International Journal of Smart Sensor and Adhoc Network

Volume 1 | Issue 2 Article 9

2011

Reliable Routings in Networks with Generalized Link Failure Events

Shailesh Kumar
Department of CS&E, Sri Siddhartha Institute of Technology, Tumkur, India, kavankvsit@gmail.com

M. SIDDAPPA Dr.
Department of CS&E, Sri Siddhartha Institute of Technology, Tumkur, India, msiddu_ssit@rediffmail.com

Reliable Routings in Networks with Generalized Link Failure Events

Shailesh Kumar[1], Dr. M.SIDDAPPA[2]
Department of CS&E, Sri Siddhartha Institute of Technology, Tumkur, India
E-mail: kavankvsit@gmail.com[1], msiddu_ssit@rediffmail.com[2]

Abstract- ***Traditional methods for ensuring reliable transmissions in circuit- switched networks rely on the pre computation of a backup path for every working path or for every network link These methods work fine as long as the network experiences only single link failures. They do not guarantee undisturbed communication, however, in the case of multiple link failures. Such failures are not seldom and often are correlated: a single failure in the physical network (a cut in the conduit carrying wiring or fibers used for several links) results in several failures in the abstract network layer (see for a discussion on multiple link failures).***

This type of link failures can be modeled using the notion of generalized failure events. A single generalized failure leads to the failure of several links in the network. Links that belong to the same failure even are also said to be in the same shared risk link group. Recent research has focused on the problem of computing, for a given pair of nodes, two risk-disjoint paths, i.e., two paths that do not share links that belong to the same generalized failure event.

I. INTRODUCTION

We study routing problems in networks that require guaranteed reliability against multiple correlated link failures. We consider two different routing objectives: The first ensures "local reliability," i.e., the goal is to route so that each connection in the network is as reliable as possible. The second ensures "global reliability," i.e., the goal is to route so that as few as possible connections are affected by any possible failure. We exhibit a trade-off between the two objectives and resolve their complexity and approximability for several classes of networks. Furthermore, we propose approximation algorithms and heuristics. We perform experiments to evaluate the heuristics against optimal solutions that are obtained using an integer linear programming solver. We also investigate up to what degree the routing trade-offs occur in randomly generated instances.

1.1. Objective

As high-speed networks become widely deployed and more commercial services depend on them, it is extremely important to provide reliable connections to the end users. In circuit-switched networks, connections are established by reserving resources along end-to-end paths that are kept up for the duration of the communication. Such networks are, for example, the so-called all-optical networks [1]. In all-optical networks, connections are established through light-paths and several light-paths are multiplexed in the same optical fiber that can carry data at rates of the magnitude of Terabits/sec. In these networks, reliable communications are even more crucial since a short network outage can lead to massive data losses and can affect many connections.

1.2 Motivation

Traditional methods for ensuring reliable transmissions in circuit- switched networks rely on the precomputation of a backup path for every working path or for every network link These methods work fine as long as the network experiences only single link failures. They do not guarantee undisturbed communication, however, in the case of multiple link failures. Such failures are not seldom and often are correlated: a single failure in the physical network (a cut in the conduit carrying wiring or fibers used for several links) results in several failures in the abstract network layer (see for a discussion on multiple link failures).

1.3 Problem statement

This type of link failures can be modeled using the notion of generalized failure events. A single generalized failure leads to the failure of several links in the network. Links that belong to the same failure even are also said to be in the same shared risk link group. Recent research has focused on the problem of computing, for a given pair of nodes, two risk-disjoint paths, i.e., two paths that do not share links that belong to the same generalized failure event.

1.4 Solution to the problem

This is the so-called "diverse routing problem." This problem and variations of it have been studied in this paper, we consider the problem of computing reliable routings in networks with generalized failure events under two different notions of reliability. In what we call "local reliability," we seek routings in which each connection spans as few as possible different failure events. In a sense, we minimize the failure probability of each connection, assuming that all failure events occur equally likely. In what we call "global reliability," we seek routings in which any single failure event affects as few as possible connections. Under this objective, we are interested in minimizing the distortion to the network operation in case of a failure event.

2. Literature survey

2.1. Existing System

- Existing schemes do not consider a long-range dependence property which is an important characteristic of the current internet traffic
- if we calculate the effective bandwidth just based on the parameters of long-range dependent traffic considering some QoS
- Such as loss probability, the utilization of the bandwidth can be very low due to huge rate fluctuation

- TCP performance degrades significantly in Mobile Ad hoc Networks due to the packet losses. Most of these packet losses result from the Route failures due to network mobility.

- TCP assumes such losses occur because of congestion, thus invokes congestion control mechanisms such as decreasing congestion windows, raising timeout, etc, thus greatly reduce TCP throughput.
- However, after a link failure is detected, several packets will be dropped from the network interface queue; TCP will time out because of these packet losses, as well as for Acknowledgement losses caused by route failures.
- There is no intimation information regarding about to the failure links to the Node from its neighboring Node's. So that the Source Node cannot able to make the Route Decision's at the time of data transfer.

2.2. Limitations

- The Stale routes causes packet losses if packets cannot be salvaged by intermediate nodes
- The stale routes increases packet delivery latency, since the MAC layer goes through multiple retransmissions before concluding a link failure
- Use Adaptive time out mechanisms
- If the cache size is set large, more stale routes will stay in caches because FIFO replacement becomes less effective.

2.3. Proposed System

- Prior work in LSR used heuristics with ad hoc parameters to predict the lifetime of a link or a route. However, heuristics cannot accurately estimate timeouts because topology changes are unpredictable.
- Prior researches have proposed to provide link failure feedback to TCP so that TCP can avoid responding to route failures as if congestion had occurred.
- We propose proactively disseminating the broken link information to the nodes that have that link in their caches. We define a new cache structure called a cache table and present a distributed cache update algorithm. Each node maintains in its cache table the Information necessary for cache updates.
- The Source Node has the information regarding about the Destination and the Intermediate Node links failure, So that it is useful from Packet loss and reduce the latency time while data transfer throughout the Network.

2.4. Advantages of Proposed system

- We defined a new cache structure called a cache table to maintain the information necessary for cache updates.
- We presented a distributed cache update algorithm that uses the local information kept by each border node to send message through the route which has maximum bandwidth available.
- The algorithm enables LSR to adapt quickly to topology changes.
- Fully distributed operation.
- Less flooding traffic during the dynamic route discovery process.
- Explicit exploitation of uni-directional links that would otherwise be unused.
- Broken routes could be repaired locally without rediscovery.
- Sub-optimal routes could be shortened as they are used.

3. Implementation

The modules that are included in this project are

- **Request discovery**
- **Accuracy of routes**
- **Packet forwarding**
- **Storing new paths**

3.1 Design Overview

On-demand Route Maintenance results in delayed awareness of mobility, because a node is not notified when a cached route breaks until it uses the route to send packets. We classify a cached route into three types:

pre-active, if a route has not been used;
active, if a route is being used;
post-active, if a route was used before but now is not.

It is not necessary to detect whether a route is active or post-active, but these terms help clarify the cache staleness issue. Stale pre-active and post-active routes will not be detected until they are used. Due to the use of responding to ROUTE REQUESTS with cached routes, stale routes may be quickly propagated to the caches of other nodes. Thus, pre-active and post-active routes are important sources of cache staleness.

When a node detects a link failure, our goal is to notify all reachable nodes that have cached that link to update their caches. To achieve this goal, the node detecting a link failure needs to know which nodes have cached the broken link and needs to notify such nodes efficiently. This goal is very challenging because of mobility and the fast propagation of routing information.

Our solution is to keep track of topology propagation state in a distributed manner. Topology propagation state means which node has cached which link. In a cache table, a node not only stores routes but also maintain two types of information for each route:

(1) How well routing information is synchronized among nodes on a route.

(2) Which neighbor has learned which links through a ROUTE REPLY. Each node gathers such information during route discoveries and data transmission.

The two types of information are sufficient, because *each* node knows for each cached link which neighbors have that link in their caches. Each entry in the cache table contains a field called *DataPackets*. This field records whether a node has forwarded 0, 1, or 2 data packets. A node knows how well routing information is synchronized through the *first* data packet.

When forwarding a ROUTE REPLY, a node caches only the downstream links; thus, its downstream nodes did not cache the first downstream link through this ROUTE REPLY. When receiving the first data packet, the node knows that upstream nodes have cached all downstream links. The node adds the upstream links to the route consisting of the downstream links. Thus, when a downstream link is broken, the node knows which upstream node needs to be notified.

The node also sets DataPackets to 1 before it forwards the first data packet to the next hop. If the node can successfully deliver this packet, it is highly likely that the downstream nodes will cache the first downstream link; otherwise, they will not cache the link through forwarding packets with this route. Thus, if DataPackets in an entry is 1 and the route is the same as the source route in the packet encountering a link failure, downstream nodes did not cache the link. However, if DataPackets is 1 and the route is different from the source route in the packet, downstream nodes cached the link when the first data packet traversed the route. If DataPackets is 2, then downstream nodes also cached the link, whether the route is the same as the source route in the packet. Each entry in the cache table contains a field called ReplyRecord. This field records which neighbor learned which links through a ROUTE REPLY. Before forwarding a ROUTE REPLY, a node records the neighbor to which the ROUTE REPLY is sent and the downstream links as an entry. Thus, when an entry contains a broken link, the node will know which neighbor needs to be notified. The algorithm uses the information kept by each node to achieve distributed cache updating.

When a node detects a link failure while forwarding a packet, the algorithm checks the *DataPackets* field of the cache entries containing the broken link:

(1) If it is 0, indicating that the node has not forwarded any data packet using the route, then no downstream nodes need to be notified because they did not cache the broken link.

(2) If it is 1 and the route being examined is the same as the source route in the packet, indicating that the packet is the first data packet, then no downstream nodes need to be notified but all upstream nodes do.

(3) If it is 1 and the route being examined is different from the source route in the packet, then both upstream and downstream nodes need to be notified, because the first data packet has traversed the route.

(4) If it is 2, then both upstream and downstream nodes need to be notified, because at least one data packet has traversed the route.

The algorithm notifies the closest upstream and/or downstream nodes and the neighbors that learned the broken link through ROUTE REPLIES. When a node receives a notification, the algorithm notifies selected neighbors: upstream and/or downstream neighbors, and other neighbors that have cached the broken link through ROUTE REPLIES. Thus, the broken link information will be quickly propagated to all reachable nodes that have that link in their caches.

The Definition of a Cache Table

We design a cache table that has no capacity limit. Without capacity limit allows DSR to store all discovered routes and thus reduces route discoveries. The cache size increases as new routes are discovered and decreases as stale routes are removed.

There are four fields in a cache table entry:

Route: It stores the links starting from the current node to a destination or from a source to a destination.

SourceDestination: It is the source and destination pair.

DataPackets: It records whether the current node has forwarded 0, 1, or 2 data packets. It is 0 initially, incremented to 1 when the node forwards the first data packet, and incremented to 2 when it forwards the second data packet.

ReplyRecord: This field may contain multiple entries and has no capacity limit.

A ReplyRecord entry has two fields: the neighbor to which a ROUTE REPLY is forwarded and the route starting from the current node to a destination. A *ReplyRecord* entry will be removed in two cases: when the second field contains a broken link, and when the concatenation of the two fields is a sub-route of the source route, which starts from the previous node in the source route to the destination of the data packet.

3. Advantages

- We defined a new cache structure called a cache table to maintain the information necessary for cache updates.
- We presented a distributed cache update algorithm that uses the local information kept by each border node to send message through the route which has maximum bandwidth available.
- The algorithm enables LSR to adapt quickly to topology changes.
- Fully distributed operation.
- Less flooding traffic during the dynamic route discovery process.
- Explicit exploitation of uni-directional links that would otherwise be unused.
- Broken routes could be repaired locally without rediscovery.
- Sub-optimal routes could be shortened as they are used.

4. REFERENCE

[1] Esteban L. Vall´es, Andres I. Vila Casado Mario Blaum, J. Villasenor and Richard D. Wesel "Hamming Codes Are Rate-Efficient Array Codes IEEE 2005 Es" IEEE 2005.

[2] Shenghao Yang and Raymond W. Yeung "Characterizations of Network Error Correction/Detection and Erasure Correction" IEEE 2007

[3] Pavel Kubalik,Petr Fiser,Hana Kubatova "Minimization of Hamming code generation in self. checking circuits"

[4] M. Blaum, P. Farrell, and H. van Tilborg., Handbook of Coding Theory. V.S.Press and W.C.Huffman. Elsevier Science B.V., 1998, ch. 22.

[5] R. Blahut, Theory and Practice of Error Control Codes. Addison- Wesley., 1983.

[6] S. Wicker, Error Control Systems for Digital Communication and Storage. Prentice Hall., 1995.

[7] J. Fan., "Array codes as low-density parity-check codes," Proceedings of Second Int. Symposium on Turbo Codes (ISTC 2000), pp. 423–426, Brest, France, Sept 4–7, 2000.

[8] K. Yang and T. Helleseth., "On the minimum distance of array codes as LDPC codes," IEEE Trans. on Information Theory, vol. 49, no. 12, pp. 3268–3271, Dec. 2003.

[9] R. G. Gallager, "Low-density parity-check codes," IRE Trans. Inform. Theory, vol. IT-8, pp. 21–28, Jan. 1962.

[10] M. Blaum and R. Roth., "New array codes for multiple phased burst correction," IEEE Trans. On Information Theory, vol. 39, no. 1, pp. 66– 67, Jan. 1993.

[11] D. Raphaeli., "The burst error correcting capabilities of a simple array code," IEEE Trans. on Information Theory, vol. 51, no. 2, pp. 722–728, Feb. 2005.

[12] D. A. Patterson, P. Chen, G. Gibson, and R. H. Katz., "Introduction to redundant arrays of inexpensive disks (RAID)," Proceedings of IEEE COMPCON Spring '89, pp. 112–117, March 1989.