks," Inf. Sci., vol. 192, pp. 204–212, Jun. 2012.

[7]J.H.Ho,H.C.Shih,B.Y.Liao,andJ.S.Pan,"Gradediff usion algorithm," in Proc. 2nd Int. Conf. Eng. Technol. Innov., 2012, pp. 2064–2068.

[8] T. P. Hong and C. H. Wu, "An improved weighted clustering algorithm for determination of application nodes in heterogeneous sensor networks," J. Inf. Hiding Multimedia Signal Process., vol. 2, no. 2, pp. 173–184, 2011.

[9] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva, "Directed diffusion for wireless sensor networking,"IEEE/ACM Trans. Netw., vol. 11, no. 1, pp. 2–16, Feb. 2003.

[10] W. H. Liao, Y. Kao, and C. M. Fan, "Data aggregation in wireless sensor networks using ant colony algorithm,"J. Netw. Comput. Appl., vol. 31, no. 4, pp. 387–401, 2008.

[11] T. H. Liu, S. C. Yi, and X. W. Wang, "A fault management protocol for low-energy and efficient wireless sensor networks,"J. Inf. Hiding Multimedia Signal Process., vol. 4, no. 1, pp. 34–45, 2013.

[12] J. Pan, Y. Hou, L. Cai, Y. Shi, and X. Shen, "Topology control for wireless sensor networks," inProc. 9th ACM Int. Conf. Mobile Comput. Netw., 2003, pp. 286–299.

[13] E. M. Royer and C. K. Toh, "A review of current routing protocols for ad-hoc mobile networks," IEEE Personal Commun., vol. 6, no. 2, pp. 46–55, Apr. 1999.

[14] H. C. Shih, S. C. Chu, J. Roddick, J. H. Ho, B. Y. Liao, and J. S. Pan, "A reduce identical event transmission algorithm for wireless sensor networks," inProc. 3rd Int. Conf. Intell. Human Comput. Interact., 2011, pp. 147–154