An Efficient Caching Scheme and Consistency Maintenance in Hybrid P2p System

Ms. N. Sruthilaya[1], Mr.S. Sundareshwaran[2]
[1]*Assistant Professor, Department of MCA, Mohamed Sathak Engineering College.*
[2]*Final year MCA, Mohamed Sathak Engineering College*

***Abstract*—It has been observed that transient failures are common in IP backbone networks and there have been several proposals based on local fast 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 links or node failures.**

***Index Terms*—Network Availability, Transient Failures, Local Fast Rerouting, Link State Routing, Fault Tolerance.**

1.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 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 most 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 backups 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.

2. LITERATURE SURVEY

Mastering Caching in Java: Key Concepts and Implementation Details by Ahmet Temel Kundupoglu

(2025). This article provides an in-depth exploration of caching techniques in Java applications, covering essential concepts such as Time-to-Live (TTL), cache invalidation, publish-subscribe caching, write-through caching, distributed locking, and the Redlock algorithm. Kundupoglu offers practical examples using libraries like Caffeine, Guava, and Redisson, demonstrating how to implement these caching strategies to enhance application performance and scalability.

Understanding Application-Level Caching in Web Applications: A Comprehensive Introduction and Survey of State-of-the-Art" by Jhonny Mertz and Ingrid Nunes (2020)**.** This comprehensive survey delves into application-level caching within web applications, emphasizing the integration of caching logic directly into the application's codebase. The authors discuss the benefits of this approach, such as improved performance and scalability, and address challenges like cache invalidation and consistency. The paper also reviews adaptive caching solutions that adjust to changing workloads to prevent performance degradation over time.

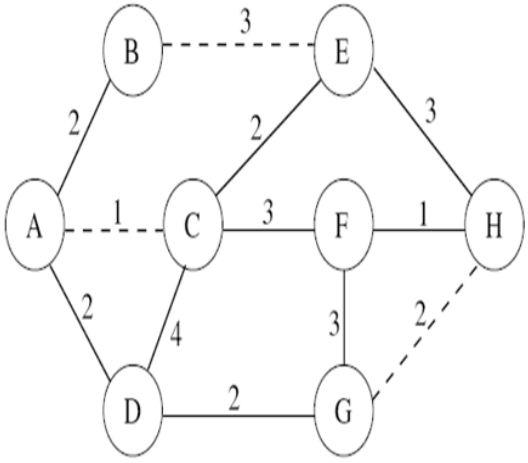
Improving Java Performance and Energy Dissipation through Efficient Code Caching by Yu Sun and Wei Zhang (2009)**.** Although published slightly earlier, this study remains relevant as it investigates hardware-based code catching strategies aimed at enhancing Java application performance and reducing energy consumption. The authors propose methods for writing dynamically generated Java code directly into the instruction cache, bypassing traditional pathways that involve the data cache.

Ehcache: Java's Open-Source Cache" (2022) by Greg Luck.Ehcache offers features like in-memory data storage, disk overflow, and distributed caching. Its straightforward API and robust functionality make it a popular choice for developers seeking to implement caching in Java applications. Merits: 1. High Performance: Ehcache is designed for high performance and can handle large volumes of data.Ehcache allows for flexible configuration, including cache size, expiration policies, and cache clustering. Ehcache's flexible configuration can also make it complex to configure, especially for beginners.Ehcache has a steep learning curve, requiring significant time and effort to master.

Exploring Advanced Caching Patterns in Java by Emily Chen (2023)**.** This article examines sophisticated caching patterns in Java, including read-through, write-behind, and cache-aside strategies. Chen discusses the scenarios where each pattern is most effective and provides implementation examples using modern Java frameworks. The article also addresses common pitfalls in caching, such as cache stampedes and data inconsistency, offering solutions to mitigate these issues. Advanced caching patterns can significantly improve the performance of Java applications by reducing the number of database queries and computations. Advanced caching patterns can help Java applications handle increased traffic and user loads without compromising performance.

2.1 EXISTING SYSTEM**:** The existing system describes the concept does not describe how the packet to redirected once node within the path is unavailable or corrupted. This scheme uses a separate cache layer to store data, and the application is responsible for maintaining cache consistency.

This scheme uses a cache layer to store data, and the cache is updated whenever data is read from the underlying storage of multipath routing from the source to root within the network. It also has various techniques to handle data loss, delayed timing, loss of acknowledgement.



But it did. When the energy of a sensor node is exhausted, wireless sensor network leaks will appear, and the failed nodes will not relay data to the other nodes during transmission processing. Thus, the other

sensor nodes will be burdened with increased transmission processing. This scheme uses a separate cache layer to store data, and the application is responsible for maintaining cache consistency. This scheme uses a cache layer to store data, and the cache is updated whenever data is read from the underlying storage. This scheme uses a cache layer to store data, and the cache is updated whenever data is written to the underlying storage.

2.2 PURPOSE OF WORK

The purpose of this project is to design and develop an efficient caching scheme and consistency maintenance mechanism for Hybrid Peer-to-Peer (P2P) systems. The goal is to improve the performance, scalability, and reliability of Hybrid P2P systems by reducing latency, minimizing data inconsistencies and ensuring seamless data sharing among peers.

2.3 PROPOSED SYSTEM

We propose Fault Node Recovery. Under this technique, 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. This scheme combines the benefits of cache aside, read-through, and write-through schemes to provide an efficient and consistent caching mechanism. The algorithm proposed in this paper is based on the GD algorithm, with the goal of replacing fewer sensor nodes that are inoperative or have depleted batteries, and of reusing the maximum number of routes. These optimizations will ultimately enhance the WSN lifetime and reduce sensor node replacement cost. The proposed scheme reduces the number of caches missing and improves overall system performance. The proposed scheme ensures cache consistency by updating the cache layer whenever data is read or written.

3.MODULES

3.1. Blacklist

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.

3.2. Multipath Routing

Multipath routing is a promising routing scheme to accommodate these requirements by using multiple pairs of routes between a source and a destination. Multipath routing is the routing technique of using multiple alternative paths through a network, which can yield a variety of benefits such as increased bandwidth, or improved security. The multiple paths computed might be overlapped, edge-disjointed or node-disjointed with each other. Extensive research has been done on multipath routing techniques.

3.3. Directed Diffusion Algorithm

The goal of the DD algorithm is to reduce the data relay transmission counts for power management. The DD algorithm is a query-driven transmission protocol. The collected data is transmitted only if it matches the query from the sink node. In the DD algorithm, the sink node provides the queries in the form of attribute-value pairs to the other sensor nodes by broadcasting the query packets to the whole network. Subsequently, the sensor nodes send the data back to the sink node only when it fits the queries.

3.4. Grade Diffusion Algorithm

The GD algorithm not only creates the routing for each sensor node but also identifies a set of neighbour nodes to reduce the transmission loading. Each sensor node can select a sensor node from the set of neighbour nodes when its grade table lacks a node able to perform the relay. The GD algorithm can also record some information regarding the data relay. Then, a sensor node can select a node with a lighter loading or more available energy than the other nodes to perform the extra relay operation. That is, the GD algorithm updates the routing path in real time, and the event data is thus sent to the sink node quickly and correctly.

3.5. Fault Node Recovery

Fault node recovery (FNR) algorithm for WSNs based on the grade diffusion algorithm combined with the genetic algorithm. The FNR algorithm creates the grade value, routing table, neighbor nodes, and payload value for each sensor node using the grade diffusion algorithm. In the FNR algorithm, the number of nonfunctioning sensor nodes is calculated during the wireless sensor network operation, The sensor nodes transfer the event data to the sink node according to the GD algorithm when events appear.

3.6. Fast Reroute

Techniques developed for fast recovery from multiple-link failures provide more than one forwarding edge to route a packet to a destination. Whenever the default forwarding edge fails or a packet is received from the node attached to the default forwarding edge for the destination, the packets are rerouted on the backup ports. In the authors present a framework for IP fast reroute detailing three candidate solutions for IP fast reroute that have all gained considerable attention. When a forwarding link on a tree fails, the packet may be switched to the other tree.

4. RESULT AND CONCLUSION

In real wireless sensor networks, the sensor nodes use battery power supplies and thus have limited energy resources. In addition to 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 Since mobile nodes are moving in the network, we should save more battery lifetime, bandwidth and threshold in future nodes, but also reduces the replacement cost and reuses the most routing paths to increase the WSN lifetime.

5. FUTURE ENHANCEMENT

Deploy ability. We can evaluate the overhead due to LOLS using several large real topologies and shown that it scales better than the recently proposed scheme FCP which has similar failure resilience objectives. Our plan is to implement a prototype of LOLS using Mini net to demonstrate it does deploy ability. We can evaluate the overhead due to LOLS using several large real topologies and shown that it scales better than the recently proposed

scheme FCP which has similar failure resilience objectives. In future, we can enhance this concept with thousands of nodes. In which we can construct sensor nodes to transfer the data and a sink node to hold the data.

REFERENCES

[1] R. Agrawal, T. Imielinski, and A. Swami, “Mining Association Rule between Sets of Items in Large Databases,” Proc. ACM SIGMOD Conf. on Management of Data, pp. 207-216, May 2023.

[2] R. Agrawal and R. Srikant, “Fast Algorithm for Mining Association Rules,” Proc. Int’l. Conf. on Very Large Databases, pp. 478- 499, Sept. 2024.

[3] R. Agrawal and R. Srikant, “Mining Sequential Patterns,” Proc. Int’l. Conf. on Data Engineering, pp. 3-14, Mar. 2023.

[4] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, “Basic Local Alignment Search Tool,” J. of Molecular Biology, vol. 215, no. 3, pp. 403-410, Oct. 2022.

[5] M.-S. Chen, J.-S. Park, and P. S. Yu, “Efficient Data Mining for Path Traversal Patterns,” IEEE Trans. on Knowledge and Data Engineering, vol. 10, no. 2, pp. 209-221, Apr. 2022.

[6] J. Han and M. Kamber, “Data Mining: Concepts and Techniques, 2nd Edition,” Morgan Kaufmann, Sept. 2000.