Routing Framework for FTIMA – A Fault Tolerance Infrastructure for Mobile Agents

KIRAN IJAZ, UMAR MANZOOR, ARSHAD ALI SHAHID
Department of Computer Science
National University of Computer and Emerging Sciences, NUCES-FAST
H-11/4 A.K Barohi Road, Islamabad.
PAKISTAN

Abstract: - Route planning is defined as finding an optimal path for an object from start node to goal node while considering the constraints of the system. Due to inherent nature of urgency and criticality of ATM networks, optimized path routing technique is considered as the backbone for their success or failure. In this paper we have proposed a fault tolerant routing framework for a transactional system to make better decisions in choice of routes before using them. We have provided standby options so that in case of route failures we can route our transactions to next working path. Our framework has been designed for ATM networks, as in these systems we need a path which is highly reliable and provides timely service through proper routing of transactions. In case of any criticalities in a path the transaction is routed intelligently to another optimal path without any delays or human intervention. We have observed through the simulated experiments that the proposed approach yields optimal performance using k paths instead of one.

Key-Words: - Route Planning, ATM Networks, FTIMA Architecture, Transactional System, Fault Tolerant Systems, K Shortest Paths

1 Introduction
Most of the larger enterprise applications in the areas of finance, banking, and electronic commerce rely on transaction processing for delivering their business functionality. Enterprise applications often require concurrent access to distributed data shared amongst multiple components so underlying network plays an important role [1].

The network should stay operative and provide reliable communication even when a significant number of network nodes and transmission links fail. Hence packet forwarding should be controlled in a fully distributed manner and routing units should more or less be independent of each other.

The problem we address in this paper is how to provide fault tolerant and robust transactional system and a server or link failure prone network, by providing opportunity of multiple routing paths for a transaction. Multiple paths also provide the facility of load balancing. A related advantage is the ability of dynamic and transparent allocation of another k-1 paths (in case of occurring any hurdle in a route ubiquitously, without requiring the explicit change in state or human intervention).

Route planning can be defined as finding an optimal path for an object from a start to a goal configuration without colliding with obstacles in the environment, moving in loops, or reaching dead ends. In certain transactions it is also important to guarantee fault tolerant paths in the face of network failures.

We propose a fault tolerant routing framework for a transactional system to make better decisions in choice of routes before using them. We have provided standby ‘options’ so that in case of route failures we can route our transactions to next working path. Our implementation has been designed for ATM networks. Due to their inherent nature of urgency and criticality, optimized path routing technique is considered as the backbone for their success or failure.

We propose the use of k-shortest paths algorithm proposed by David [2] in Fault Tolerant Infrastructure for Mobile Agents – FTIMA. This algorithm is based on the extension of a single shortest path finding algorithm named as Dijkstra. It finds k-Shortest Paths instead of one, providing k-1 alternate routing opportunities between a given node pair. The proposed algorithm is not limited to finding fully disjoint paths and does not remove any links or nodes from the graph [2].
2 Path Optimization

Information is transported from source to destination through a set of physical links called paths. Each path has a fixed bandwidth and length which represents a connection between two nodes which may or may not be adjacent. In general, most applications require a short non-cyclic path, because redundant steps take extra time to execute.

In an ATM-Banking system we need a path which is highly reliable and provides timely service through proper routing of transactions. In case of any criticalities in a path the transaction should be routed intelligently to another optimal path without any delays or human intervention.

A path may or may not be available for a transaction to be routed. There can be several reasons for this non-availability; a link between two routers or a router itself may not be in functional position, or the path does not have capacity to route transactions more than its threshold value $\theta$ and it is already routing maximum traffic equal to $\theta$ as shown in figure 1.

Given an arbitrary network topology and an arbitrary traffic matrix, a problem can be formulated to determine a partitioning of network resources into a set of physical paths which is in some sense optimal over the set of all physical path partitions. A solution approach to this problem is investigated based on the assumption of a k-shortest path routing technique, also known as multi-path routing.

3 Traffic Scheduling over Multiple Network Paths

Multi-path routing enables a network’s traffic to be split among two or more paths in order to reduce latency, improve throughput, and balance traffic loads. Multi-path routing establishes multiple network paths between series of routers to provide more efficient load balancing and higher-performance paths as compared to unipath routing.

Our proposed Fault Tolerant Infrastructure for Mobile Agents – FTIMA selects the shortest path between source and target machine, and routs the transaction if it is available. But if a router is not available or some link in the path is down i.e. the whole path is not available; what should be the behavior of system in the above case? There are certain options for the system to respond in:

1. Reject the transaction and display message; “Sorry the system is unavailable. Try again after some time”.
2. Wait for some time and when the path is available then route the transaction.
3. Route the transaction through any other path which could be the longest one.
4. Route the transaction on the second less optimal path.

Now let us evaluate each option one by one; if we take first and second case then the system’s performance becomes very low i.e. it will not be...
reliable as the path may remain down for a long time and the transactions will remain blocked for long. Now, if we consider third option as a solution to our problem then in worse case any other path may be the longest of all and can take much more time to route a transaction, which is not feasible at all and affects the timely service of our system. There should be some smart way of solving this problem. In our system we have solved this problem through mapping shortest path problem to k-Shortest Paths problem i.e. if the shortest path is down the transaction can be routed through the next shortest path; if this path is also not available then select the third shortest path and so on. See section 7 for further explanation.

3.1 Shortest Paths
For the purpose of multi-path routing, the ‘k – Shortest Paths’ problem was well studied dated back to the late 1950’s [2, 5]. Different versions of this problem exist subject to various types of restrictions. It is a long-studied generalization of the shortest path problem, in which not one but several paths in increasing order of length are sought. The k shortest paths problem is to list the k paths connecting a given source-destination pair in the graph with minimum total length. See figure 2.

![Figure 2](image)

Figure (2) Dotted lines in the above network show two most optimal paths between A (Source) and B (Destination).

The proposed approach yields optimal performance using k paths instead of one. K shortest path is designed to find the scheduled paths between the vertices in a directed graph. For scheduled paths, each edge has a distance constraint, so that a path must contain edges with successively shorter distances. The paths returned are neither disjoint (i.e. more than one path can share a node) nor cyclic (i.e. paths which visit a vertex more than once).

4 Related Work
The survey papers of Deo and Pang [8], Quinn and Deo [9], and Schwartz and Stern [10], show that in related work the dominant interest is in single-shortest and k-shortest path problems, without the requirement that paths must be non-looping and link disjoint. The algorithms like Bellman [11], Dijkstra [12], and Moore [13] use centralized approach for calculating non-looping or disjoint path sets. Wagner proposed the algorithm for edge-sparse graphs with small integer link weights. The proposed multiple path computation algorithm in [14] searches for maximally disjoint with minimum overlapping of paths on each other so that link failures have minimum impact on the routing paths available.

Topkis in [15] proposed a centralized computation algorithm of k-shortest paths in a simple graph and the k-paths calculated need not be disjoint only initial links in the returned paths should be disjoint. Suurballe in [16] has found k-shortest disjoint paths. Grover’s algorithm [17] calculates the k-shortest link-disjoint paths in a distributed manner. This algorithm is mostly used as a restoration path set generator in self-healing networks. metaDijkstra proposed in [18] utilizes Dijkstra for calculation of k-paths. Once a path is found the network/graph is modified by deleting one link from the path then the new path is calculated. This procedure is repeated k times for k-shortest paths.

5 Fault Tolerant Transactional System – FTIMA Paths
What is the central part marked as ‘A’ in the Figure 3(a)? It is the network of different routers and the connecting links. The network is under the observation of multiple agents which monitor its state and maintain the HashMaps. Point A in the figure 3 part a, can be elaborated into the network shown in figure 3 part b or in a much bigger and complex network with more routers and links.

In our system (FTIMA) an Observer Agent (OA) is assigned to each possible path to route a transaction between the source and the target machine.

A mobile agent called Ping Agent is created for continuous monitoring of each path. Ping agent hops/moves from one router to the other in a timely fashion to check the availability of a certain path and updates OA about current path conditions. OA then decides the most appropriate path to route the transaction. Since ATM Network handles financial transaction, the security is our prime concern. The security and integrity of transactions in our system is ensured by triple desk and MD5 Hash code so that any kind of external intrusions can be stopped [21].

Similarly the reliability of the system is ensured by fault tolerance at agent level which is maintained...
by modified version of the sliding window protocol proposed by Summiya et al. in [1]. Proper path selection plays an important role for the *availability* of a transactional system [2] and thus is considered backbone for system level fault tolerance of a transactional system. Proper path selection topology is needed to guarantee timely and reliable routing of a transaction.

### 6 K – Shortest Paths

Dijkstra's algorithm is probably the best-known and most implemented shortest path algorithm. It is simple, easy to understand and implement, yet impressively efficient [20]. Dijkstra's algorithm finds the shortest path from a chosen source to a destination in a given graph. Dijkstra's algorithm partitions vertices in two distinct sets, the set of *unsettled* vertices and the set of *settled* vertices. Initially all vertices are unsettled, and the algorithm ends once all vertices are in the settled set. A vertex is considered settled, and moved from the unsettled set to the settled set, once its shortest distance from the source has been found. The algorithm is so powerful that it finds all shortest paths from the source to all destinations. This is known as the *single-source* shortest paths problem.

In the process of finding all shortest paths to all destinations, Dijkstra's algorithm can also be modified to compute, *k-Shortest Paths* from a given graph [2]. This variation of Dijkstra supports not one shortest but k shortest paths.

We can think of Dijkstra as generating a tree that spreads out from the root (start node). Similarly we can think of allowing k branches to overlap at a node instead of one. *K-Shortest Path Algorithm* is based on this concept [2]. It partitions vertices in two distinct sets similar as Dijkstra, the set of candidate vertices and the set of vertices taken for a path. Initially start vertex is added to the candidate set which continue to grow at each iteration i.e. all neighbors of the vertex under observation are added to the candidate queue except the nearest one.

For each iteration a vertex with lowest weight is taken from the priority queue to check weather it should be included in k shortest paths or not. The algorithm ends when k paths are found or when there is no candidate vertex in the queue.

According to the famous words of Niklaus Wirth

We all know that *algorithm + data structures = programs* [20]. The following data structures are used for this algorithm:

- **Paths**: HashMap stores the information of k paths.
- **TargetPaths**: Contains information of one of the k path which is currently under consideration.
- **Candidates**: It is a Priority Queue of a set of candidate vertices. Each vertex includes vertex itself, weight of vertex, and parent vertex → (vertex, weight, Parent Vertex)

**s**: Source, **t**: Target, **k** = No of Paths, **G**: Graph

K-ShortestPaths (s, t, k,G)

1. TargetPaths = φ
2. Paths.put ( t, TargetPaths )
3. Candidates = ( s, 0, null )
4. While (TargetPaths < k && Candidates != φ )
   - pi = candidates.poll ( )
   - Mpi = Paths.get ( pi.node)
   - If ( mpi == null)
     - Paths.put ( pi.node, new mpi )

---

**Figure (3)**

(a) Transactional system  
(b) An elaboration of Point A of Figure 3 (a)
End If
If ( mpi < k )
    mpi.addPathInfo(pi);
    While ( G.pi.AdjacentVertices ≠ φ )
        node = NextAdjacentVertex ( )
        candidates.add(node, pi.weight + edgeWeight, (pi.node,node), pi)
    End While
End If
End While
End Function

Complexity Analysis
Let graph have n nodes and e edges. The algorithm creates at most e*k candidates. Storing and removing them in the PriorityQueue takes O(log (k*e)). So we have a complexity of O(e*k*log(k*e)). [2, 19]

7 Test Case
Given a network G of n machines having ew positively weighted links. Now the problem is to find k number of non-cyclic shortest paths in increasing order of length between the source machine s and the target machine t. Paths returned need not be disjoint.
K-shortest paths are found to have alternate path options for routing a transaction so that if; 1) one path is down or 2) it has several transactions routing on it and load on it is equal to the threshold value θ, then that transaction is routed from the other less optimal path available. Now let us consider a simple scenario to further elaborate the above phenomena. Consider a network for a fault tolerant transactional system – FTIMA which is given in the figure 3(b).

Network
G(n, e) = G(10,13)
n = {M1, M2, M3, M4, M5, M6, M7, M8, M9, M10}
e_w = { (M1 M2)3, (M1 M3)2, (M2 M4)1, (M2 M8)5, (M3 M5)3, (M3 M6)4, (M4 M6)3, (M4 M8)3, (M5 M7)3, (M5 M9)10, (M6 M9)5, (M7 M10)2, (M8 M9)3, (M9 M10)1 }

After time t the system wants to route a transaction T4 from machine 1 (M1) to machine 10 (M10). Three optimal paths between M1 and M10 are computed by K-Shortest Paths. Each path has θ of 3 i.e. three or less transactions can be routed through a path at a time. The network conditions at t are shown in figure 4.

1. Paths P_i returned by K-Shortest Path Algorithm are
   P_1 = M1 → M3 → M5 → M7 → M10
   P_2 = M1 → M2 → M4 → M8 → M9 → M10
   P_3 = M1 → M2 → M8 → M9 → M10

2. Weight W_i of each path is
   W_1 = 12
   W_2 = 13
   W_3 = 14

According to the current network conditions T4 cannot be routed from path P_1 as OA_{P_1} (observer agent of path 1) has information that the load on this path is already equal to θ and it cannot bear more transactions so the Transaction Manager TM cannot schedule more transactions for P_1 until the load has decreased i.e. transactions already on this path have completed successfully.

Figure (4) Sample network of a fault tolerant transactional system – FTIMA, which is in its Initial conditions.

The next second optimal route is P_2 but ping agent of this route has informed OA_{P_2} that a link is down in P_2 so no transaction can be routed through this path. OA_{P_1} informs the system that the third optimal path P_3 is clear and the transaction can be routed from this path. This whole scenario indicates the need of multiple path options for fault tolerance in a distributed transactional system. It also guarantees system’s availability and reliability as a transaction is not routed on unavailable and insecure path i.e. having greater load of transactional traffic and thus increased probability of packet loss.
8 Conclusion
In this paper we proposed a fault tolerant routing framework for an ATM network to make better decisions in selection of routes before using them. In these systems we need a path which is highly reliable and provides timely service. We have observed through simulated experiments that the proposed approach yields optimal performance using k paths instead of one.

References: