Improving the Dependability of Computer Networks

by

Songkuk Kim

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science & Engineering)
in The University of Michigan
2006

To my family.

ACKNOWLEDGEMENTS

First of all, I would like to express my sincere gratitude to my adviser, Professor Kang G. Shin for his patient guidance, invaluable advice, and encouragement. He helped and supported me to find the right direction with insightful comments while allowing me to explore research areas for myself. He was a great mentor to me personally and academically. I would also like to thank Professors Farnam Jahanian, Mingyan Liu, and Achilleas Anastasopoulos for serving on my thesis committee and giving me constructive comments and suggestions.

I am thankful to former and present members of Real-Time Comuputing Laboratary including Seungjae Han, John Reumann, Hani Jamjoom, Padmanabhan Pillai, Mohamed El-Gendy, Hai Huang, Taejoon Park, Min-Gyu Cho, Jinkuk Lee, Sharath Kodase, and Daji Qiao for their productive discussion, kind collaboration, technical assistance and friendship. I am proud to be part of all these outstanding colleagues.

I am grateful to my parents and brothers for their love, faith, and assistance. My deepest thanks go to my wife, Minkyong Kim. She is my best friend, excellent colleague and true life-long lover. She is the source of my strength. Without her, I could not have finished this work. Finally, I would like to thank my daughter, Na-Yun, for her beautiful smiles.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

The Internet has expanded enormously during the last three decades. Its capability of tolerating failures of subsystems such as links, routers, and name servers has been improved but the current Internet still cannot guarantee timely recovery from such failures. Though people try to run safety- and business- critical applications such as remote medical services and business video conference over the Internet, mission-critical applications have not yet been deployed widely, because timely recovery, an essential requirement for mission-critical applications, is not guaranteed yet.

In packet-switched networks, *Quality of Service* (QoS) parameters—such as message throughput, end-to-end delay, and delay jitters—change frequently. To provide real-time communication of continuous media in these networks, we need to provide **performance**-related QoS guarantees. Among various QoS guarantees, bounding the end-to-end delay is the key requirement in real-time communication services.

In recent years, the rapid improvement in network connectivity and link capacity has expanded the application domain of real-time communication services. As the network gets larger and more complex, network failures occur more frequently. Thus, in addition to support for performance-related QoS, the application domain requires support for **Dependability**-QoS (DQoS), whose parameters include availability and reliability.

Paxson measured the stability of routes between selected nodes, and reported that up to 2.2% of probes experienced connection outages [34]. Labovitz *et al.* showed that about 50% of routes were available for less than 99.9% of the time [27, 26, 24, 25]. It is known that Border Gateway Protocol (BGP) may incur a persistent route oscillation problem when the network has a topology change [9, 29, 6]. The routing instability can result in packet loss, increased latency or the loss of connectivity in the Internet, which either cause failure of real-time applications or significantly degrade the performance of real-time communication.

To support DQoS, one must consider both *transient* and *persistent* network failures. A typical example of transient failures is temporary packet losses due to network congestion or data corruption. For example, reliable transport protocols can handle transient packet losses by acknowledgment and retransmission, or by forward-error-correction (FEC). Persistent failures refer to failures where the rate of correct message deliveries within a certain time interval is lower than a certain threshold, due to the breakdown of network components (links and switches). To handle persistent network failures, various dependability schemes have been proposed, which can be broadly classified as *reactive* or *proactive*.

Reactive schemes deal with failures *after* their occurrences [5]. To restore a real-time connection after a network-component failure, one has to set up a new connection without any faulty component. Since no resource is reserved *a priori* for the purpose of fault-tolerance, this method does not incur overhead in the absence of failures. However, it cannot give any guarantee on failure recovery due to a potential resource shortage or contention for recovery from failures. An extension [4] to this approach uses *delayed* retries to spread simultaneous recovery attempts. This method adds a random delay before starting each recovery process. If a recovery process fails, it retries along the same path with an exponential back-off. This method, however, may require several trials to succeed, thus delaying service resumption and increasing network traffic. The recovery can take several seconds or longer, especially in heavily-loaded networks.

Proactive schemes achieve dependability by reserving additional resources *a priori* in networks. Examples of proactive schemes include multiple-copy, dispersity routing, Single Failure Immune (SFI), and spare resource allocation methods. The multi-copy method [21, 35] sends multiple copies of a packet simultaneously through disjoint paths. This method attempts to achieve both timely and reliable delivery. Although it may be able to handle network failures without service disruption, it introduces a large resource overhead. It also cannot guarantee timely delivery due to its reliance on best-effort delivery of packet copies. Dispersity routing [3] combines forward-error-correction with multiple-copy transmissions. This allows a tradeoff between resource overhead and fault-tolerance capability. The SFI method [42] reserves additional resources in the vicinity of each real-time channel, and the failed components are detoured by using the reserved resources. The SFI method was extended to address special patterns of multiple failures in a hexagonal mesh network [43]. A *dependable real-time protocol* (DRTP) is another proactive scheme[13, 14, 15]. Each dependable real-time (DR) connection consists of one *primary* and one or more *backup* channels. When the primary channel of a DR-connection fails, one of its backups is promoted to become a new primary.

Although proactive schemes are effective in providing timely recovery, their resource overhead is higher than reactive schemes. To reduce the resource overhead of proactive schemes, a resource-sharing technique, *backup multiplexing*, was proposed [11, 13, 14, 15]. Instead of reserving all the resources necessary for each backup, only a small fraction of the necessary resources are reserved and then shared by all backups running the same link. The amount of total spare resources is determined on a hop-by-hop basis by considering the relation among all the backups traversing the same link.

Multiplexing backups on the same resources may hamper the dependability of DR-connections. The fault-tolerance of a DR-connection depends primarily on the probability of backup activation. Backup channels are said to have *conflicts* if they traverse the same link and their corresponding primaries share links. Since a backup cannot be activated when the spare resources are already taken by other backups, we may not activate conflicting backups that are multiplexed over the same spare resources when their corresponding primaries fail simultaneously. To provide better fault-tolerance, backup conflicts should be minimized.

Routing of backups is a key element of DRTP, but, despite its importance, it has not yet been addressed adequately. We proposed three different routing schemes for primary and backup channels, and evaluate their fault-tolerance and resource overhead.

The procedure of establishing and maintaining backup channels is complex and expensive. To reduce such costs of preparation and standby, we proposed a hybrid scheme, in which a *D-connection* is composed of a *primary* channel and a preplanned *backup path*.

When we establish a primary channel, we preselect a backup route to establish a detour channel immediately without contention upon a link failure. However, we do not perform any signaling along the backup route. Though a backup path is preselected, a backup channel is not established *a priori*. When a failure occurs on a link, the interrupted real-time connections try to reserve resources and to set up backup channels along the preselected backup routes. To prevent bandwidth shortage, we set aside a certain amount of bandwidth on each link as spare resources. The spare resources can be used to deliver best-effort traffic until backups channels are established. Because the amount of spare sources is fixed, we do not need to advertise the amount periodically.

In addition to routers and links, the Domain Name System (DNS) affects the availability of the Internet. DNS converts domain names to IP addresses. Because domain names are easier for humans to remember than IP addresses, most Internet services are exposed to users and applications with their domain names. To contact remote services, clients consult DNS to resolve domain names. As most Internet services are preceded by DNS query/reply transactions, it is important to shorten the DNS lookup delay; several approaches[10, 20] have been proposed with a focus on shortening the delay.

Besides the problem of shortening lookup delay, another important problem in DNS is providing fault-tolerance. The current DNS relies on replication for tolerating name-server failures. The replica-based approach may be effective when the failure or loss of one replica is disjoint from that of other replicas, such as OS-crashes, power-outages, and hardware-malfunction. This approach, however, cannot overcome malicious attacks. Because a replica system redirects incoming requests to other replicas when one replica fails, DNS requests generated by an attack will be redirected to other live replicas. If the amount of this traffic exceeds a threshold, a replica system may result in a chain of failures.

DNS has a tree-like hierarchy whose top consists of root name servers. Because the failure of root name servers disables the entire network, they are better protected than the lower-level (or local) name servers. The effect of failure of local name servers is less severe, but, nonetheless, the local machines cannot find IP address of domain names when their local name server fails; the local domain becomes virtually disconnected from the Internet. In this thesis, we focus on the availability of local name servers.

One simple solution for local name server failures is that every client has its own name server. This approach, however, loses the benefits of using cached DNS entries at local name servers: shortening client-perceived latency, reducing the DNS traffic to the outside world, and lowering the load of remote name servers by using local name servers. Because many clients visit the same sites, aggregating DNS queries/replies at local name servers improves the cache hit ratio. When a client requests the IP address of a certain machine, the DNS query/reply results are cached at the local name server with *Time-To-Live* (TTL), which specifies how long the information can be kept in the cache. When another client requests the IP address of the same machine before the corresponding entry's TTL expires, the local name server replies the client using the cached information. If each client has a name server at its local machine and does not share the results of DNS activities, the cache hit ratio will be lower. Thus, it is not desirable to deploy DNS servers in every client machine.

To maintain DNS service in the presence of failures, we propose the *DNS agent* that has three functionalities. In the absence of failures, the agent works as a local DNS caching agent. The agent relays DNS queries/replies between a client and local DNS server, and caches the results. When the agent has valid information for a DNS query, it replies to the query without relaying the query to a local name server. When local name servers are not available, an agent shares its cache contents with other agents in neighbor machines using a peer-to-peer system. If the agent cannot find valid information in the neighbors' caches, it consults remote name servers to resolve the query. DNS agents neither incur high overhead during preparation and standby nor overload outside machines when they take over the role of local name servers.

To evaluate our proposed scheme, we collected DNS traces from the name servers of the EECS Department at the University of Michigan, and conducted simulations using these traces. Using our DNS agent is always beneficial irrespective of whether DNS servers fail or not. In the absence of DNS server failures, we can reduce the load of name servers, reduce DNS lookup time, and save local network bandwidth. In case of local name server failures, we can continue DNS service, suppressing outgoing DNS traffic with a high cache hit ratio.

In this thesis, we investigated how to prepare a backup plan to improve the dependability of computer networks. In Chapter 2, we propose and evaluate three routing schemes for backup channels which do not need any central server. We explore the tradeoff between the amount of routing messages and the dependability of real-time channels with backups. In Chapter 3, we present a hybrid scheme that reduces the preparation and standby costs by setting aside a pool of spare resources and deferring resource allocation for each backup channel. We also modify one of the backup routing algorithms presented in Chapter 2 to reduce the amount of routing messages and the routing information kept in each router. In Chapter 4, we propose DNS agents which enable clients to use DNS service at the time of server failures. We analyze traces collected from the local name servers of the EECS Department at the University of Michigan and evaluate the dependability of DNS agents.

CHAPTER 2

Design and Evaluation of Routing Schemes for Dependable Real-Time Connections

A *dependable real-time protocol* (DRTP) is proposed in [13, 14, 15]. DRTP consists of the following four steps: (1) establishment of primary and backup channels, (2) detection of network failures, (3) failure reporting and channel switching, and (4) resource reconfiguration. How to route the primary and backup channels for a dependable real-time connection is a key element of DRTP, which, despite its importance, has not yet been addressed adequately. In this chapter, we propose three different routing schemes for backup channels, and comparatively evaluate their performances in terms of fault-tolerance and resource overhead.

2.1 Dependable Real-Time Protocol

Each dependable real-time (DR-) connection consists of one *primary* and one or more *backup* channels. Upon detection of a failure on the primary channel of a DR-connection, one of its backups is promoted to become a new primary. Since a backup is set up before occurrence of failures to the primary, it can be activated immediately upon detection of a failure on the primary, without the time-consuming, and sometimes unsuccessful, channel (re)-establishment process.

A backup channel does not carry any real-time traffic[1] until it is activated, and hence, it does not use resources in the absence of failures. However, a backup channel is not free, since it requires reservation of at least[2] as much resources as its primary channel, in order to provide the same QoS as its primary when it is activated. As a result, equipping each DR-connection even with a single backup which is disjoint from its primary reduces the network capacity in accommodating real-time connections by at least 50%, which is in many cases too expensive to be practically useful. To deal with this problem, a resource-sharing technique, called *backup multiplexing*, was introduced in DRTP [13, 14, 15]. The basic idea of backup multiplexing is that, on each link, instead of reserving all of the resources necessary for each backup, only a small fraction of the necessary resources are reserved and then shared by all backups running through the link, i.e., *overbooking* link resources for backups. The amount of total spare resources is determined on a hop-by-hop basis by considering the relation among all the backups traversing the same link.

The fault-tolerance of a DR-connection depends primarily on the probability of backup activation, or the probability of activated backups requiring no more than the reserved amount. The failure of backup activation can occur because the spare resources are over-booked by backups to reduce the resource overhead. Resource overbooking and sharing by backups would be perfectly acceptable if their corresponding primaries are completely disjoint, or do not overlap.

Backup channels are said to have *conflicts* if they traverse the same link and their corre-

[1]It may carry best-effort traffic, though.

[2]Note that a backup may run through a longer path than the corresponding primary.

sponding primaries overlap, or share link(s). Some of the conflicting backups multiplexed over the same spare resources may fail to be activated when their corresponding primaries fail (near-)simultaneously. To provide better fault-tolerance, backup conflicts should be minimized, and in the case of a conflict, more resources need to be reserved, if possible, not to multiplex conflicting backups.

Figure 2.1 illustrates the idea of backup multiplexing using a simple 3×3 mesh network. Each connection between two nodes has two unidirectional links. Although there are 24 uni-directional links, we only consider 13 of them in the following examples. There are three DR-connections, D_1, D_2 and D_3. The primary and backup channels of these connections are shown with solid and dashed arrows, respectively. In this example, we assume that only a single link can fail between two successive recovery actions. Consider link L_6 which is part of the routes of the backup channels B_1 and B_2. Because the primary channel P_1 and P_2 do not overlap, any single link failure can cause at most one of these backups to be activated. Thus, B_1 and B_2 will never contend for the reserved resources on L_6, and therefore, the backup multiplexing on L_6 successfully reduces the resource overhead without affecting the fault-tolerance capability. Now, let's consider link L_7 which is used by the backup channels B_1 and B_3. Since P_1 and P_3 overlap at L_{13}, if L_{13} fails, both DR-connections need to be switched to their backups. Hence, the resource needs on L_7 exceed the reserved amount, and L_7 can accommodate only one connection, say B_1. As a result, backup channel B_3 cannot be activated because of the lack of resources and DR-connection D_3 cannot be recovered from the failure of L_3. Therefore, the backup multiplexing on L_7 degrades the fault-tolerance capability.

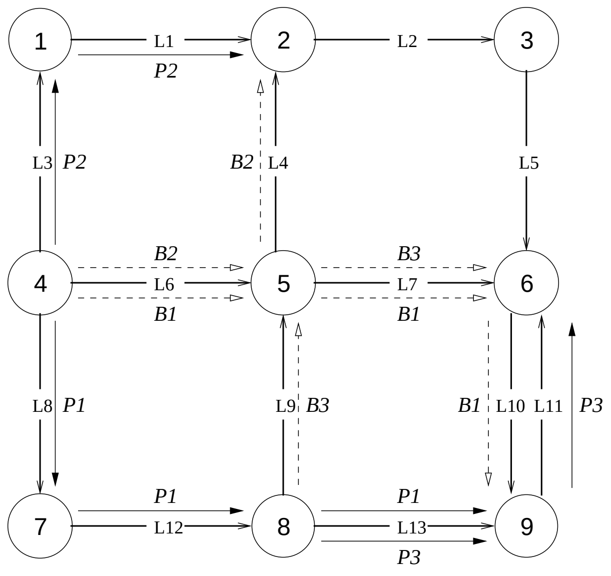


Figure 2.1: An example of backup multiplexing

Based on the above observations, one can see that routing channels under backup multiplexing has a significant bearing on the resulting fault-tolerance capability. An ideal backup channel B should (1) provide the same QoS as its primary upon its activation; (2) overlap minimally with its primary channel; and (3) overlap minimally with other backups whose primaries overlap with B's primary. Clearly, traditional routing algorithms [16] such as minimum-hop routing or maximum load-balanced routing, cannot solve the backup routing problem.

To find a backup route that meets these three requirements, one must know where primary channel paths run, where the corresponding backup paths are, and the amount of resources available on these paths. However, it is undesirable (for scalability reasons) to require every router to keep all this information. Especially, maintaining information on all DR-connections at each router is impractical because the required amount of information is O(number of DR-connections $\times$ average path length). Thus, we develop a mechanism that requires every router to maintain only *abridged* information.

2.1.1 Notation

We will use the following symbols/notation.

- N: the total number of links in the network.
- E: average node degree of the network.
- P_i: the primary channel of DR-connection D_i.
- B_i: the backup channel of DR-connection D_i.
- $LSET_r$: the set of links in route r.
- $PSET_i$: the set of primary routes whose backup routes go through link L_i, i.e., $\{P_i : L_i \in B_i\}$, where P_i and B_i are respectively primary and backup routes traversing link L_i.
- Accumulated Primary route Link Vector, $APLV_i$, whose j^{th} element, denoted by $a_{i,j}$, represents the total number of primary channels that traverse link L_j and whose backup channels go through link L_i. Then, $a_{i,j} = |\{P_k : P_k \in PSET_i$ and $L_j \in LSET_{P_k}\}|$,
- $\|APLV_i\|_1$: the L_1-norm of $ARLV_i$ which is defined as $\sum_{j=1}^{N} a_{i,j}$.
- SC_i: the number of backups on L_i that can be activated successfully using the spare resources.

2.1.2 DR-Connection Management

To support the DR-connection service, every router is equipped with a DR-connection manager which consists of two modules: one routes backup channels and the other multiplexes backups. The former exchanges and maintains the information necessary to select backup routes. We assume that a portion of network resources is set aside for DR-connections. The total amount of resources for DR-connections cannot exceed this portion, and these resources can be used for non-real-time traffic when they are not used by DR-connections. That is, the resources reserved for backups can be used for non-real-time traffic until they are activated.

Management of each DR-connection consists of the following four steps.

1. Select a primary route and reserve resources when a client/server node requests a DR-connection to be set up.

2. Find a backup route after successfully establishing the primary.

3. Send a *backup-path register* packet along the newly-selected path.

4. Release the resources of the primary and backup routes when the corresponding DR-connection is terminated.

Every router maintains $APLV$ for each of links attached to it, but the entire $APLV$s are not stored in each router's link-state database. $APLV$ is used for routing and multiplexing backups. To maintain $APLV$ for a link, a router needs $LSET$s of its $PSET$. Storing all $LSET$s may require large memory space, because an excessive number of backup channels can go through a single high-speed link. To cope with this problem, when a node sets up or releases a backup channel, it sends the $LSET$ of the corresponding primary route in a *backup-setup request*. When a router receives a backup-setup request, it checks the amount of available resources. The router registers this new backup in the backup channel table and updates $APLV$ for the link that the backup channel traverses using $LSET$. Finally, the router forwards the request to the next router in the backup path. When a router rejects the request for setting up a backup channel, it sends a *backup-release request* in which $LSET$ of the corresponding primary route is included.

2.2 Link-State Routing Schemes

A node can select a backup path that has minimum conflicts, if it has complete knowledge of $APLV$s for all the links in the network. The j^{th} element of $APLV_i$ represents the number of DR-connections whose backups and primaries traverse L_i and L_j, respectively. For the example network in Figure 2.1, we have $PSET_7 = \{P_1, P_3\}$, $LSET_{P_1} = \{L_8, L_{12}, L_{13}\}$, $LSET_{P_2} = \{L_1, L_3\}$, and $APLV_7 = (0,0,0,0,0,0,0,1,0,0,1,1,2)$. $APLV_7$ indicates that if L_7 is selected as a link of the backup route for a DR-connection whose primary channel goes through L_{12}, it will generate conflicts with two other backups. $APLV_i$ represents the number and the positions of backup conflicts that will occur when L_i is used as a backup path link. Thus, if a node has knowledge of all $APLV$s in the network, it can select a backup path that will create minimum conflicts. However, it is too costly for every node to acquire and maintain all $APLV$s, as there are N APLVs, each with N integers.

We therefore develop two link-state routing schemes that use an abridged form of $APLV$: (1) P-LSR that infers and exploits the probability of backup conflicts using

$\|APLV\|_1$; (2) D-LSR that uses a bit-vector form of $APLV$, called *Conflicts Vector*, to decide deterministically if a link has backup conflicts.

2.2.1 P-LSR: Probabilistic Avoidance of Backup Conflicts

The idea behind P-LSR is that the probability of link L_i's backup conflicts rises as the number of links in $PSET_i$ — which is equal to $\|APLV_i\|_1$ — increases. Without knowing where primary routes run, it is very logical to select a link with a smaller $\|APLV\|_1$ to minimize backup conflicts and maximize the probability of successful backup activation on L_i. For the example network in Figure 2.1, $\|APLV_2\|_1 = 0$, $\|APLV_4\|_1 = 2$, and $\|APLV_7\|_1 = 5$. As a link for a backup route, L_2 is preferable to L_4 or L_7.

To activate a backup, B_x, on L_i without any conflict, P_k should be disjoint from all the primary routes whose backups traverse L_i, i.e., $LSET_{P_x} \cap \{\bigcup_{P_j \in PSET_i} LSET_{P_j}\} = \emptyset$. Let $\phi_{B_x,i}$ denote the probability of successful activation of B_k on L_i. Then, $\phi_{B_x,i}$ can be calculated as:

$$\phi_{B_x,i} = \left(\frac{N - |LSET_{P_x}|}{N}\right)^{\|APLV_i\|_1}. \tag{2.1}$$

Since our goal is to select a backup route that has the maximum probability of successfully activation, we need to know the relation between the probability of backup activation and links' $\|APLV\|$ in the backup route. The relation can be derived easily as follows. Consider a DR-connection D_x whose primary channel P_x has already been established. Our goal is then to find the best backup route, B_x, by maximizing the probability of successful activation upon P_x's failure. The probability of successfully activating B_x, denoted by Φ_{B_x}, can be calculated by

$$\Phi_{B_x} = \prod_{L_i \in LSET_{B_x}} \phi_{B_x,i}. \tag{2.2}$$

Since the $\log$ function is monotone, maximizing Φ_{B_x} is equivalent to maximizing $\log \Phi_{B_x}$ where

$$\begin{aligned} \log \Phi_{B_x} &= \log \left(\prod_{L_i \in LSET_{B_x}} \phi_{B_x,i} \right) && (2.3) \\ &= \sum_{L_i \in LSET_{B_x}} \log \left(M^{\|APLV_i\|_1} \right) \\ &= (\log M) \times \left(\sum_{L_i \in LSET_{B_x}} \|APLV_i\|_1 \right) && (2.4) \end{aligned}$$

where $M = \frac{N-|LSET_{P_x}|}{N}$. Since $\log M$ is a negative constant, a path B_x that has minimum $\sum_{\ell \in LSET_{B_x}} \|APLV_\ell\|_1$ will maximize the probability of backup activation. Such a path can be found using the Dijkstra's algorithm by assigning $\|APLV_i\|_1$ as the cost for link L_i.

$\|APLV\|_1$ and the available bandwidth (the sum of the un-allocated bandwidth and the spare bandwidth shared by the backup channels) are stored in each link-state database. To select a backup path after establishing a primary channel, a router assigns C_i as the cost of link L_i, and chooses a minimum-cost path using the Dijkstra's algorithm, where

$$C_i = Q + \|APLV_i\|_1 + \varepsilon. \tag{2.5}$$

Q is a very large constant ($\gg max(APLV_i)$) if P_x traverses L_i or the available bandwidth is smaller than the QoS requirement; 0 otherwise. A small positive constant, ε ($\ll 1$), is used to select the shortest route as the backup path if there are several candidate routes with the same degree of channel overlapping. The resulting path will be the shortest backup

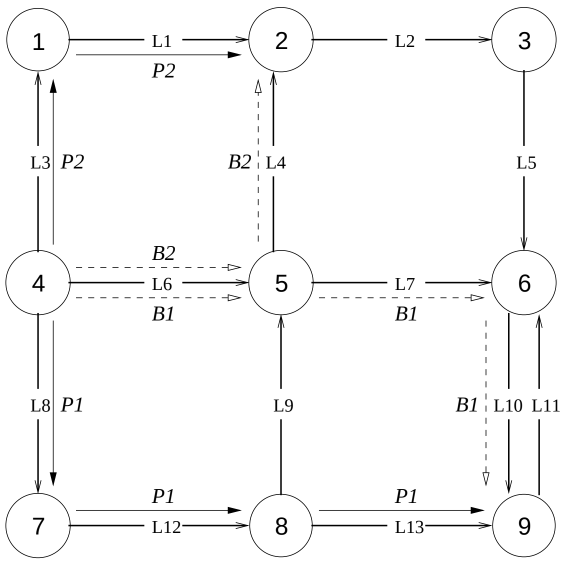


Figure 2.2: An example network topology

route that meets the QoS requirement, minimally overlaps with its corresponding primary channel route, and maximizes the probability of successful activation.

2.2.2 D-LSR: Deterministic Avoidance of Backup Conflicts Using Conflict-Vector

As defined earlier, $APLV$ contains information on the number and position of backup conflicts. The position information is more important than the number information. D-LSR uses a simple data structure, *Conflict-Vector* (CV), that shows only the location of backup conflicts. The CV of link L_i, denoted by CV_i, is an N-element bit-vector, the j^{th} element of which, $c_{i,j}$, is 1 if the j^{th} element of $APLV_i$, $a_{i,j}$,$>$ 0; 0 otherwise. Thus, $c_{i,j} = 1$ if and only if there is at least one primary channel running through L_j whose backup traverses L_i.

A simple example is given in Figure 2.2 in which there are two DR-connections D_1 and D_2. Since both backup channels, B_1 and B_2, go through L_6, $PSET_6 = \{P_1, P_2\}$. From $LSET_{P_1}$ and $LSET_{P_2}$, one can easily compute $APLV_6$ and $CV_6 =$ (1,0,1,0,0,0,0,1,0,0,0,1,1).

The CVs for the other links can be computed similarly.

When a node attempts to choose a link for backup, B_x, after establishing its corresponding primary P_x, the node can use CV_i and $LSET_{P_x}$ to check if L_i creates backup conflicts. If $L_j \in LSET_{P_x}$ and $c_{i,j} = 1$, L_i will introduce backup conflicts. Thus, the node selects L_i such that $\sum_{L_j \in LSET_{P_x}} c_{i,j}$ is minimum. To choose a backup route with minimum conflicts while meeting the QoS requirement, the node assigns L_i the link cost C_i and selects the minimum-cost route using the Dijkstra's algorithm, where

$$C_i = Q + \sum_{L_j \in LSET_{P_x}} c_{i,j} + \varepsilon. \tag{2.6}$$

Q and ε are added for the same reason as in P-LSR. In D-LSR, CVs and the available bandwidths are stored in the link-state database.

Consider the example network in Figure 2.3, where two DR-connections are established and node 8 selects the backup for D_3 whose primary is running through L_{13} and L_{11}. Suppose the links have enough bandwidths available to provide the required QoS. The calculated link costs are given in Table 2.1. As shown in Figure 2.3, (L_9, L_4, L_2, L_5) is selected as the backup channel route, B_3^*, of the DR-connection. Note that if L_{13} fails, both connections a and c fail simultaneously. However, since the backup routes are disjoint, both connections can successfully recover from the link failure. As compared to B_3 in Figure 2.1, B_3^* offers better fault-tolerance than B_3, although B_3^* has a larger hop distance.

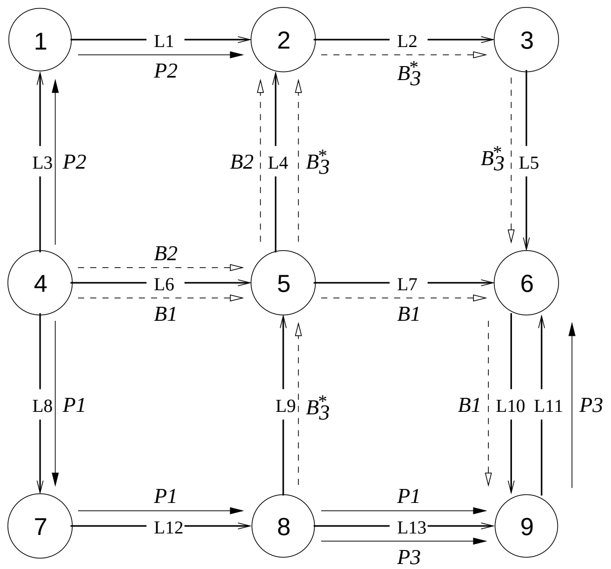


Figure 2.3: Backup route selection for DR-connection D_3

L_i	$c_{i,12}$	$c_{i,13}$	C_i
1	0	0	ε
2	0	0	ε
3	0	0	ε
4	0	0	ε
5	0	0	ε
6	1	0	$1+\varepsilon$
7	1	0	$1+\varepsilon$
8	0	0	ε
9	0	0	ε
10	0	0	$1+\varepsilon$
11	0	0	$Q+\varepsilon$
12	0	0	ε
13	0	0	$Q+\varepsilon$

Table 2.1: Calculated link costs

2.3 Routing with Bounded Flooding

Link-state routing is easy to implement, but the extended link-state packet requires a larger packet size and introduces additional routing traffic. To deal with this problem, we propose a different on-demand routing scheme based on bounded flooding, which was originally proposed by Kweon and Shin [23] for QoS (not DoS) routing.

Suppose a destination (client) node requests a DR-connection service from the source (server) node, and specifies its QoS requirement by indicating the minimum desired link bandwidth of the connection. In order to establish such a DR-connection, the source node floods a special channel-discovery packet (CDP) towards the destination. To reduce the resulting overhead, the source node limits the number of hops each CDP can take before reaching the destination. That is, when an intermediate node receives a CDP, it will decide whether to forward the packet to one of its neighbors by checking if the minimum-hop route via that neighbor can lead the CDP to the destination within the source-specified hop-count limit. This scheme can be viewed as *bounded flooding*. The destination node is responsible for selecting the best primary and backup routes for the real-time connection based on the flooded information. Before proceeding to the proposed algorithm, it is necessary to introduce the relevant data structures.

2.3.1 Data Structures and Notation

Each network node maintains a distance table (DT) and a link-state table (LST). Let N_i denote the set of node i's neighbors. Hop count is used to build distance tables, although any other distance metric can be used. The distance table at node i is a 2-dimensional matrix containing, for each destination j and for each neighbor $k \in N_i$, the minimum hop count from i to j via k, which is denoted by $D^i_{j,k}$. So, the minimum distance from node i to destination j is

$$D^i_j = \min_{k \in N_i} D^i_{j,k} + 1. \tag{2.7}$$

The minimum hop count can be calculated using the Dijkstra's algorithm or the Bellman-Ford distance-vector algorithm. The distance tables are updated only upon change of the network topology. The link-state table at node i contains, for each outgoing link (i, k), the following fields:

- *total_bw*: total bandwidth reserved for DR-connections.
- *prime_bw*: bandwidth consumed by all the primary channels.
- *share_bw*: bandwidth shared by all the backup channels.

Moreover, the degree of backup multiplexing on a link ℓ is characterized by the *link over-booking percentage*, which is defined as the ratio of the resource needs for all the backup channels through link ℓ to *share_bw(ℓ)*.

To establish a DR-connection from the source i to the destination j, the source initiates the bounded flooding of a CDP, which contains the following fields:

- *srce_id (dest_id)*: an integer which uniquely identifies the source (destination) node.
- *conn_id*: a pair of integers which uniquely identifies a DR-connection. Each connection identifier consists of two parts: *srce_id* and a connection time-stamp (unique with respect to the same source node). Such composition of connection identifiers ensures their uniqueness throughout the network.
- *hc_limit*: maximum hop count that the CDP can take before reaching the destination.
- *hc_curr*: hop count of the route taken thus far by the CDP to reach the current node.
- *bw_req*: bandwidth requested for the DR-connection.
- *max_ok*: maximum link over-booking percentage along the route to the current node.

- *list* of nodes that the CDP has traversed so far. Every time the CDP is forwarded, the current node is appended to *list*. This information is needed for the destination to select the best routes for the primary and backup channels of the connection, and to guarantee loop-free flooding.

The flooding bound of the CDP is specified by *hc_limit*, which is equal to $\rho \times D_j^i + \mu$, where $\rho \geq 1$ and $\mu \geq 0$. In order to improve the chance of granting the requested DR-connection, multiple routes must be given opportunities to run the connection over them. Therefore, the values of λ and μ are determined by making a tradeoff between the routing overhead and the connection-acceptance probability.

Since the information of established connections is necessary for a new connection's admission test, each node has to maintain two "transient" tables: *pending connection table* (PCT) and *established connection table* (ECT). Each entry of a PCT represents a connection request passed through this node, and consists of four fields:

- *conn_id*: the connection identifier.
- *bw_req*: the connection bandwidth requested.
- *min_dist*: hop count of the shortest route taken by some CDP to the current node.
- *time_out*: a real number which specifies this entry's time to live. Upon expiration of the timer, this entry is no longer valid and thus deleted from the PCT. In order to prevent a false deletion, *time_out* must be no less than the average link delay multiplied by the hop-count limit.

Each entry of an ECT represents a DR-connection whose primary or backup channel goes through this node, and contains the following fields:

- *conn_id*: the connection identifier.
- *bw_req*: the connection bandwidth requested.
- *type*: '1' indicates a primary channel and '0' indicates a backup.
- *pred_id*: the identifier of the predecessor node along the channel route.
- *succ_id*: the identifier of the successor node along the channel route.

Besides PCT and ECT, each node has to maintain a set of candidate route tables (CRTs), one for each outstanding connection request destined for this node. The function of these tables is to allow the destination to choose the best primary and backup routes among those routes which the CDPs have safely traversed to reach the destination. Each entry of a CRT represents one candidate route for the corresponding connection request, and contains the following fields:

- *max_ok*: maximum link over-booking percentage along the candidate route.
- *hop_count*: hop count of the candidate route. It is copied from the *hc_curr* field of the CDP.
- *list* of nodes on the candidate route.

2.3.2 Action by the Source Node

The destination node initiates a connection request and uni-casts the request message to the source node. Upon receiving the request message, the source node i composes a

CDP m, and performs the following tests for each of its neighbors $k \in N_i$:

Distance test:

$$D^i_{dest\_id,k} + 1 \leq hc\_limit(m). \tag{2.8}$$

Bandwidth test:

$$bw\_req(m) \leq total\_bw(i,k) - prime\_bw(i,k). \tag{2.9}$$

If node k passes both tests, node i updates and forwards the packet to node k. The CDP is updated by re-calculating *max_ok(m)*, increasing *hc_curr(m)* by one, and appending i to *list(m)*. Besides, node i updates its PCT by adding a new entry. If none of the neighbors passes both tests, the CDP is discarded.

2.3.3 Action by an Intermediate Node

Upon receiving a CDP m, node i checks if *dest_id(m)* matches its own identifier. If *dest_id(m)* $\neq i$, this intermediate node i checks if *conn_id(m)* appears in its PCT, or if at least one CDP for the same connection request has already been flooded through the node. If this is the first CDP passing by, node i performs the following tests for each of its neighbors $k \in N_i$:

Distance test:

$$hc\_curr(m) + D^i_{dest\_id,k} + 1 \leq hc\_limit(m). \tag{2.10}$$

Loop-freedom test:

$$k \notin list(m). \tag{2.11}$$

Bandwidth test:

$$bw\_req(m) \leq total\_bw(i,k) - prime\_bw(i,k). \tag{2.12}$$

If node k passes all these tests, node i updates and forwards the packet to k. The CDP is updated in the same way as in the source node. Besides, node i updates its PCT by adding a new entry. If none of the neighbors passes all the tests, the CDP is discarded.

However, if node i has already received at least one copy of the CDP, then before executing the above three tests, node i performs an additional test on the incoming CDP:

Valid-detour test:

$$hc\_curr(m) \leq \alpha \times min\_dist(conn\_id(m)) + \beta. \tag{2.13}$$

The CDP won't be flooded if it fails the test. By using this additional "valid-detour test," where α and β are two pre-determined parameters, we further reduce the number of CDPs.

2.3.4 Action by the Destination Node

If *dest_id(*m*)* $= i$, the destination node i checks if *conn_id(*m*)* appears in one of its CRTs, or if at least one CDP for the same connection request has already reached the destination. If yes, node i updates the CRT by filling a new entry based on the information provided in this CDP. Otherwise, node i creates a new CRT for this connection request, and sets a timer which is no less than the average link delay multiplied by the hop-count limit. Upon expiration of the timer, any outstanding CDP corresponding to this connection request is no longer valid and has been discarded by some intermediate node, then node i starts the route selection process and then the route confirmation process.

Among all the candidate routes listed in a CRT, only those with zero *max_ok* value might be selected as the primary channel route. The destination node chooses the shortest route (i.e., the one with the smallest *hop_count* value) to be the primary channel route. All the remaining candidate routes in the CRT are eligible to be the backup channel route, and the destination node chooses the shortest one that minimally overlaps with the primary channel route. The destination node starts the route confirmation process for both primary and backup channels simultaneously. Confirmation messages are sent in the reverse direction of the primary and backup channel routes recorded in the CRT and the table is deleted or expires afterwards. Upon arrival of a confirmation message, each intermediate node along the route attempts to reserve the requested bandwidth for the DR-connection, and updates its LST and ECT.

However, the primary and/or backup channel routes with enough resources to meet the required QoS may not always exist. If the route selection process fails, or if an intermediate node fails to reserve the requested bandwidth for the DR-connection, a failure message will be sent towards the source and the destination. Upon receipt of a failure message, each intermediate node along the route to the destination frees the reserved bandwidth and updates its LST and ECT; the source will start the channel discovery process all over again.

When a DR-connection is terminated by an application program, the disconnection process is initiated by either the source or the destination. In this process, a channel-release packet, which has a data structure similar to the CDP's, is passed along the primary and backup channels of the DR-connection. Upon receipt of a channel-release packet, each intermediate node frees the reserved bandwidth and updates its LST and ECT.

2.4 Backup Multiplexing

A node's attempt to choose a backup route without any conflict may not always be successful. It is therefore possible to activate more than one backup on a link when a link failure occurs. For example, let $APLV_1 = (0, 1, 2, 1, 2)$. Then, if L_3 or L_5 fails, two DR-connections will attempt to activate their backups through L_1. If the spare resources reserved on L_1 can accommodate only one of the two DR-connections, one of the two will fail to activate its backup. To handle such a case, the DR-connection manager responsible for L_i has to reserve more spare resources. The conflicting backups are not multiplexed over the same spare resources.

The DR-connection manager for a link checks if more spare resources need to be reserved using the APLV and SC of the link, where SC_i is the number of backups that can be activated simultaneously using the spare resources. Since all DR-connections are assumed to require an identical amount of bandwidth, SC_i can be calculated by dividing the total spare bandwidth reserved on L_i by the bandwidth of a DR-connection. If any element of $APLV_i$ is larger than SC_i, at least two conflicting backups are multiplexed on the same spare resources. In this case, it is necessary to reserve more spare resources.

When a node receives a backup-setup request, the DR-connection manager of the node updates the $APLV$ of the link that the backup traverses using the $LSET$ of the corresponding primary (this information is included in the backup-setup request). Using the new $APLV$, the DR-connection manager can determine if it will multiplex the new backup on the current spare resources or if it will reserve more resources.

A DR-connection manager may not be able to increase spare resources due to the shortage of resources, even when the new backup has conflicts with other backups. In such a case, we have two choices: (1) reject the request, or (2) multiplex the new backup on the previously-reserved spare resources with other backups that the new backup has conflicts with. We opt to take the second approach. Although multiplexing conflicting backups degrades fault-tolerance, there is a chance that one of the conflicting backups may be rejected, or one of the primary channels on the same link may terminate before a link failure that will trigger activation of conflicting backups. If a primary channel is released, its resources will be returned to the pool of free resources, and the DR-connection managers assign these free resources to spare resources.

2.5 Evaluation

We have conducted an in-depth simulation study to comparatively evaluate the three proposed routing schemes in terms of fault-tolerance and overhead. In this study, we measured the probability of successfully establishing a DR-connection, and the fault-tolerance of established connections under various load conditions and network configurations. We also evaluated the routing overhead for discovering backup routes and their resource overhead.

2.5.1 The Simulation Model

In order to investigate the performance of the proposed routing schemes under different network configurations, we selected networks with 60 nodes and the average node-degrees (E) of 3 and 4. The networks are generated by using the **Waxman** topology generator [40]. Each node acts as a router or switch, and links are assumed to be bi-directional, with an

parameters	value	comments
N	60	number of nodes
E	3,4	average node degree
C	100 Mbps	link bandwidth capacity
λ	$\{0.2, 0.3 \cdots, 1.0\}$/sec	connection request arrival rate
bw_req	2.5 Mbps	bandwidth request
t_req	$20 \leq$ t_req < 60 min	connection lifetime

Table 2.2: Simulation parameters

identical bandwidth capacity (C) in both directions.

The simulation study uses two traffic patterns. One, denoted by unform traffic, is uniform random selection of source and destination nodes. The other, denoted by non-uniform traffic, is random pre-selection of 10 nodes as destinations for 50% of DR-connections. For simplicity, we assume that DR-connection requests arrive as a Poisson process with rate λ. Instead of using more realistic traffic models, we only consider simple traffic patterns, because our goal is to comparatively evaluate the proposed routing schemes, as opposed to providing absolute performance figures. Moreover, we assume that each connection requires a constant bandwidth (*bw_req*) and has a uniformly-distributed lifetime, *t_req*, between 20 and 60 minutes. The *network load* is defined as the total bandwidth reserved for all active real-time connections. Since we fix *bw_req* and *t_req* as constants, the network load depends only on the network capacity and the request arrival rate λ. The relevant parameters of the simulation are listed in Table 2.2, and the values are selected while keeping in mind the bandwidth and time constraints of typical video and audio applications.

In the simulation study, we use scenario files to record the connection request and release events under various *bw_req* and λ values, and compare the performance of the

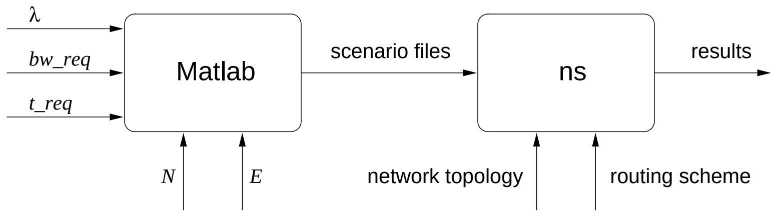


Figure 2.4: Simulation process

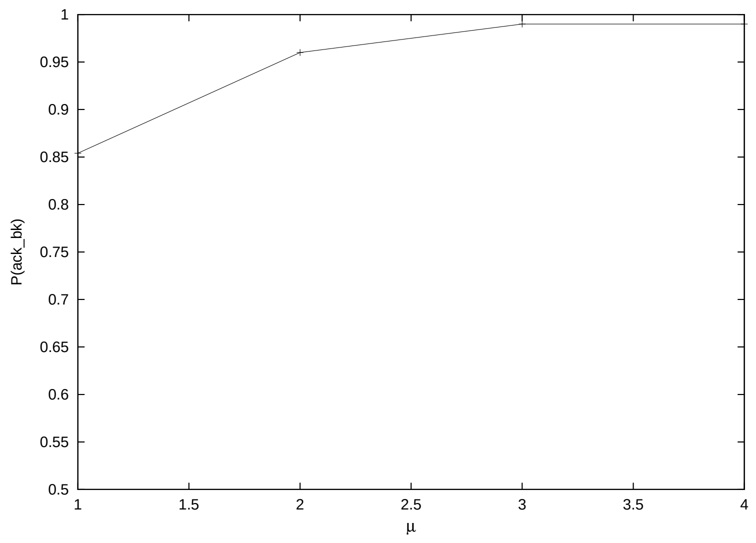


Figure 2.5: Bounded flooding with various flooding scopes

proposed schemes by simulating them using the same scenario file. Each simulation is run for the time duration of 2 hours. The scenario files are generated by **Matlab**, and the proposed routing schemes are implemented in, and simulated with, **ns**. Figure 2.4 illustrates the whole simulation process.

2.5.2 On Flooding Bound

The performance of the bounded flooding scheme depends on the bounding scope and the flooding packets. The bounding scope is limited by ρ and μ, and the number of flooding packets is restricted by α and β. We studied the impact of additive parameters μ and β, with the multiplicative parameters ρ and α set to 1.

We evaluate the performance of bounded flooding for different values of $\mu \in \{0, 1, 2, 3, 4\}$ and $\beta \in \{0, 1, 2\}$ with $\lambda = 0.05$. The results are plotted in Figure 2.5 for various μ values. P_{ack_bk} is the probability of successfully activating a backup channel if the corresponding primary channel is disabled by a single link failure. Establishing backups reduces the *number of D-connections* that the network can support, because the resources reserved for backups cannot be used for primary channels.

Increasing the flooding scope might increase the probability of finding the primary channel route for a particular connection request, but since the primary channels cannot be multiplexed, this might hurt the chance of some other connection requests in finding their primary channel routes. Hence, the number of D-connections is not affected significantly by the flooding scope. In contrast, due to backup multiplexing, increasing the flooding scope can increase the chance of finding a backup route that overlaps minimally with the primary route, and hence improves fault-tolerance. However, P_{ack_bk} is shown to remain almost unchanged for $\mu > 2$.

We measured the performance of the bounded flooding scheme for various valid-detour tests with a fixed flooding scope ($\mu = 2$). The performance remained almost same for all these tests for the following reason. If $\beta = 0$, then for each connection request, every intermediate node floods the channel-discovery packet only once, and these channel-discovery packet copies collected at the destination constitute the *basic candidate set*. When β increases, each additional candidate route must have a corresponding shorter route in the basic candidate set. Since in the proposed bounded flooding scheme, shorter routes are given priority to become the primary and backup routes, these additional candidate routes have little chance to be selected. Hence, even with a larger β, the primary and backup routes are still very likely to be selected from the basic candidate set.

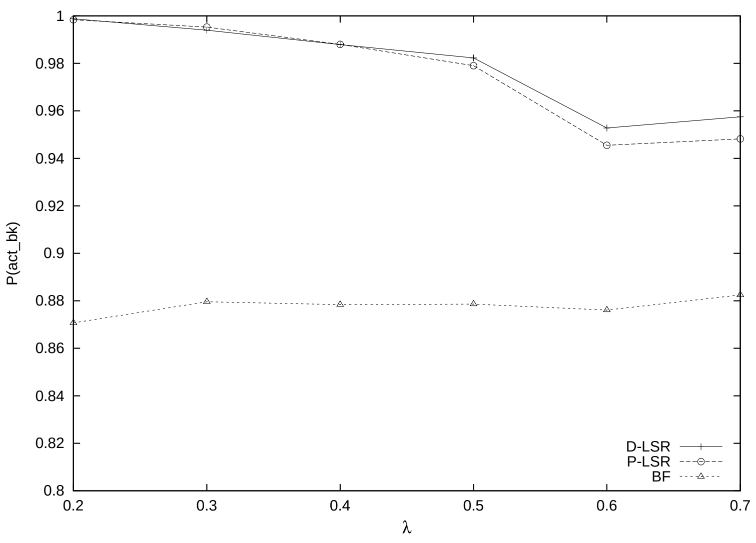
We summarize the results in these figures as: *(1) a larger flooding scope results in higher routing overhead, but better fault-tolerance, and (2) the "valid-detour test" with* $\beta = 0$ *is a good heuristic to reduce the number of flooding packets without degrading the fault-tolerance.*

2.5.3 Performance Comparison

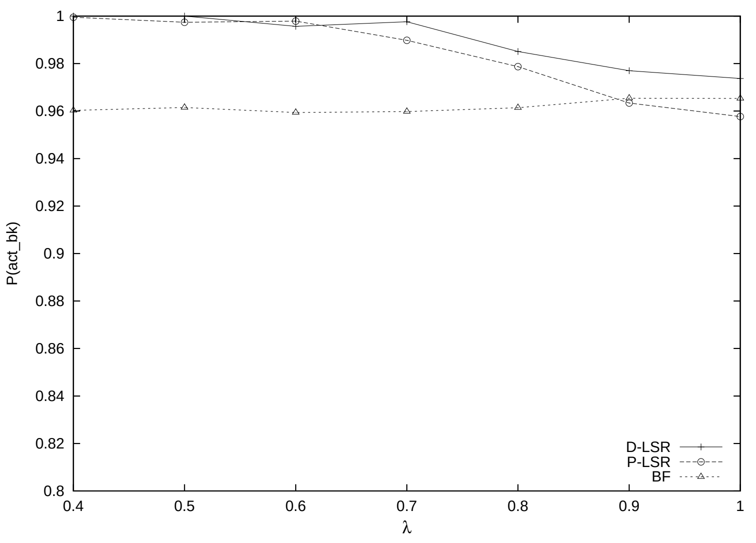
In the second part of the simulation, we set $\rho = \alpha = 1$, $\mu = 2$, and $\beta = 0$ in the bounded flooding scheme, and compare three routing schemes for different request arrival rates $\lambda \in \{0.2, 0.3, \cdots, 1.0\}$, under various network configurations. For convenience, in the following discussions, the symbol BF is used for the bounded flooding scheme. The fault-tolerance of the three routing schemes with uniform and non-uniform network traffic are plotted in Figure 2.6 and 2.7 respectively. The capacity overhead with uniform and non-uniform network traffic are shown in Figure 2.8 and 2.9, while varying arrival rates.

D-LSR offers the best fault-tolerance among all the cases considered and BF the least in most cases. This was expected since D-LSR employs the largest amount of information about network status, and BF uses the most limited information.

The fault-tolerance of both D-LSR and P-LSR degrades as the network load increases for the following reason. Some of the backups chosen by D-LSR or P-LSR traverse longer paths to go around those links that have backup conflicts. Longer backups may generate conflicts with other backups established later. This negative effect of longer backups is apparent when the network load is high, since the more requests for other backups will



(a) Average edge degree of 3

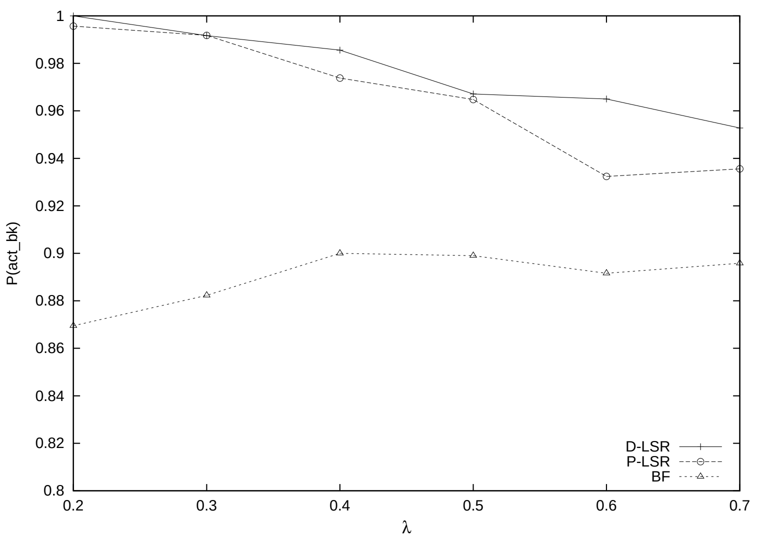


(b) Average edge degree of 4

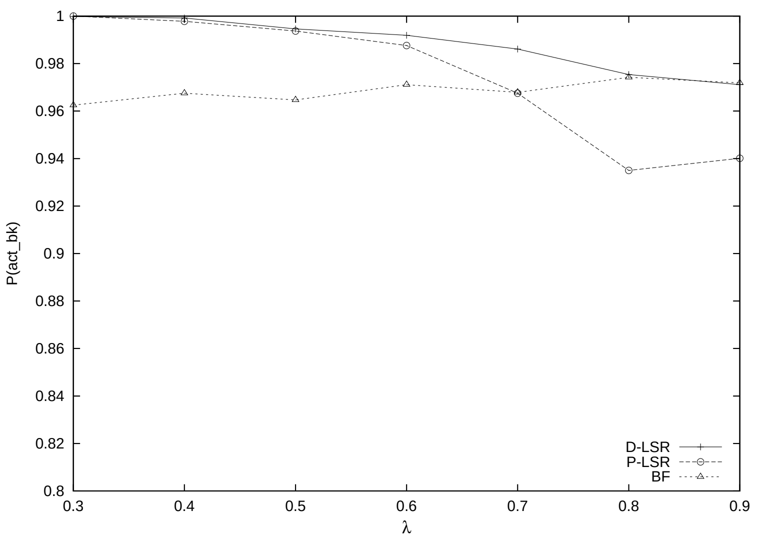
Figure 2.6: Fault-tolerance of the three routing schemes with uniform traffic

arrive before the longer backup is rejected when the arrival rate is high. BF does not show this phenomenon because backup route lengths are restricted by the bounding scope.

All three routing schemes provided higher fault-tolerance, as shown in Figure 2.6 (b) and 2.7 (b), when the network connectivity, E, is high regardless traffic patterns. When the network has more links, there are more paths between any two nodes. Thus, in a highly-



(a) Average edge degree of 3

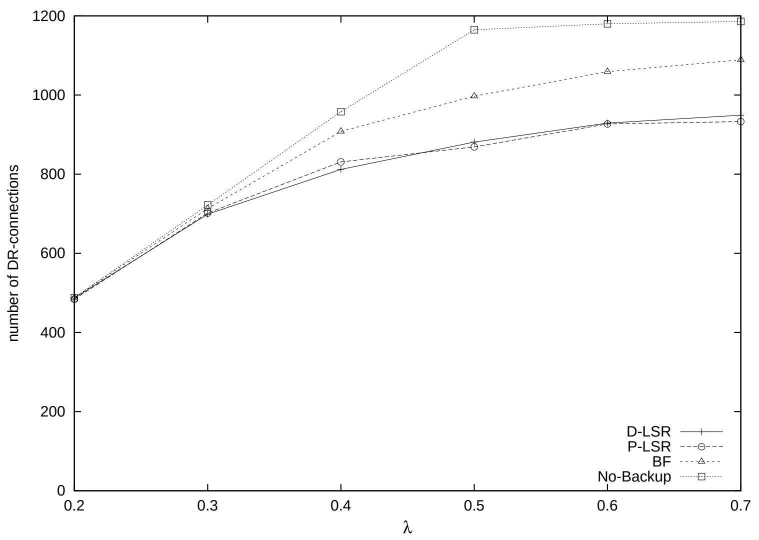


(b) Average edge degree of 4

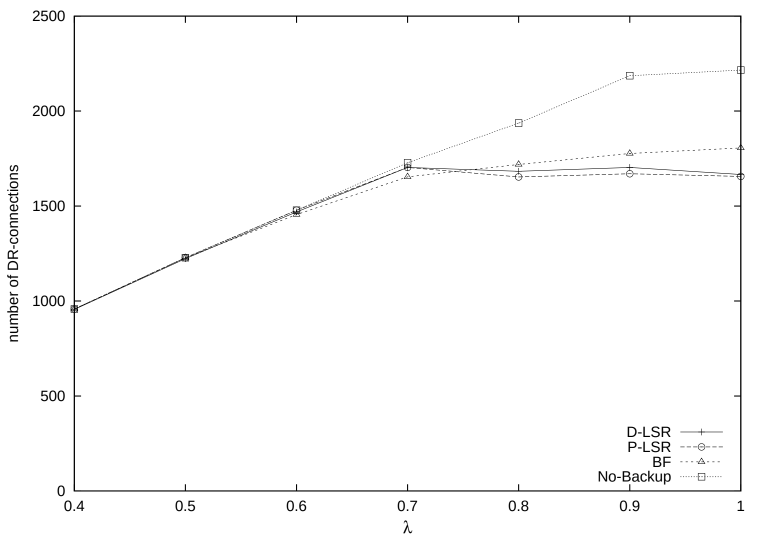
Figure 2.7: Fault-tolerance of the three routing schemes with non-uniform traffic

connected network, a node has more candidates for a backup and is more likely to find a backup that has less conflicts. Moreover, when the network connectivity is high, path selection is less critical to fault-tolerance. Since there are many candidate routes, even random selection can find a backup route with small conflicts.

As shown in Figure 2.7, when some DR-connections are concentrated on a small num-



(a) Average edge degree of 3

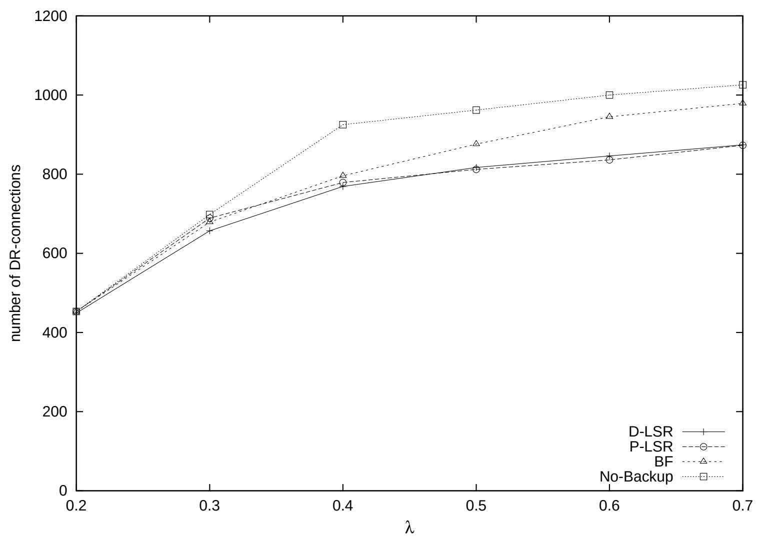


(b) Average edge degree of 4

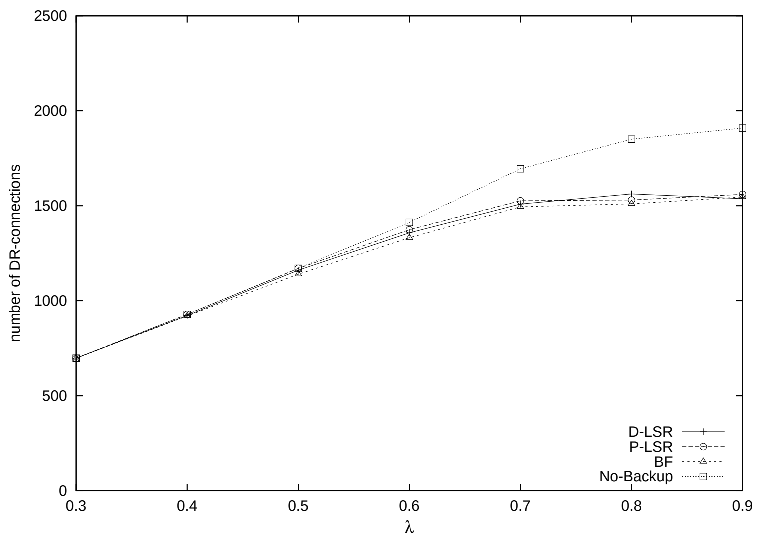
Figure 2.8: Capacity overhead of the three routing schemes with uniform traffic

ber of nodes, the performance gap between D-LSR and P-LSR is more pronounced. In such a case, some links may have many backups while others have very few. If a node should select one of two congested links, P-LSR cannot distinguish one from the other, since $\|APLV\|_1$ does not provide sufficiently-detailed information.

A network is said to be *saturated* if all of its resources are allocated to DR-connections.



(a) Average edge degree of 3



(b) Average edge degree of 4

Figure 2.9: Capacity overhead of the three routing schemes with non-uniform traffic

A saturated network cannot accept any more connections until some of the active connections terminate and release their resources. From the number of DR-connections that can be accommodated in Figure 2.8, one can see that the simulated network gets saturated as λ reaches 0.5 (0.9) for the case of $E = 3$ ($E = 4$). In the graphs, the *no-backup* scheme establishes only the primary channel of each DR-connection. To measure resource overheads of the three proposed routing schemes, we define the difference between the number of D-connections without backups and that of each routing scheme as *capacity overhead*. Clearly, since there are no backup channels, $P_{ack\_bk} = 0$ for the no-backup scheme, i.e., no fault-tolerance. If a backup channel is added for each DR-connection, the number of DR-connections in the saturated network drops.

Thus, the amount of resources reserved for backups could be indicated by the decreased number of connections that can be accommodated under each of the three routing schemes. As shown in the figure, all of the three proposed routing schemes decrease the network utilization by at most 20% when $E = 3$ and 25% when $E = 4$, but the fault-tolerance improves sharply: $P_{ack\_bk} > 85\%$. Recall that, without backup multiplexing, the network utilization would be decreased by 50% or more. What is more important is that DR-connections are shown to have high fault-tolerance and low capacity overhead until the network load reaches 70% of the maximum load.

Figure 2.9 shows the capacity overhead with non-uniform traffic. The simulated network becomes saturated earlier at λ of 0.4 (0.7) for the case of $E = 3$ ($E = 4$) with non-uniform traffic than with uniform traffic. When the bandwidth of links around the crowded nodes are depleted, the network cannot accommodate new connections because the network traffic is concentrated on a small number of nodes.

We summarize the evaluation results as follows: *(1) multiplexed backup channels improve the fault-tolerance at the expense of slightly decreasing the network utilization, and (2) the lower the network connectivity, the more sophisticated routing algorithm is necessary.*

2.6 Conclusion

In this chapter, we proposed and evaluated three routing schemes to find routes of backup channels of a dependable real-time connection. Two different methods are introduced to extend the link-state database to include the information about active real-time connections. Two link-state routing schemes determine backup routes with high-level fault-tolerance at the expense of overhead to maintain additional information in the extended link-state database. By contrast, the bounded flooding scheme does not require distribution and maintenance of link-state information, nor on-line route computation (e.g., the Dijkstra's algorithm). Instead, it is an on-demand scheme, and upon request of establishing a real-time connection, the *qualified* routes are discovered by flooding the special channel-discovery packets within a bounded area.

Using extensive simulations with **Matlab** and **ns**, we evaluated the three routing schemes in terms of fault-tolerance and resource-capacity overhead. Our simulation results show that good fault-tolerance can be achieved at a reasonable decrease in the number of real-time connections that can be accommodated. In addition, when the network load is not very high, allocation of spare resources for backup channels does not reduce the number of real-time connections that the network can accommodate.

CHAPTER 3

Improving Dependability of Real-Time Communication with Preplanned Backup Routes

To continue real-time communication service even upon failure of a network component, the network needs to reroute real-time channels that run through the broken link. If each node selects a detour route based on its information that does not reflect other nodes' path selection, this detour route may overlap with others and fail in establishing a real-time channel over the detour path due to limited resources.

This reactive rerouting approach is prone to signaling failures and may therefore degrade dependability. Banerjea [5, 4] showed that the sequential signaling is efficient to cope with this problem. Sequential signaling means that connection setup requests are served one by one. However, it is practically impossible to coordinate the order of setup requests in a distributed manner, and the sequential signaling takes a long time to reroute all the broken real-time channels. The proactive approach prepares a backup channel with resources reservation in advance and activates it upon failure of the corresponding primary channel due to a broken link/node. The proactive approach, although it offers higher dependability, incurs more overhead than its reactive counterpart.

In this chapter, we propose a novel scheme that preselects backup routes for real-time connections without reserving resources for each connection. We present a new distributed algorithm for backup routing. This new algorithm maintains the information of existing backups in an aggregate instead of per-connection form. Also, it does not require broadcasting of routing information.

3.1 A Hybrid Approach

Our goal is to provide high dependability of the proactive scheme with low overhead of the reactive scheme. To do this, we propose a hybrid scheme that decouples backup path selection from signaling for backup channels. This scheme preplans the path that a backup channel will take when it needs to be activated. We set aside a certain amount of bandwidth on each link that can be used only by backup channels to provide high fault-tolerance comparable to the proactive scheme, even when the network is highly loaded. The hybrid scheme defers the signaling for backup channel establishment and resource reservation until those are absolutely necessary.

3.1.1 Spare Resources

When almost all of the network resources are occupied by real-time channels, the network is said to be saturated. If the network is saturated when a link is broken, it is impossible to find a detour path. Thus, the dependability decreases dramatically as the network load increases. To cope with this problem, we need to set aside some resources. For the proactive approach, the backup channel establishment reserves resources, so it can prevent drastic decrease of dependability. Because our proposed scheme does not reserve bandwidth for each connection, we need a separate mechanism to set aside some resources.

In general, the bandwidth of a link is divided into two parts when the network deploys a real-time service based on resource reservation. One part of the bandwidth serves real-time traffic, and a proper bandwidth needs to be reserved before data delivery. the other part of the bandwidth must be left unreserved to prevent best-effort traffic from starvation.

In our scheme, the bandwidth is divided into three parts. The third part, spare bandwidth, is reserved for backup channel traffic. In other words, we set aside a certain amount of bandwidth for backup traffic. The spare resources of a link are used for best-effort traffic in the absence of failure.

This pre-reservation has several important differences from the per-channel base resource reservation of the proactive scheme. Our proposed scheme does not incur any complex signaling procedure for backups. Thus, our approach does not involve the intermediate routers on a backup path until the backup is activated while the proactive approach demands the intermediate routers to maintain information about backup channels and to change the amount of spare resources as backup channels are added or released.

Another advantage of the hybrid scheme is that the backup preparation does not affect the primary channel routing. As described in the previous chapter, in the case of the proactive approach, the backup channel establishment affects the primary channels to be established later. The backup resource reservation of the proactive approach requires other nodes to update the link states, because it changes the amount of available resources for real-time channels. This results in more frequent exchanges of routing messages, which are usually expensive. Our proposed hybrid scheme does not incur these additional routing messages because it does not change the amount of spare resources.

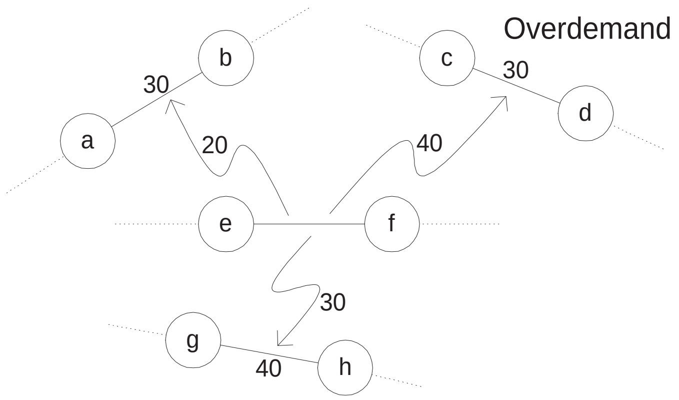
The hybrid scheme affects the best-effort traffic only when backups are activated whereas the proactive scheme changes the total available bandwidth for the best-effort traffic whenever the spare bandwidth increases or decreases. In fact, we can think of the backup channels as temporally borrowing some bandwidth from the best-effort portion of bandwidth.

3.1.2 Backup Route Selection

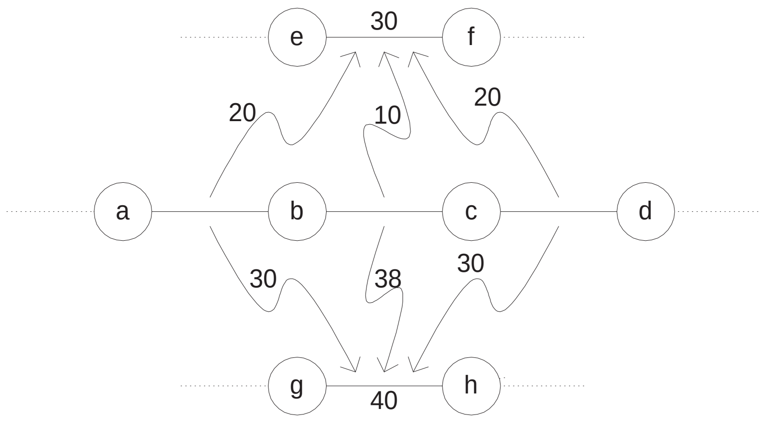
The dependability of a DR-connection is determined by its pre-selected backup route. When a link failure disables multiple primary channels, some backups of the disrupted DR-connections would fail to be activated if they traverse a link which does not have enough spare bandwidth to accommodate all the backups. To make the backup activation more likely to succeed, the backup routes should be chosen very carefully.

A link failure triggers the activation of multiple backups that traverse through different links. If the demand of backup bandwidth on each of these links is smaller than the amount of spare bandwidth, all the activation will succeed. Let b_ℓ^k denote the bandwidth demand on link ℓ when link k fails. In other words, $b_\ell^k = \sum_{d \in \mathcal{D}_\ell^k} r_d$, where $\mathcal{D}_\ell^k$ is the set of DR-connections whose primaries traverse link k with backups running through ℓ and r_d is the bandwidth requirement of DR-connection d.

Figure 3.1 (a) shows an example of spare and backup bandwidths. The number next to a link is the amount of spare bandwidth and the number next to an arrow from link k to link l denotes the amount of backup bandwidth required on link l if link k fails to activate backups. When link (e,f) breaks down, backups will be activated along pre-selected routes. Because links (a,b) and (g,h) have enough spare bandwidth, the backups routed through them will be activated successfully. In link (c,d), the backups require more bandwidth than



(a)



(b)

The number next to each link is the amount of spare bandwidth and the number next to each arrow is the amount of backup bandwidth required to activate backups. The unit is Mbps.

Figure 3.1: Overdemand of spare resources.

the spare bandwidth. So, some of the backups on the link may not to be activated. Because the backups can use the primary portion of link bandwidth, if available, the overdemand of backup bandwidth does not always cause activation failures.

The goal of backup routing is to choose a backup route such that the backup bandwidth of each link on the backup route does not exceed spare bandwidth after adding a new backup. Figure 3.1 (b) shows an example to choose the link of a backup route. We try to find a backup route for a primary channel that traverses links (a,b), (b,c), and (c,d). The new backup requires 5 Mbps. The already routed backups require bandwidth on each link as shown in the figure. Links (e,f) and (g,h) do not have overdemand from links (a,b), (b,c), and (c,d).

If we select either link (e,f) or (g,h) to construct a backup route, each link of the primary route will demand more backup bandwidth on the selected link. If link (g,h) is selected, link (b,c) will demand 43 Mbps backup bandwidth because the new backup requires 5 Mbps. This exceeds the spare bandwidth of 40 Mbps. If we choose link (e,f), however, we can avoid the overdemand because the spare bandwidth of link (e,f) is still larger than the increased backup bandwidth demand on link (e,f) from each link of the primary route. Thus, link (e,f) is a better choice than link (g,h) for the new backup.

As shown in the above example, when we choose a backup route for a DR-connection d, link ℓ is an appropriate choice of the backup route if $s_\ell \geq b_\ell^k + r_d, \forall k \in PR_d$, where s_ℓ is the amount of the spare bandwidth of link ℓ and PR_d denotes the set of links of d's primary route. To maximize the probability of successful backup activation, a backup route should minimize the number of links that overdemands from its corresponding primary route. Also, a backup should be as disjoint as possible from its primary route. To find the shortest route among those that satisfy the requirements, we use Dijkstra's algorithm after assigning a cost, C_ℓ, to link ℓ:

$$C_\ell = \begin{cases} M, & \text{if } \ell \in PR_d, \\ m, & \text{if } \exists k, \text{such that} k \in PR_d \text{ and } s_\ell < b_\ell^k + r_d, \\ \varepsilon, & \text{otherwise} \end{cases} \tag{3.1}$$

where $M \gg m \gg \varepsilon$.

When a source node establishes a DR-connection, the node selects a backup route after establishing a primary channel. So, the source node can easily check the first condition of the above equation. However, it is not trivial for the node to check the second condition because the node must know b_ℓ^k's. One possible way is that every node maintains all the b_ℓ^k's for every pair of links. However, this approach is not practical because it involves a large amount of information to be exchanged between nodes. To cope with this problem, we devised a new protocol that examines the second condition in a distributed manner.

3.2 Protocol for a Hybrid Scheme

3.2.1 Notation and Data Structures

In a network $G(\mathcal{N}, \mathcal{L})$ with $|\mathcal{N}|$ nodes and $|\mathcal{L}|$ links, each link has a unique id between 1 and $|\mathcal{L}|$. Let l_i be the link whose id is i.

- SV (spare bandwidth vector): An $|\mathcal{L}|$-dimensional integer vector whose i^{th} element denotes s_i, the amount of spare bandwidth of l_i. Every node maintains SV.

- BV$_i$ (backup bandwidth vector): An $|\mathcal{L}|$-dimensional vector whose j^{th} element is b_j^i. BV_i is maintained by node n if l_i is adjacent to n.

- APV (accumulated properness vector): An $|\mathcal{L}|$-dimensional one-bit vector. This vector is calculated before selecting a backup route to check which link is appropriate to compose a backup route. If the i^{th} bit is set to 1, l_i is suitable for the backup route.

3.2.2 Establishing a Primary Channel and Selecting a Backup Route

When a source node sets up a DR-connection, the node establishes a primary path first. It can use any routing methods for primary channels, because our backup routing is orthogonal to the primary routing. After setting up a primary path, the source node send a QUERY message along the primary route.

When an intermediate node receives a QUERY message, the node relays the QUERY message to the next node. When the QUERY message arrives at the destination node, the node prepares a RESULT message. The RESULT message has a field that contains the APV. At the beginning, every bit of the APV is set to 1. The destination node sends the confirmation message to the source node along the primary path in reverse direction.

When an intermediate router receives the RESULT message through l_i, it computes the links that overdemand resources on l_i and sets the corresponding bits of the APV to 0. In other words, if $b^i_j + r_d > s_j$, the j^{th} element of APV is set to 0. Because this intermediate node maintains SV and BV_i, the node can update APV easily. After updating the APV, the node relays the RESULT message to the next node.

When an RESULT message arrives at the source node, each bit of the APV in the RESV message is as follows.

$$APV^i = \begin{cases} 0, & \text{if } \exists k, \text{ such that } k \text{ is one of the links} \\ & \text{composing the primary route and } s_i < b^k_i + r_d, \\ 1, & \text{otherwise} \end{cases}$$

where APV^i denotes the i^{th} element of APV. The above equation is equivalent to the second condition of Eq. (1). $APV^i = 1$ if and only if l_i has enough bandwidth to accommodate backups when any link of the corresponding primary route fails. The source node selects a backup route using Dijkstra's algorithm after assigning link cost C_l:

$$C_\ell = \begin{cases} M & \text{if } \ell \in PR_d, \\ m & \text{if } APV^l = 0 \\ \varepsilon & \text{otherwise} \end{cases} \tag{3.2}$$

3.2.3 Maintaining BV

As described above, every router should maintain BV for each link attached to it to choose links that are proper as a backup route signaling to set up a primary channel. BV_i represents the resource demand on each link to activate backups when link l_i is broken. More precisely, the j^{th} element of BV_i is the sum of bandwidth requirements of backups on l_j. If ℓ is link $(v1, v2)$, both $v1$ and $v2$ maintain BV_ℓ.

To keep BV's up to date, each node should know backup routes of DR-connections whose primaries go through its links. After a source node chooses a backup route for a given primary channel, the source node informs nodes on the primary route of its decision. Also, when the backup route is no longer needed, the source node notifies the intermediate nodes to decrease BV. An intermediate node updates BV based on the data sent by the source node.

Because we cannot assume that the source node closes down the real-time connection gracefully, we take the soft-state approach to maintain BV. In the case of RSVP, the connection information is invalidated if it is not refreshed before a timer expires. However, maintaining BV as soft state is more complex because BV is summation of information about backups, whereas RSVP keeps the information on a per-connection basis.

The basic idea is that a source node periodically sends Backup Bandwidth Demand (BBD) message carrying link ids of a backup route and bandwidth requirement to intermediate nodes on the corresponding primary route. An intermediate node adds the backup bandwidth requirement into a temporal BV when it receives BBD. An intermediate node updates BV's periodically by replacing them with temporal BV's in which the node has accumulated bandwidth demands of backups. To make this operation idempotent, we incorporate a version number into BV and BBD messages. The detailed procedure is as follows.

1. A source node sends a BBD message along with the primary path. The BBD message constitutes the resource requirement of a backup route, the list of nodes that the message will visit, a version number for each router, and a backup route. All the version numbers are set to 0 when the end node sends BBD message for a backup route for the first time.

2. When an intermediate router receives a BBD message, it compares the corresponding version number in the message with that of its BV. If the version of the message is smaller than the that of BV, the bandwidth requirement is added into both BV and temporal BV. If the two version numbers are the same, the resource requirement is added only into temporal BV. If the message has a higher version than BV, the bandwidth requirement is not added into either BV or temporal BV. The intermediate node relays this BBD message to the next node after increasing the

corresponding version number in the message by one.

3. When the destination node receives the BBD message, it generates BBD-ACK message that has updated version numbers and sends BBD-ACK to the source node along the reverse primary route.

4. The intermediate nodes relay the BBD-ACK message until the message reaches the source node.

5. The end node records the version number in the real-time connection table and set up a timer with interval T. When the timer expires the node resends the BBD message.

6. All the nodes have a timer that periodically expires at an interval T. When the timer expires, the node replaces BVs with temporal BVs, resets all elements of temporal BV to 0, and increases the version number of BV by 1.

The above algorithm ensures two important features: 1) the BV and temporal BV are not increased more than once during one period for the same real-time connection, even when the end node sends the BBD message more than once before the node's timer expires; 2) the BV will be properly decreased within two periods even when the source node does not disconnect the connection gracefully.

3.3 Performance Evaluation

We evaluated the proposed scheme using the simulation. We implemented the reactive scheme, the proactive scheme, and the hybrid scheme using **ns**. We generated scenario files where DR-connection requests are listed. Each DR-connection request is composed of a source node, destination node, bandwidth requirement, start-time, and end-time. The scenario files are fed to the simulator. The simulator attempts to establish a DR-connection based on the source node, the destination node, and the bandwidth requirement. If it succeeds, the simulator records the path taken by the primary and the backup channel on a trace file and terminate the DR-connection on the given end-time. We analyzed the trace files generated during the simulation to evaluate performance metrics.

For simplicity, we assume that each DR-connection requires 1Mbps bandwidth. The lifetime of a DR-connection was chosen between 20 and 60 min with uniform distribution, so the average running time is 40 minutes. We conducted simulation with two network topology: an 8x8 mesh, an 8x8 torus and a random topology.

3.3.1 Load Index and Performance Metrics

To measure how many DR-connections each scheme accommodate at various network loads, it is very important to choose the load index that can represent the load imposed on the network and the performance metrics to compare different schemes. For best-effort traffic, the overall resource usage or the amount of traffic can be a good load index. The resource usage and the traffic amount, however, are one of the performance metrics in the real-time QoS networking. The path of a real-time channel is not always the shortest path. Depending on the routing schemes, a real-time channel traverses a different path and consumes a different amount of bandwidth.

The amount of resources consumed by real-time channels can be expressed:

$$\lambda' \times \overline{RT} \times \overline{h'} \times \overline{b'}$$

where λ' is the rate of setting up real-time channels, $\overline{RT}$ is the average lifetime, $\overline{h'}$ is the average path length of established real-time channels, and $\overline{b'}$ is the bandwidth requirement of established real-time channels. Since we assume that every real-time connection requires the same amount of resources, b, for our simulation, $\overline{b'} = b$. Since the network resources are limited, the setup rate has an upper limit:

$$\lambda' \leq \frac{B}{\overline{RT} \times \overline{h'} \times b} \leq \frac{B}{\overline{RT} \times \overline{D} \times b}$$

where $\overline{D}$ is the average distance between two nodes and $\overline{h'} \geq \overline{D}$. We define the maximum setup rate of real-time connections that the network accommodates:

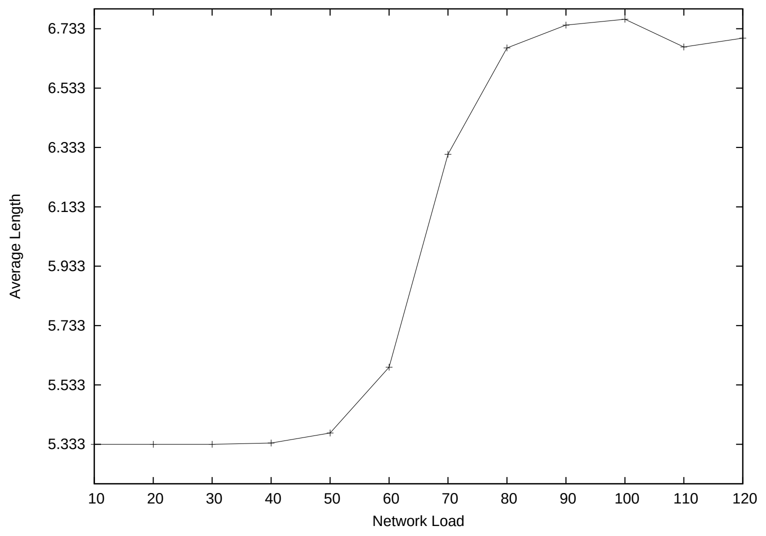
$$\lambda'_{max} = \frac{B}{\overline{RT} \times \overline{D} \times b}$$

Because the network cannot accept setup requests at a higher rate than $lambda'_{max}$, we increase the request rate of real-time connections, $lambda$ up to λ'_{max} to see how each scheme performs. We defined the network load as the ratio of λ to λ'_{max} as follows.

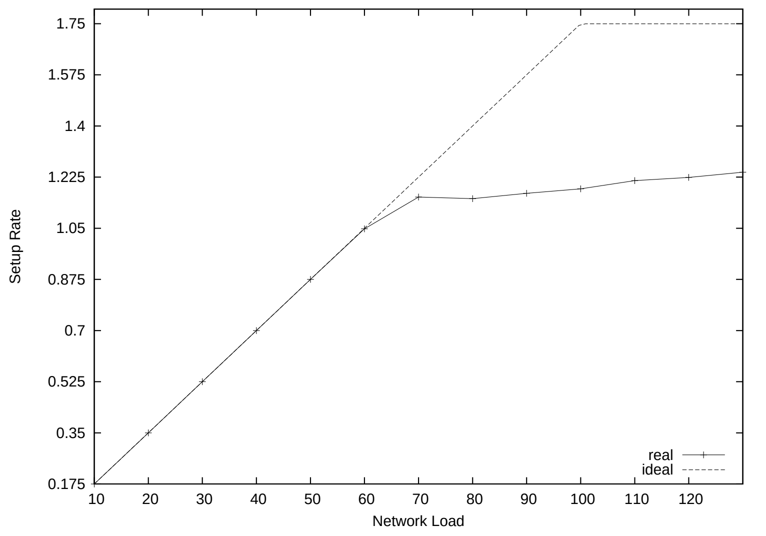
$$NetworkLoad(\%) = \frac{\lambda}{\lambda'_{max}} * 100$$

As the network load increases, all the bandwidths of some links are consumed by real-time channels. So, real-time channels traverse non-shortest routes i.e., $\overline{h'}$ becomes larger than $\overline{D}$. Thus, the network becomes saturated before the network load reaches λ'_{max}.

Figure 3.2 shows how $\overline{h'}$ and λ change in the simulation network. The average distance, $\overline{D}$, is 5.33 and the maximum setup rate, λ'_{max}, is 1.75. We established real-time connections without backup channels. The line *real* represents the simulation results; in Figure (b), the line *ideal* shows the theoretical setup rate where the $\overline{h'}$ stays equal to $\overline{D}$.



(a) Average Length Connections



(b) Acceptance Rate

Figure 3.2: Example of performance metrics and load index

After the network load reaches 80%, the network is saturated and cannot accept more DR-connection requests. When this happens, increasing the network load does not make a considerable difference in the network performance.

$\overline{h'}$ is one of performance metrics. We use *normalized average hop count* $\overline{nh'}$, defined as $\frac{\overline{h'}}{\overline{D}}$, to make the metric independent of the network characteristics. $\overline{nh'}$ of primaries shows the bandwidth each primary channel consumes in proportion to the average bandwidth usage of the shortest path. Usually, $\overline{nh'}$ of backups is longer than that of primaries. As $\overline{nh'}$ of backups becomes longer, the backups consume more bandwidth when they are activated, and the end-to-end delay of backups is increased.

The dependability is the probability that a DR-connection can continue the service even in the case of failures to the primary channel of the DR-connection. To measure the dependability, we failed every link alternately and tried to activate the corresponding backups. We calculated the probability of successful backup activation on each link and computed the average probability over all the links. More precisely, the dependability is defined as:

$$Dependability = \frac{\sum_{\ell \in \mathcal{L}} \frac{|\mathcal{S}_\ell|}{|\mathcal{D}_\ell|}}{|\mathcal{L}|}$$

where $\mathcal{L}_\ell$ is the set of backups successfully activated when link ℓ is broken, and $\mathcal{D}_\ell$ is the set of DR-connections whose primaries traverse ℓ.

Another important metric is the *average acceptance rate*, α, defined as $\alpha = \frac{\lambda'}{\lambda}$. Because the acceptance rate is closely related to bandwidth consumption, α shows *capacity overhead.*

3.3.2 Topology Characteristics

As mentioned earlier, we used 8x8 mesh, 8x8 torus, and a random network topologies. We generated the random topology using the Waxman 2 model with Georgia Tech Internetwork Topology Models (GT-ITM)[7]. Initially, we generated a network with 100 nodes and pruned nodes that have only one link to make every node have at least two links attached to it.

These three topologies have several different characteristics that affect the performance of fault-management schemes. Table 3.1 shows characteristics of three topologies.

	8x8 mesh	8x8 torus	random topology
Nodes	64	64	78
Links	112	128	129
Average Distance	$\frac{16}{3}$	$4 \times \frac{64}{63}$	3.706
Node Degree	3.5	4	3.308

Table 3.1: Characteristics of topologies used for simulation

Node Degree	2	3	4	5	6	7
8x8 mesh	4	12	48			
8x8 torus			64			
random topology	28	21	13	12	1	3

Table 3.2: Node degree distribution

Though the torus and the mesh have the same number of nodes and similar number of links, the average distance of the torus network is only about $\frac{3}{4}$ of that of the mesh network. By distance, we mean the length of the shortest path between two nodes. The random network has the shortest average distance though the node degree is the lowest. Figure 3.3 shows the distribution of distances. The longest distance of the mesh is almost twice of that of the torus.

Though three topologies have similar average node degrees, the distribution of node degrees is different. Table 3.2 shows the distribution. In the torus, every node has the same node degree of 8. In the random topology, 28 nodes have only two links, so there are a small number of choices for detour routes.

3.3.3 Comparison with the Reactive Schemes

Figure 3.4 shows the dependability of the two reactive schemes and the hybrid scheme. The reactive-immediate scheme tries to reroute all DR-connections at the same time. The

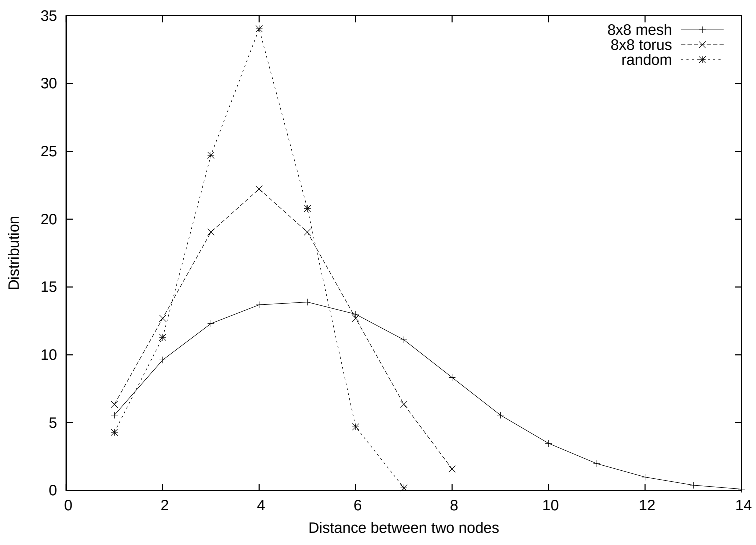


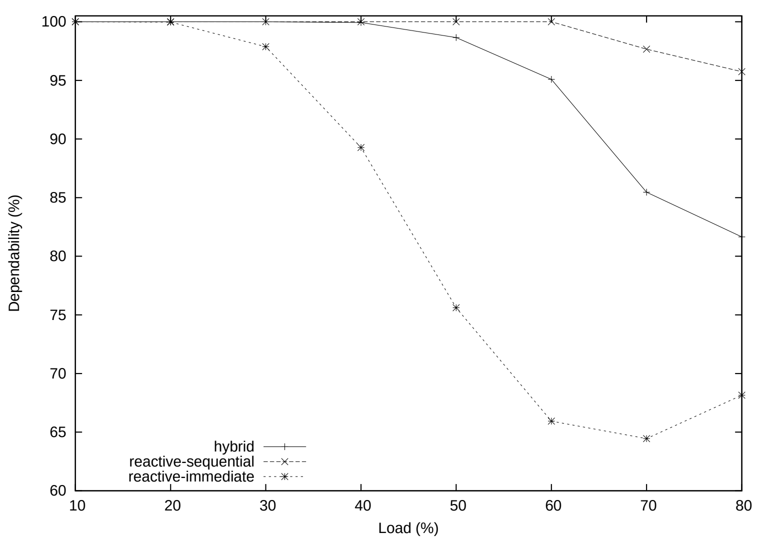
Figure 3.3: Distribution of distances

reactive-sequential scheme reroutes DR-connections one by one.

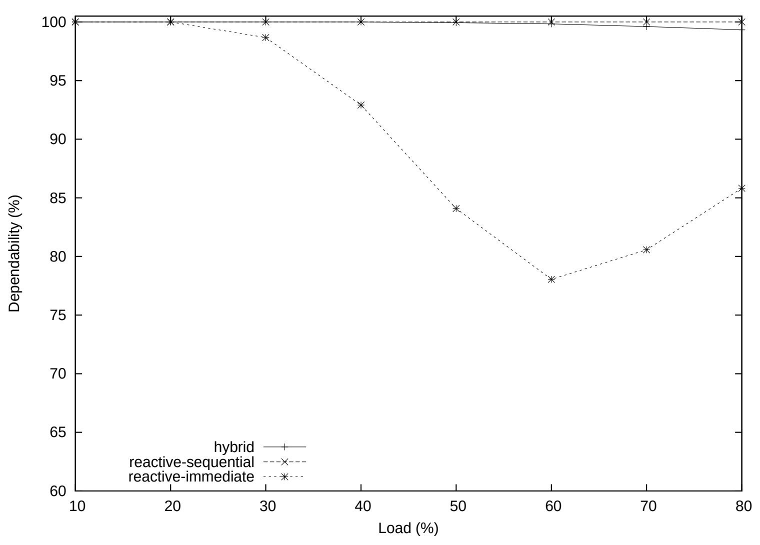
The reactive schemes do not utilize spare resources and suffer from resource shortage when the network load is high. To make a fair comparison and to evaluate the effect of spare resources, we provisioned spare bandwidth for reactive schemes. In this simulation, the bandwidth allocated for primary channels on each link is 100 Mbps and an additional spare bandwidth is provided for backups. We changed the spare bandwidth to see how each scheme performs with different amounts of spare bandwidth.

As shown in Figure 3.4, the immediate scheme provides considerably low fault-tolerance. It suffers from backup conflicts because each node tries to set up a backup channel independently without considering the backup route selection of other nodes. This is why the immediate scheme does not perform well with 16% spare bandwidth, with which the hybrid scheme and the sequential scheme provide 99% fault-recovery.

Both the hybrid scheme and the sequential scheme improve the dependability as the spare bandwidth increases. Because both schemes avoid backup conflicts by considering



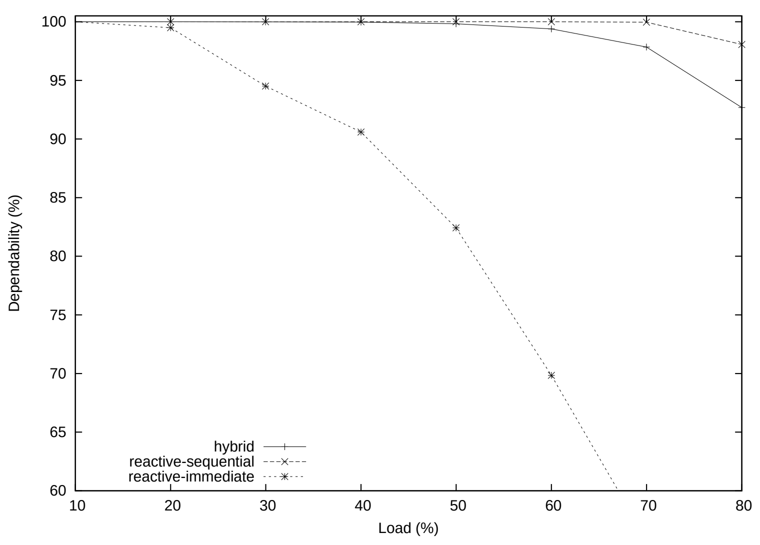
(a) Extra 6% spare bandwidth



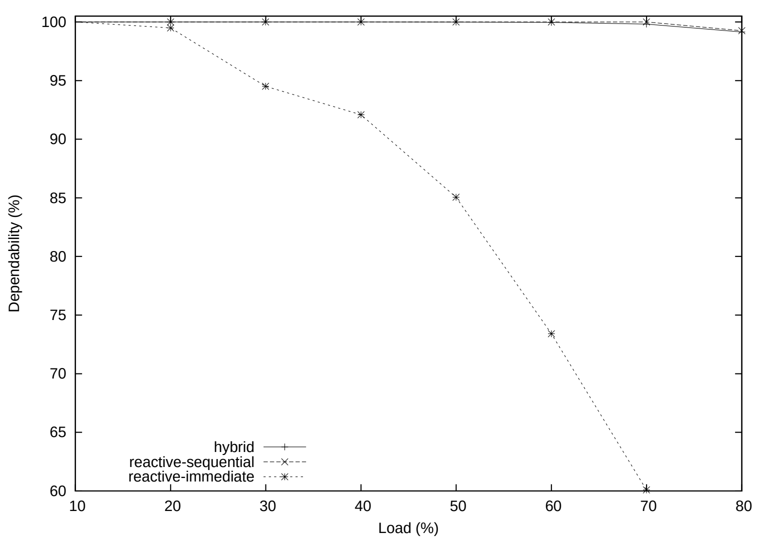
(b) Extra 16% spare bandwidth

Figure 3.4: Dependability of hybrid and reactive schemes on 8x8 mesh

of backup routes of other DR-connections, they can choose routes longer than the shortest path, which may be already taken, while the immediate scheme alway tries to route though the shortest path. As the spare resource increases, the hybrid and the sequential schemes are more likely to find longer detrour routes.



(a) Extra 6% spare bandwidth



(b) Extra 10% spare bandwidth

Figure 3.5: Dependability of hybrid and reactive schemes on 8x8 torus

3.3.4 Comparison with the Proactive Schemes

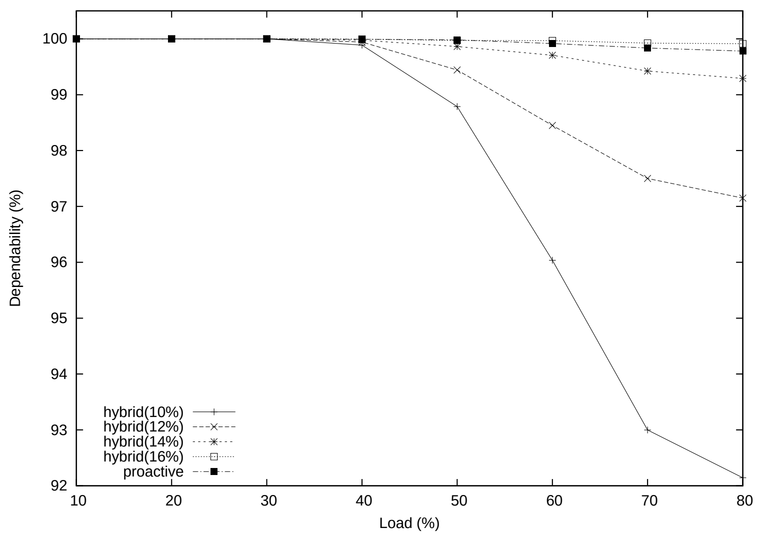
The sequential scheme shows slightly better fault-tolerance than the hybrid scheme. The sequential scheme utilizes all the network resources excluding the broken link whereas the hybrid scheme cannot use the whole primary path.

The sequential rerouting is practically impossible to implement and incurs a very long recovery delay. To reroute the DR-connections one by one in a distributed manner, it is necessary to enumerate their recovery order. Moreover, each node, which wants to establish a backup channel, waits for the new link status that reflects the recently-established backups. Because the fast recovery is one of the most important requirements for real-time communication, the sequential scheme is not applicable to real-time communication.

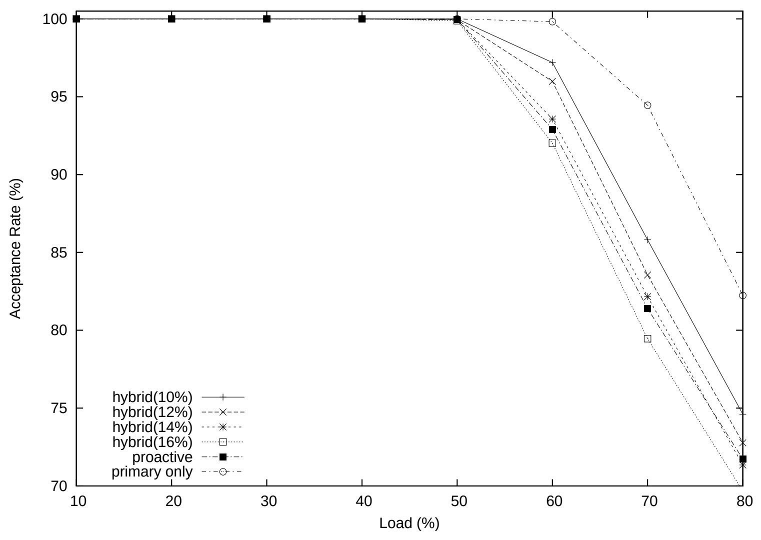
Figure 3.5 shows the performance of each scheme in 8x8 torus network. Both the hybrid and sequential schemes perform much better in this network than in the 8x8 mesh network. As stated in the previous section, the torus has a shorter average distance between two nodes. The average length of the backups is shorter in the torus than in the mesh and the backups use less bandwidth. Thus, with less spare bandwidth the torus can accommodate more backups. To provide 99% dependability, the hybrid scheme needs only 10% extra bandwidth.

However, the immediate scheme performs worse in the torus than in the mesh topology. The shorter average distance means that the DR-connection-source nodes are closer to the broken link. So, the end nodes that reroute DR-connections are more closely located to each other in the torus and the end nodes are more likely to choose same links for backup routes. The immediate scheme suffers from more contention in the torus.

As shown in this comparison with the reactive schemes, careful selection of backup routes improves dependability dramatically. To avoid contention, backups need to be distributed widely. With knowledge about backups of other DR-connections, the hybrid scheme distributes backups successfully and utilizes the spare bandwidth efficiently.



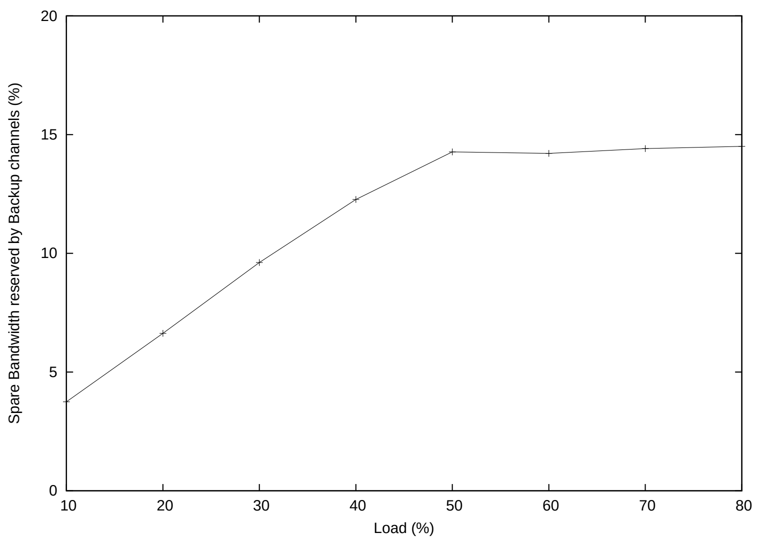
(a) Successful backup activation



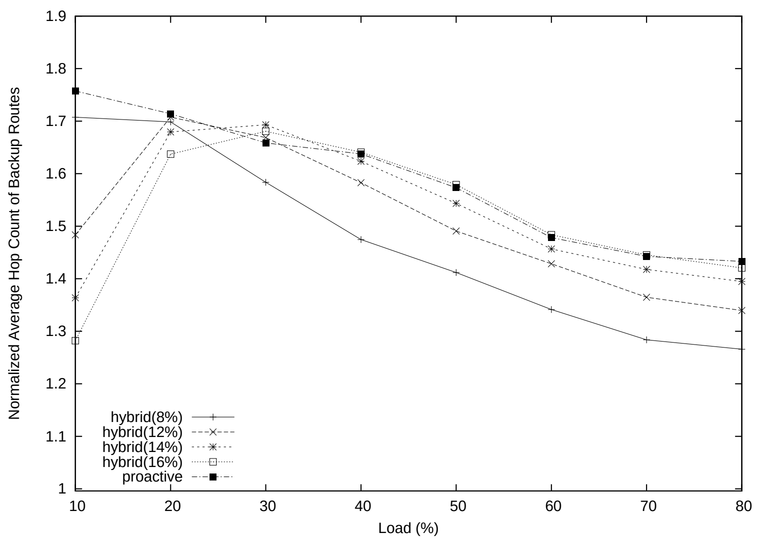
(b) Acceptance rate

Figure 3.6: Dependability of hybrid and proactive schemes on 8x8 mesh

We compared the hybrid scheme with the proactive scheme. Because the proactive scheme reserves spare bandwidth dynamically, it does not need separate spare bandwidth. To compare two schemes under the same condition, we assumed each link to have 100 Mbps bandwidth for DR-connections. Because the hybrid scheme requires separate spare bandwidth, we reserved a certain amount of bandwidth for backups out of the



(a) Spare bandwidth in the proactive scheme

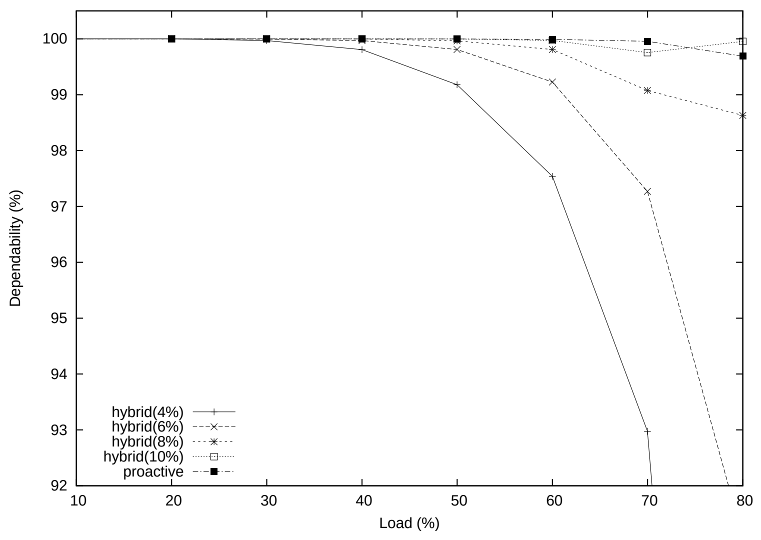


(b) Normalized average hop count of backups

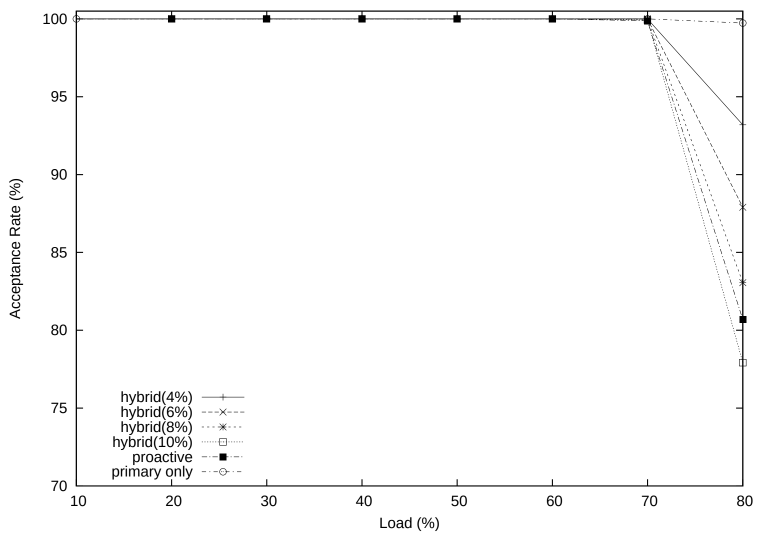
Figure 3.7: Backup route characteristics of hybrid and proactive schemes on 8x8 mesh

100 Mbps bandwidth. To see the performance of the hybrid scheme with different amounts of spare bandwidth, we varied the spare bandwidth from 4% to 20% of the total bandwidth.

Figure 3.6 shows the performance of the proactive scheme and hybrid schemes. The hybrid scheme improves dependability as the spare bandwidth increases. When 16% of total bandwidth is provisioned as spare bandwidth, the hybrid scheme shows dependability



(a) Successful backup activation

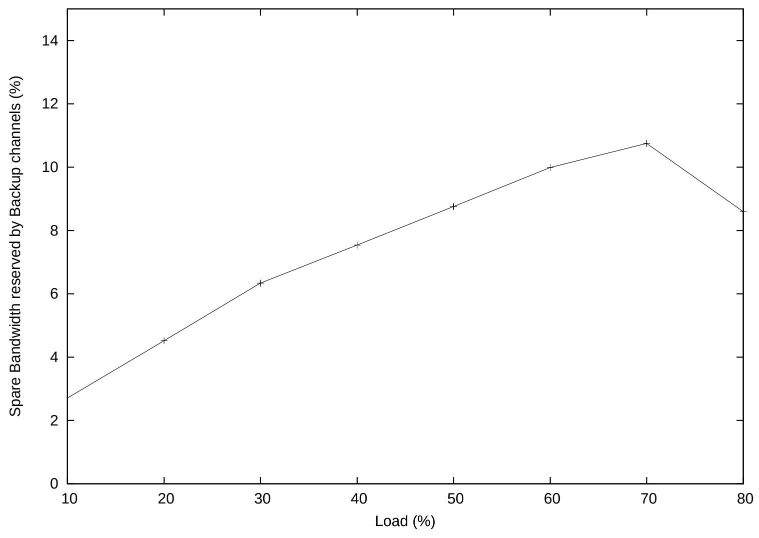


(b) Acceptance rate

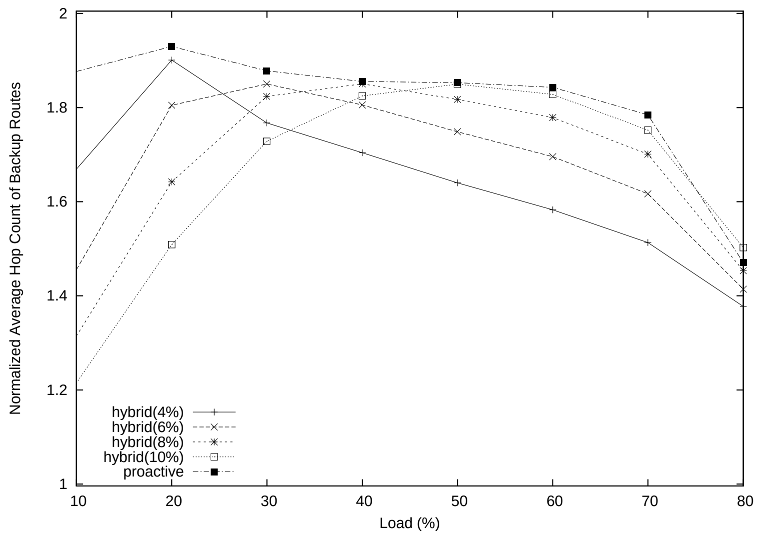
Figure 3.8: Dependability of hybrid and proactive schemes on 8x8 torus

comparable to the proactive scheme.

However, as more bandwidth is reserved for backups, less bandwidth is available for primaries. Figure 3.6 (b) shows the acceptance rate of requests for DR-connections. In the figure, the 'primary only' represents the acceptance rate when we establish real-time channels without backups. The difference between the primary only and each scheme is



(a) Spare bandwidth in the Proactive Scheme



(b) Normalized average hop Count of Backups

Figure 3.9: Backup route characteristics of hybrid and proactive schemes on 8x8 torus

the *capacity overhead*.

The proactive scheme shows capacity overhead similar to that of the hybrid scheme with 14% spare bandwidth. We can find the reason in the Figure 3.6 (c). The figure shows the amount of bandwidth reserved by backup channels in the proactive scheme. Because the proactive scheme reserves spare bandwidth according to the network load, more bandwidth is reserved for backups as the network load increases. After the network load reaches 50%, the spare bandwidth does not increase, because the primaries occupy the remaining bandwidth.

The proactive scheme reserves a maximum of about 14% bandwidth for backups at maximum. This is the reason why the proactive scheme shows the acceptance rate similar to the hybrid scheme with 14% spare bandwidth. However, the proactive scheme provides higher dependability than the hybrid scheme with this 14% spare bandwidth. To match the dependability of the proactive scheme, the hybrid scheme requires a little more bandwidth, because the proactive scheme utilizes much more information when it selects backup routes.

Figure 3.6 (d) shows the average hop count of backups. When the network load is low, the backups of the proactive scheme is longer than those of the hybrid scheme with a similar amount of spare bandwidth. Because the hybrid scheme sets aside a certain amount of bandwidth regardless of the network load, the hybrid scheme has more room for backups, and it can find backup routes within near area. The proactive scheme starts without any spare bandwidth and increases it. As the backup bandwidth increases, it can find a shorter route for a given DR-connection using the spare bandwidth reserved for other backups.

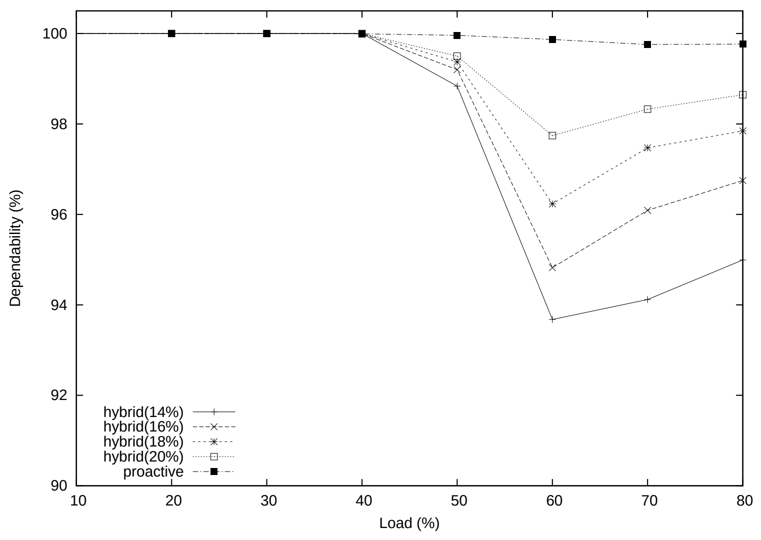
To see how the topology affects the performance of each scheme, we conducted the simulation with a torus and a random topology. Figure 3.8 shows the performance in the 8x8 torus topology.

As described in Section 3.3.2, the torus has a shorter average distance between two nodes as well as a higher edge degree. A shorter average distance means that each backup requires less backup bandwidth. As shown in Figure 3.8 (a), with 10% spare bandwidth, the hybrid scheme provides over 99% dependability.

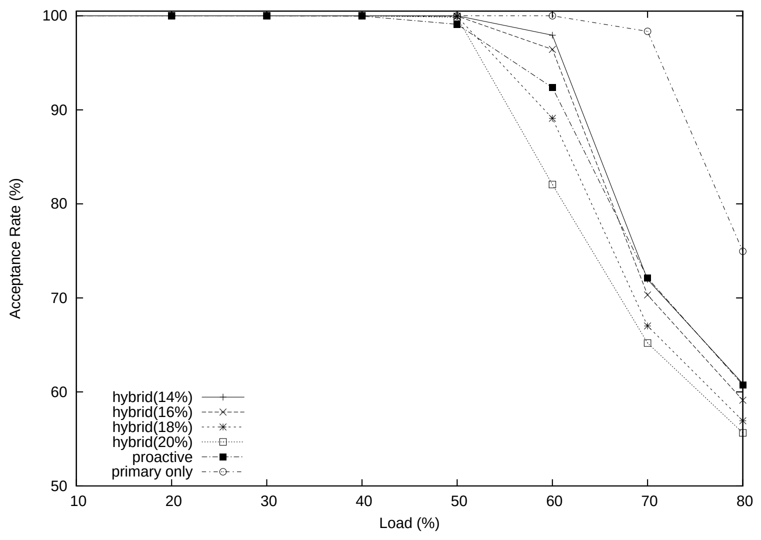
Figure 3.8 (c) shows the spare bandwidth reserved for backup channels in the proactive scheme. The maximum spare bandwidth is about 10%. In the torus, both proactive and hybrid schemes need less spare bandwidth than in the mesh network, because the torus has more edge degree in addition to a shorter average distance. A higher edge degree means that there are more disjoint routes within a certain boundary between two nodes. So, the backups are distributed more widely without conflicts and less spare bandwidth accommodates more backups.

When the network load is 80%, the spare bandwidth of the proactive scheme decreases. Because the primary channel reserves the bandwidth before routing the corresponding backup channel, whenever the bandwidth is available, the free bandwidth is allocated to a primary channel and the corresponding backup channel squeezes into the spare bandwidth shared by other backups. This results in less bandwidth allocated to backup channels and more backup conflicts. The reduction of the backup bandwidth accompanies the degradation of dependability. As the spare bandwidth of the proactive scheme decreases below 10%, the dependability of the proactive scheme also gets deteriorated.

Because less bandwidth is reserved for backups in the torus, more bandwidth is available for primaries. Figure 3.8 (b) shows the acceptance rate. Until the network load reaches 70%, the acceptance rate is 100%. Compared with the acceptance rate in the mesh topology, the torus improves the acceptance rate considerably.



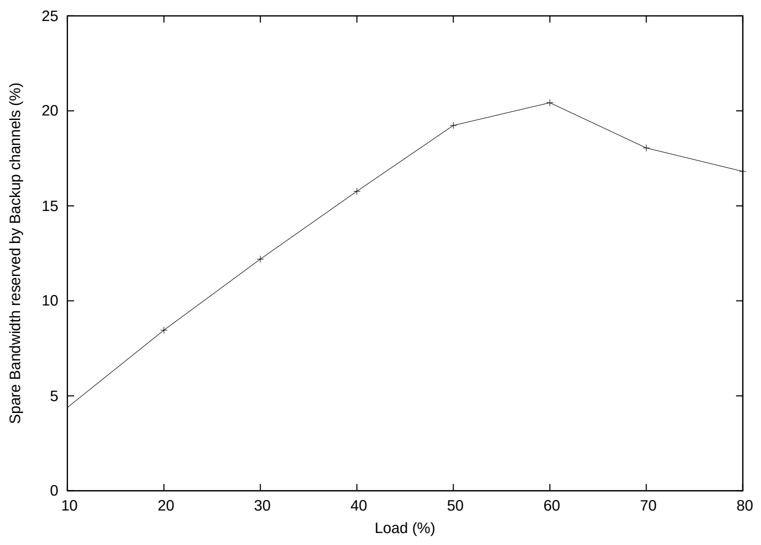
(a) Successful backup activation



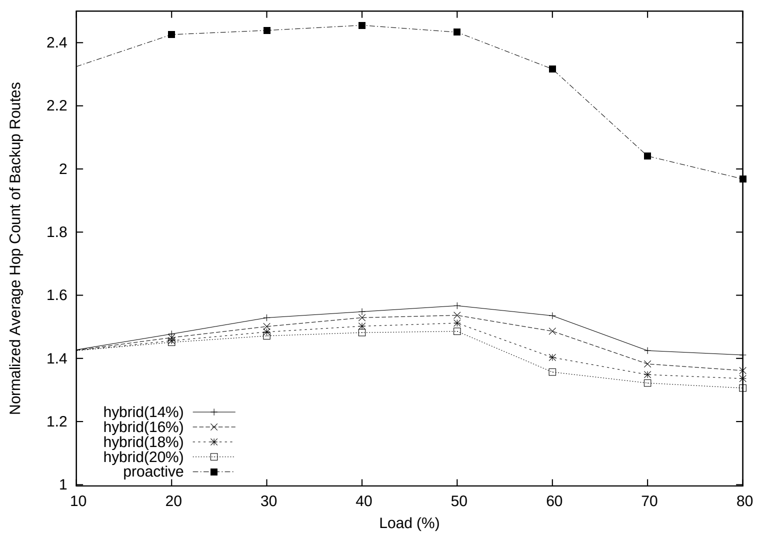
(b) Acceptance rate

Figure 3.10: Dependability of hybrid and proactive schemes on random network

The average hop count of backups shows a similar pattern. As the network load increases, the hybrid scheme selects longer backup routes to avoid overdemanding resources. After the network load reaches a certain point, it is impossible to build a route without any overdemanded link. Then, the length of backup routes decreases because a shorter path incurs less conflicts and the routing algorithm selects the shortest path when several paths



(a) Spare bandwidth in the proactive scheme



(b) Normalized average hop count of backups

Figure 3.11: Backup route characteristics of hybrid and proactive schemes on random network

have the same number of conflicts.

The 8x8 torus has 16 more links than the 8x8 mesh. The additional 16 links improve the performance of DR-connections dramatically. The torus uses 30% less spare bandwidth, provides higher dependability, and shows lower capacity overhead.

The random topology is similar to the torus in the average distance. However, in the node degree, the random topology is considerably different. The average node degree of the random topology is just a little lower than that of the mesh. However, there is a large deviation in the node degree of the random topology. More than a half of the nodes have 2 or 3 links whereas 4 nodes have 6 or 7 links.

The diverse node degrees introduce some disadvantages to the DR-connections. Because many nodes have a small number of links, there are fewer detour routes and more backups conflicts occur. Though the average node degree is smaller, the average distance of the random topology is shorter than the mesh and the torus. This is because a few nodes with many links act as crossroads. This results in nonuniform traffic flows in the network.

Figure 3.10 (a) shows the dependability in the random topology. The hybrid scheme provides a little lower dependability even with 20% spare bandwidth. As shown in Figure 3.10 (c), the proactive scheme reserves a maximum of 20% spare bandwidth. With a similar amount of spare bandwidth, the proactive scheme provides higher dependability because the proactive scheme reserves different amounts of spare bandwidth on each link according to network traffic, whereas the hybrid scheme uses the same amount of spare bandwidth on each link.

The proactive scheme shows less capacity overhead in Figure 3.10 (b). When traffic is concentrated in some links, the proactive scheme reserves spare bandwidth in other links. So, the congested links have more bandwidth for primaries, and backups run through other less congested links. This improves dependability and decreases the capacity overhead.

Figure 3.10 (d) shows the average hop count of backup routes. The proactive scheme selects significantly longer routes than the hybrid scheme. The poor connectivity of the random topology affects more the backup routing of the proactive scheme because the proactive scheme uses less spare bandwidth.

3.4 Related Work

Since Han and Shin [13, 14] proposed the backup multiplexing scheme, many proactive schemes have been proposed. In the backup multiplexing scheme, the same spare resources can be shared by multiple backups, if the corresponding primaries do not traverse same links. This approach needs to broadcast the information about spare bandwidth on each link to select backup routes. Also, it broadcasts the information of available bandwidth to select primary routes more frequently, because changing the amount of spare bandwidth affects the amount of the available bandwidth. Our hybrid scheme is the first approach that does not broadcast the information about the shared spare bandwidth.

Though Han and Shin explored several routing heuristics for backup channels, they did not propose a distributed algorithm for backup routing. Kodialam *et al.* [22] developed a routing algorithm that selects a backup path based on the amount of the aggregate bandwidth used on each link by primary channels, the aggregate bandwidth used on each link by backup channels, and the link residual bandwidth. The algorithm tries to minimize the amount of backup bandwidth increased by a new backup when selecting a route for a new backup. However, because the algorithm does not have any information about the paths of other backups, the algorithm overestimates the spare bandwidth by assuming that every disrupted DR-connection has conflicts on the same link.

Li *et al.* [28] recently proposed a distributed backup route selection algorithm for the proactive scheme. They use full information about spare resources as we do. Though their algorithm is similar to ours, there are several differences. First, their algorithm involves backup channel signaling. Moreover, though it is not clearly stated, their scheme needs to broadcast the amount of spare resources whereas our hybrid scheme does not need broadcasting.

Li *et al.* use the increment of spare resources as their metric to choose a backup route. In other words, they try to minimize the amount of spare resources. Whereas, we select a backup route to minimize the number of links that do not have enough spare resources. Because two algorithms use different metrics for path selection, the information exchanged for backup selection is also different. In our algorithm, routers exchange a bit-vector that represents a list of links that are suitable for a backup route, while in Li's algorithm routers exchange an integer-vector representing the amount of spare resources that will be needed when a link fails. Because the size of the vector is the same as the number of links in the network, the integer-vector consumes more bandwidth than the bit-vector and may not be delivered in a single packet.

3.5 Conclusion

In this chapter, we presented the hybrid scheme that preselects backup routes without reserving bandwidth for each backup channel. Because a certain amount of spare bandwidth is reserved for backups in advance, the hybrid scheme does not involve global routing messages for the spare bandwidth, which is one of main drawbacks of the proactive scheme. Also, we devised a novel distributed routing algorithm that does not require routers to maintain information for each DR-connection.

We evaluated the effectiveness of the hybrid scheme by simulation. We compared the hybrid scheme with the proactive and reactive schemes for various network topologies. The hybrid scheme provides as high dependability as the proactive scheme without broadcasting the information about spare bandwidth. When the network is homogeneous, the hybrid scheme is more effective. Thus, we successfully reduce the overhead of the proactive scheme without degrading performance.

CHAPTER 4

Improving the Availability of DNS service

When the DNS service becomes unavailable to a domain, the computers in that domain will be virtually disconnected. They cannot contact outside machines because they cannot retrieve the IP addresses for human readable domain names. To back up the DNS service, we developed mechanisms that enable the machines to continue use of the DNS service in a co-operative manner without resorting to any centralized local servers.

4.1 Introduction

DNS is an essential service in the Internet. Since domain names are much easier for humans to remember than IP addresses, most Internet services expose their domain names to users and applications. To contact remote services, clients consult DNS servers to resolve the domain names of remote services. Since DNS query-and-reply transactions precede most Internet services, it is important to improve the availability of DNS service. There have been several proposals to shorten DNS lookup delay[10, 20, 37]; all of them focus on improving the performance of DNS rather than the availability of DNS.

DNS service translates domain names to IP addresses. When a client requests the IP address of a domain name, the DNS query/reply result is cached at the local name server with a TTL field, which specifies how long the cache entry will be valid. When another client requests the IP address of the same domain name before the corresponding entry's TTL expires, the local name server can reply to the client using the cache entry.

Client machines rely on a local name server to use DNS service. If the local name server fails, all the clients within the domain cannot resolve names to IP addresses and thus cannot reach remote sites. To improve the availability of local name servers, each domain can be equipped with up to three name servers. Since the number of name servers is limited and the information about the name servers is open to the public, it is easy to attack name servers.

DNS servers can fail because of OS crashes, power outages, and hardware malfunction. Malicious attacks can also cause server failures. DNS replicates name servers to tolerate server failures. Although the replica-based approach is effective if the failure of a replica is independent of that of other replicas, this approach may be unable to overcome attacks by malicious users. A replica system usually has a mechanism to redirect incoming requests to other replicas when one fails, but this mechanism may work against the system in case of malicious attacks by redirecting attacking traffic to other live replicas. If the amount of attacking traffic exceeds a certain threshold, a replica system may result in a chain of failures.

There have been numerous attempts to improve the security of servers, but these attempts still leave servers vulnerable to certain attacks. Instead of trying to protect DNS servers from malicious attacks and to reduce server failures, we focus on providing a continuous DNS service in the event of server failures. More specifically, we aim to provide continuous DNS service to clients even when all the local name servers fail.

When local name servers fail, the client machines cannot find IP addresses of domain names. Thus, the local domain becomes virtually disconnected from the Internet. One simple way of increasing the availability of DNS service is that every client keeps its own name server. This approach, however, will lose all the benefits of using local name servers: shortening the client-perceived latency, reducing the DNS traffic to the outside world, and lowering the load of remote name servers. Using this approach, the cache hit ratio will be much lower because the cached information is less frequently refreshed and the name server cannot utilize the results of other clients' DNS activities. Thus, it is not desirable to deploy a DNS server in every client machine.

We propose a DNS agent which provides a backup plan for continuing DNS service even when all the local name servers fail. We activate a backup plan when local name servers fail. This plan enables clients to use DNS service at the time of server failures. The plan should minimize the preparation cost, and should not overload outside machines when it tries to tolerate failures. Overloading outside machines causes a chain of reactions. To prevent such reactions, we need to confine the effects of failures within a certain boundary.

To maintain DNS service in the presence of server failures, a DNS agent provides the following three functionalities:

- In the absence of failures, an agent works as a local DNS caching agent. It relays DNS queries/replies between a client and a local DNS server, and caches the results. If it has unexpired information for a DNS query, it replies without relaying the query to a local name server.

- When local name servers are not available, an agent shares its cache contents with other agents in neighbor machines (i.e., machines in the same domain).
- If an agent cannot find valid information in the neighbors' caches, it consults remote name servers to resolve the DNS query.

In designing the DNS agent, we have following three goals:

G1: **Minimize the preparation cost.** It is important to keep the preparation cost of a DNS agent low in order to deploy it widely.

G2: **Provide high performance.** The backup plan with the DNS agent should achieve performance that is comparable to the normal-operation mode. Since almost every Internal application involve DNS, it is important to provide high performance. When local name servers fail, it should allow clients to use the Internet without suffering any significant performance degradation.

G3: **Avoid increasing the load of outside machines.** When local DNS fails, we switch to a backup plan. If the backup plan increases the load of other machines, it may result in overall performance degradation or trigger a chain of failures in the worst case. When our backup plan is activated to tolerate failures caused by intentional attacks, other domains could also be under the attacks. Recently, we have observed several fast spreading Internet worms such as Code Red and Slammer worm [33, 44, 32], which can be used for a large-scale attack, involving several sites simultaneously. Thus, it is important to isolate the side effects of recovery.

A DNS-agent works as a proxy agent when local name servers are alive. The agent relays queries and replies between a stub resolver and local name servers. Because a DNS-agent has its own cache and stores replies from local name servers, it can answer the stub resolver without contacting local name servers if it has valid information.

The DNS cache size is usually not a limiting factor of cache hit ratio. Since each DNS information is small and the number of items is limited, cached items are rarely evicted due to an insufficient space. The DNS cache hit ratio is determined by the TTL and the frequency of requests. If the request frequency is much higher than the inverse of TTL, most of the requests result in cache hits. For popular sites, the request frequency is high because they are visited by many users. To improve the cache hit ratio, we need to provide a mechanism that aggregates the DNS replies.

DNS caching has some similarities to web content caching: Content caching is done mostly in a passive manner and contents have TTL values. Most web-proxy servers do web cache sharing. Many protocols for cache sharing have been proposed and analyzed [12, 8, 39]. For example, cooperative cache sharing is shown to be effective for TTL-based caches. Although web content caching differs from DNS caching in terms of cache entries' size and number, they both are TTL-based. We propose to apply cooperative cache sharing for DNS. More specifically, we propose a DNS agent that keeps a small cache and shares the cached information with other DNS agents in neighbor machines to use DNS information which is already fetched and stored in other machines.

To evaluate our proposed scheme, we collected DNS traces from the name servers of the EECS Department at the University of Michigan during two different periods. We then conducted simulation using these traces. With the proposed DNS agent, we can reduce the load of name servers, DNS lookup time and network traffic in the absence of DNS server failures. When local name servers fail, we can continue DNS service, reducing outgoing DNS traffic with high cache hit ratio.

4.2 Overview of DNS

Before the introduction of the Domain Name System (DNS), the name-to-address mapping was held in a single file, `HOSTS.TXT`, which was maintained by the Network Information Center (NIC) and `ftp`ed to each machine. Because this method was not scalable, the DNS was proposed in 1987. The DNS is a distributed database that maps a host name to the host information including its IP address, mail exchanger, and name servers [30, 31, 2]. Information about a host is called DNS *resource records* (RRs). The popular RR types are A-type (address), NS-type (name server), CNAME-type (canonical name) and MX-type (mail exchanger).

DNS has an inverted tree structure, called *domain name space*. This inverted structure is similar to the UNIX file system. Each subtree is a subdomain and the root node of the tree is represented as a dot (.), which is often omitted. To distribute the workload, subdomains can be *delegated* to lower-level name servers recursively and become separate zones. A name server, to which a zone is delegated, is authoritative for the zone. Currently, thirteen root name servers are placed over the entire world. Top-level domains, such as `.com`, `.net`, and `.edu`, are delegated to *generic top-level domain* (gTLD) servers, and top-level domains for each country are delegated to *country-code top-level domain* (ccTLD) servers.

To retrieve host information like the IP address of a host name, we need to travel from the root name server to the authoritative name server of the zone to which the host name belongs. Each client machine has a stub resolver that sends DNS queries to local name servers instead of actually traversing the domain space. A local DNS server iteratively consults remote name servers to retrieve the host information on behalf of the stub resolver.

To improve performance, DNS uses a cache. Name servers store all the information gathered during the resolution process. If a local name server has an RR for a query, it replies to a client with the cached information. Each RR has a time-to-live (TTL) value. After TTL expires, the cached RR becomes invalid and needs to be fetched again upon clients' requests.

Figure 4.1 shows an example of name resolution for `www.umich.edu`. The stub resolver sends a recursive query to a local name server. If the local name server does not have information for the requested host name, it consults the root DNS server. Because the `.edu` zone is delegated to a lower-level server, the root DNS server sends a reply that has a referral to the name server which is authoritative for the `.edu` zone. This iterative query-reply process continues until the query reaches the name server which is authoritative for `www.umich.edu`.

Since the local name server caches all the information from this transaction, the local name server can answer quickly by skipping some or all steps of the iterative procedure for later queries for the same domain. In addition to reducing the lookup delay, caching is important for the scalability of DNS. Using the caching mechanism, DNS reduces the load for name servers and uses less network bandwidth.

One common misunderstanding is that the number of queries/replies is limited by the

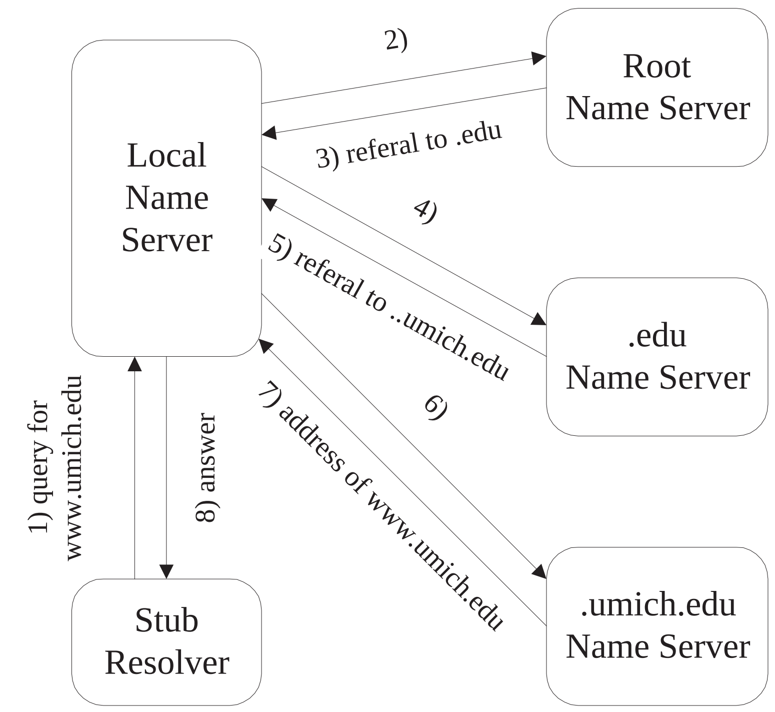


Figure 4.1: Example of domain name resolution

This example shows the steps to convert `www.umich.edu` to an IP address.

number of labels in a domain name. This is not necessarily true when the requested domain name is not the canonical name. The local name server may need to resolve the canonical name after getting a `CNAME` RR as the reply. For example, if the canonical name of `www.domainA.com` is `www.sub.domainB.net`, the authoritative name server replies a CNAME RR for a query for `www.domainA.com`. The local name server must traverse the domain name space again to resolve the canonical name (`www.sub.domainB.net`). This is common because content distribution networks (CDN) like *akamai* [1] redirect the traffic using CNAME. For example, using akamai the canonical name of `www.yahoo.com` is `www.yahoo.akadns.net`. Thus, a cache miss may result in several queries/replies, whose number may be larger than the number of labels in a domain name.

	Total Queries	Total Answers	A-type Queries	A-type Answers	Hit Ratio
July	7,344,683	5,297,053	3,356,857	2,703,199	73.7%
November	12,151,721	9,120,745	6,091,979	4,736,270	87.8%

Table 4.1: Trace statistics

4.3 Trace Collection and Analysis

The EECS Department at the University of Michigan has three name servers and more than 2,000 client machines. Because all of the name servers are located in a room, we were able to collect traces using a machine that had three network interfaces. We used `tcpdump` [17] to collect DNS traces. We collected traces for two months, July and November 2003. We used a week-long trace in each month. We only used traces of a week because a week is long enough to evaluate our scheme.

Table 4.1 shows basic characteristics of the traces. For DNS answers, we counted only the successful answers that do not contain any error codes. `A`-type queries received successful answers 73.7% in July and 87.8% in November. A DNS query can be about either a local or outside domain name. Since queries about local domain names always receive successful answers, we do not consider these queries in computing the cache hit ratio. The cache hit ratio is defined as the ratio of successfully answered queries about outside domain names to all queries about outside domain names.

The cache hit ratio is also higher for November than July because there was more DNS traffic and domain names with longer TTLs were more popular in November. DNS traffic was higher for November than July since our fall semester began in September. Figure 4.2 shows the cumulative distribution of TTL of domain names queried in each trace. We weighted each domain name with the access count of the domain name. People queried

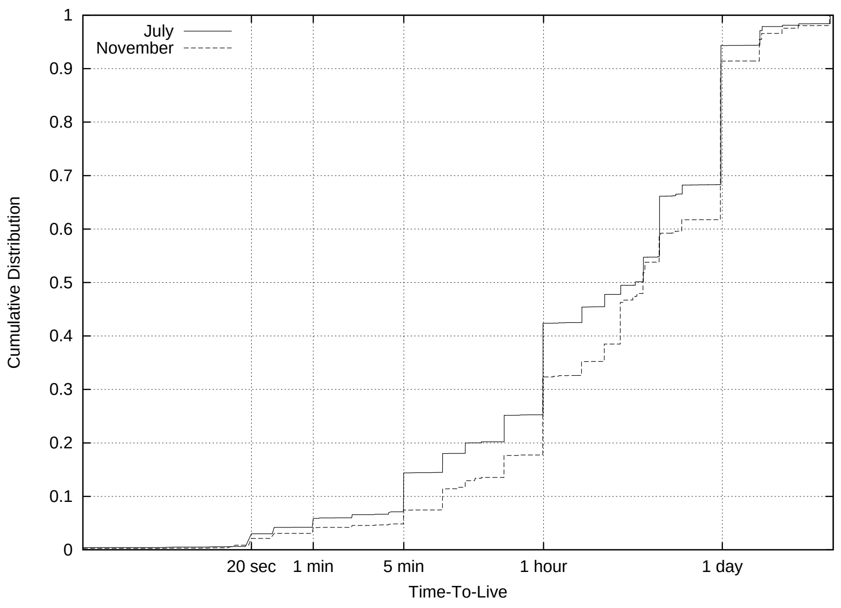


Figure 4.2: Cumulative distribution of TTL weighted by the access count

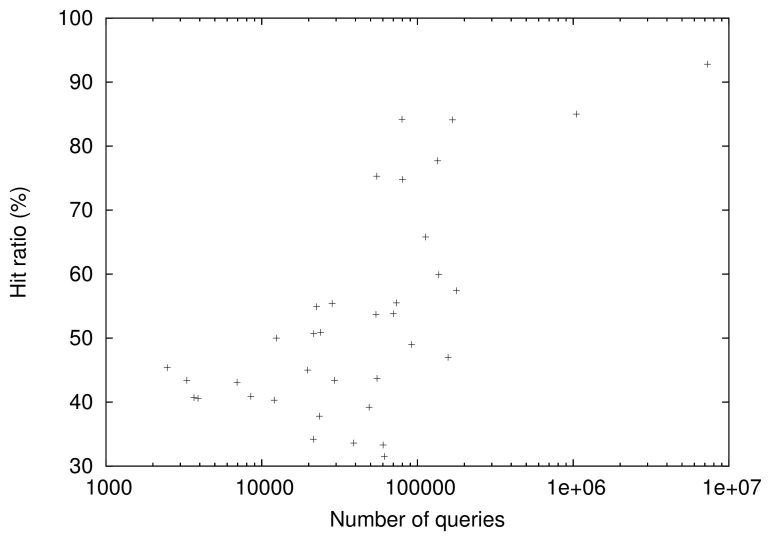
more domain names that have short TTLs in July than in November. Because information with short TTL becomes invalid quickly, the cache hit ratio was lower in July.

Table 4.2 shows characteristics of each subnet. The EECS Department network is composed of 22 subnets. Out of 22 subnets, we excluded four that have only a small number of computers; the computers in the four subnets rarely used DNS during our trace collection. The average cache hit ratio is very high, but the variance among subnets is large due to different usages.

Subnet 4 shows the highest hit ratio and dwarfs other subnets in the number of queries. Although its size is close to the average subnet, it accounts for 60% of July's queries and 74% of November's queries. This is because Subnet 4 includes `SMTP` servers and `PlanetLab` machines. `SMTP` servers use DNS information to validate mail senders, and `PlanetLab` is a platform for testing network services and creates unusually huge number of network activities.

Subnet		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
Jul.	# of Machines	42	21	23	38	58	42	70	33	43	52	51	13	13	15	11	63	20	68
	# Queries	8516	22545	60158	1048784	23913	19738	73166	79521	29344	54866	80003	3674	3306	6945	2472	38985	21467	55046
	Hit ratio (%)	40.9	54.9	33.3	85.0	50.9	45.0	55.5	84.2	43.4	75.3	74.8	40.7	43.4	43.1	45.4	33.6	34.2	43.7
Nov.	# of Machines	79	15	22	31	42	54	72	24	39	50	57	8	11	17	13	51	21	78
	# Queries	21579	28317	177932	7281688	54105	91743	136985	167839	69899	134722	112858	12436	12017	23422	3900	61193	48956	157327
	Hit ratio (%)	50.7	55.4	57.4	92.8	53.7	49.0	59.9	84.1	53.8	77.7	65.8	50.0	40.3	37.8	40.6	31.5	39.2	47.0

Table 4.2: Subnet statistics



This graph shows the relationship between the number of queries and the cache hit ratio. The data is taken from the subnet statistics of our traces.

Figure 4.3: Number of queries versus cache hit ratio

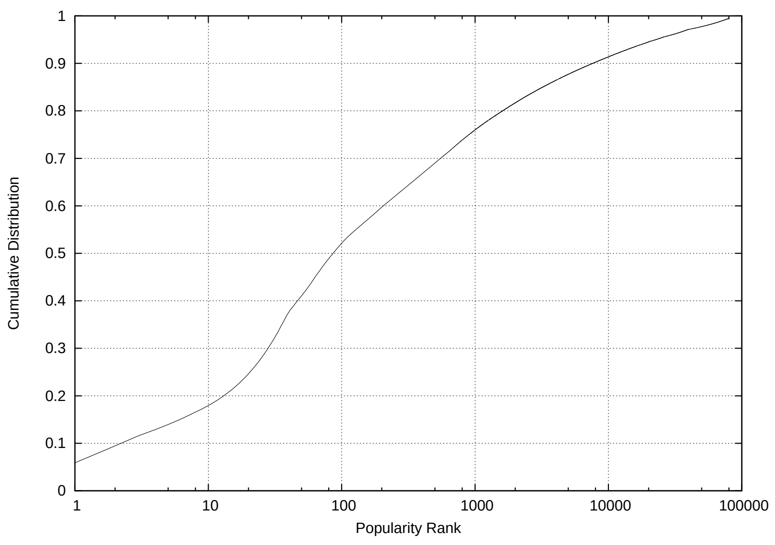
From Table 4.2, the hit ratio seems to be affected more by the number of queries than by the number of machines within a subnet. To see whether there actually is any relationship between the number of queries and the cache hit ratio, we plotted them in Figure 4.3. Although the trend is not strong, we still see a general relationship between the number of queries and the hit ratio: as the number of queries increases, the hit ratio increases.

4.3.1 Trace Characterization

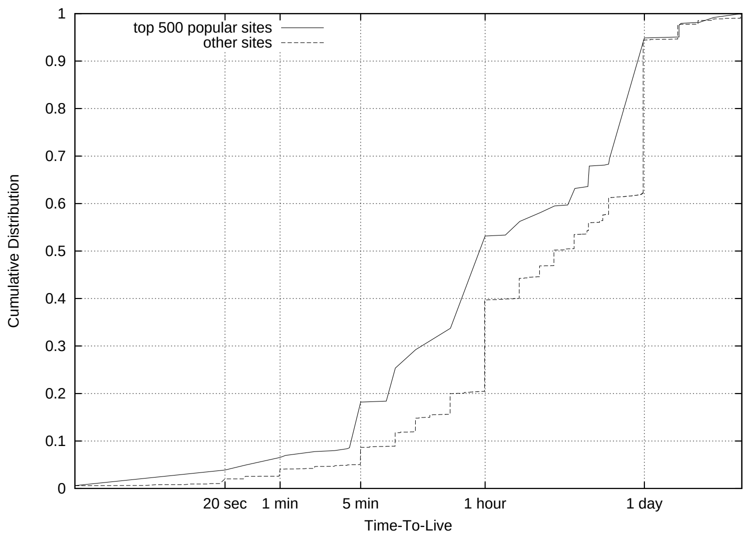
Figure 4.4 shows the characteristics of DNS requests collected during a week in July 2003. Figure 4.4 (a) shows that DNS queries exhibit a high degree of reference locality. By caching one hundred names, we can cover more than 50% of DNS queries. However, the required size of the cache increases rapidly to cover a higher percentage of queries. To cover 90% of queries, we need to cache about 10,000 name-address mappings.

Cached information has limited TTL. Queries for the expired information trigger cache misses; the information needs to be fetched again. Unfortunately, popular domain names tend to have a short TTL.

Figure 4.4 (b) shows the cumulative distribution of TTL. The solid line represents the distribution of the most popular 500 domain names and the dotted line represents the distribution of remaining domain names. Compared to the non-popular names, the popular names have twice more DNS replies with a TTL value less than five minutes. About 53% of the popular names have TTLs shorter than one hour. This is mainly because the popular web sites use DNS as a load-balancing mechanism. For a given domain name, the name server returns the RR of less loaded machine to steer new requests to them.



(a) Distribution of requests vs. popularity



(b) TTL distribution depending popularity

Figure 4.4: Characteristics of DNS requests

Since all the RRs are small and the number of items is limited, it is easy to provide name servers with an enough size cache which does not evict items due to insufficient space. The cache hit ratio is determined by the number of queries made before TTL expires. So, using the same DNS server with other clients increases the cache hit ratio [19, 18].

To understand the relationship between the number of machines within a domain and

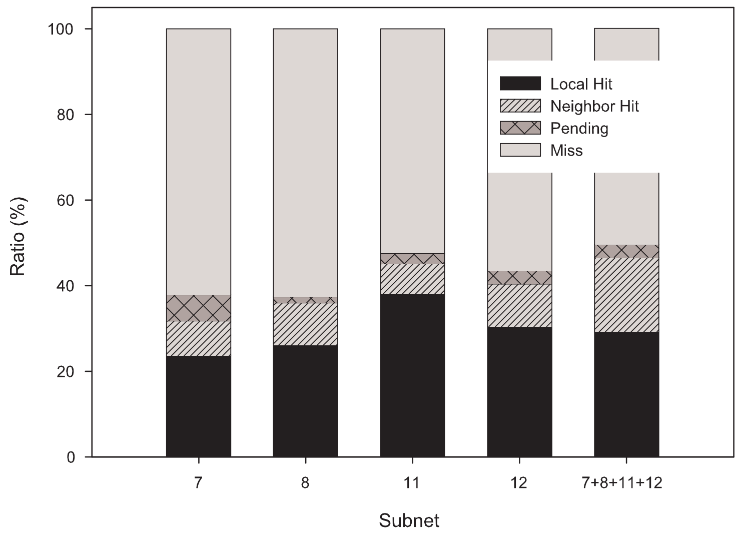


Figure 4.5: Effect of subnet size

the cache hit ratio, we created a new subnet that contains the total number of machines of four subnets: 7, 8, 11, and 12. Figure 4.5 shows how queries are processed in these four subnets and in the new subnet that we created. The new subnet has the highest overall cache hit. This is due mainly to the increase in the neighbor hit ratio, which is 17% for the new subnet.

4.4 Broadcast-Based DNS Agent

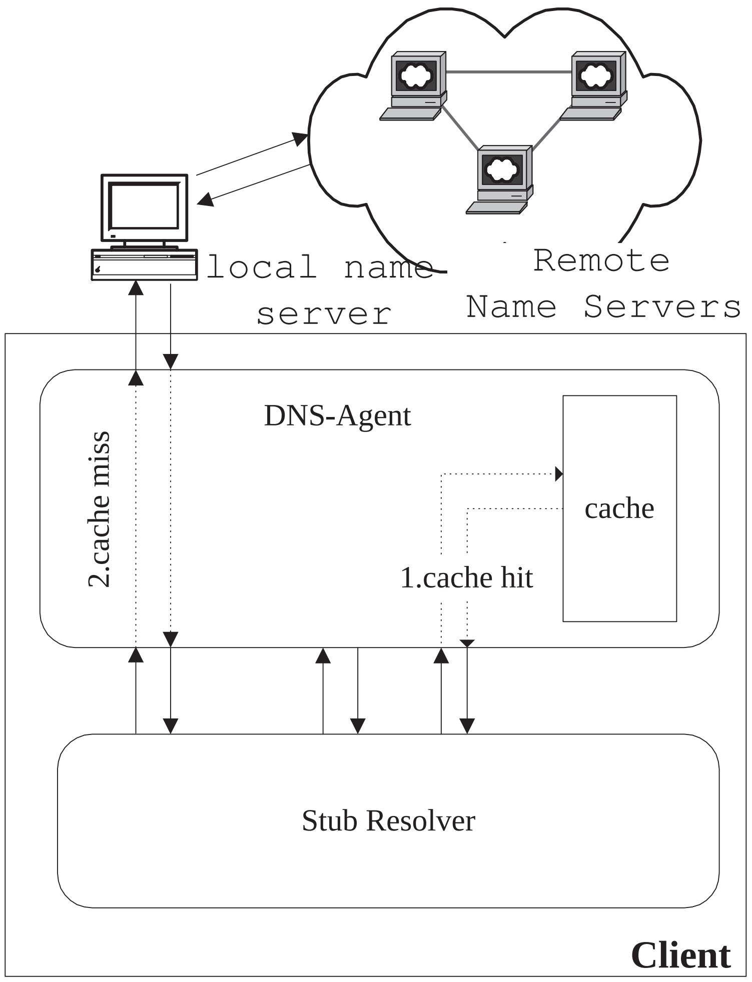
We first propose broadcast-based DNS agents that use broadcast messages to exchange cached DNS information. A client machine broadcasts its query to other clients in the network when it does not have valid information in its cache. Any client machine can reply to this query. To prevent a client from flooding a network and getting many duplicated replies, we introduce *answernet* that restricts the propagation of broadcast messages.

4.4.1 Operation of a DNS agent

A DNS agent has two operational modes: normal and backup modes. In a normal mode when local name servers operate normally, a DNS agent serve as a proxy agent. When a DNS agent receives DNS queries from a stub resolver on the same machine, it checks its cache to find a resource record (RR) for the query. If the agent's cache has a valid RR, the DNS agent replies to the stub resolver. Otherwise, the DNS agent relays the query to a local name server. When a local name server sends a reply, the DNS agent relays the reply to the stub resolver and stores the information in its cache.

When a local name server sends a reply, the message often contains authoritative name servers for the domain and their IP addresses in addition to `A` records. This information about authoritative name servers is useful only when the DNS agent works in a backup mode. Thus, we do not keep this information in the cache in the normal mode; we store only name-address mappings to minimize the preparation cost. For some domain names, the reply message has a canonical name and its IP address. For example, when a client sends a query packet for `www.google.com`, the reply message has a `CNAME` record, `www.google.akadns.net` and an `A` record for the canonical name. In general, the `CNAME` records have longer TTLs than `A` records. When a DNS agent receives this type of reply, it stores the name in the query with the A record.

Figure 4.6 shows the operation of a DNS agent in a normal mode. By accessing cached items, a DNS agent in the normal mode shortens latency, saves the network bandwidth, and reduces the load of name servers. The memory overhead is also small. Although we used a relative small cache with 1024 entries for our evaluation, no valid items were evicted due



This figure shows the operation of a DNS agent when there is no failure of local name servers.

Figure 4.6: DNS agent in a normal mode

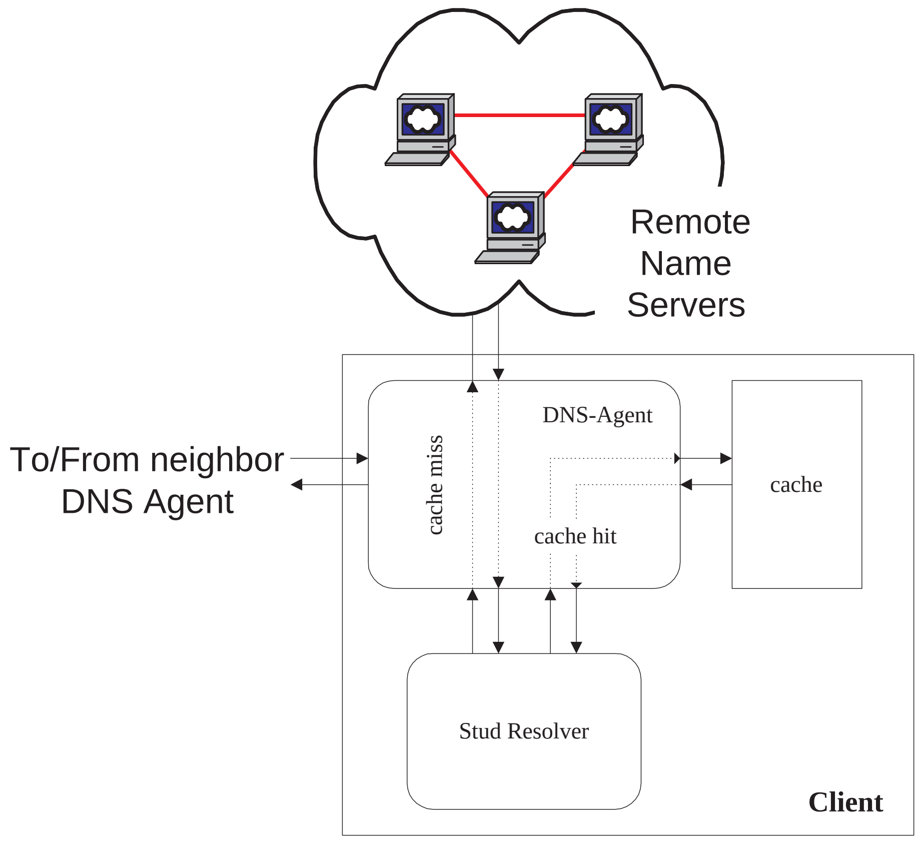
to a shortage of cache storage during our evaluation.

DNS agents work in a backup mode when local name servers are unavailable. When a DNS agent receives a query from the stub resolver, it tries to answer the queries by looking up its cache. But, not all queries can be answered from the cache because cache entries have limited lifetimes and some domain names were never queried before. One solution is to let an agent contact remote name servers directly when local servers are not available. This approach, however, produces many outgoing DNS queries, which consume the network bandwidth to the outside world and increase the burden of remote servers. To cope with this problem, DNS agents share cache contents with *neighbor agents* (i.e., agents in the same domain).

Figure 4.7 shows the operations of a DNS agent in a backup mode. An agent broadcasts a DNS query to neighbor agents when it does not have valid information in its cache. If a neighbor agent has valid information, it sends back a reply. If no other agent replies within two seconds, the requesting agent contacts remote name servers, stores the result in its cache, and replies to future queries from other machines. We chose the wait time to be two seconds, which is same as the time for a client to retry twice to query a local name server.

When a DNS agent works in a backup mode, it uses a slightly different cache-management policy. Since the agent contacts a remote name server in the backup mode, it stores all the additional information such as authoritative name servers and CNAME RRs. This additional information helps a DNS agent resolve a query using a small number of iterative queries. This policy increases memory usage, but it is nonetheless worth using the extra resource to reduce the lookup latency.

Although an agent has a better chance to get a reply as the scope of broadcast increases, we must limit the scope to prevent DNS queries from flooding the network. For example, we should not allow two thousand machines in our department to broadcast DNS queries over the departmental network. Because almost every large network is divided into subnets, it is easy to confine the broadcasting to a subnet. However, subnet-wise broadcasting may still cause two problems if the subnet is densely populated. One problem is the com-



This figure shows the operation of a DNS agent when local name servers fail.

Figure 4.7: DNS agent in a back-up mode

putational overhead on each machine. Since an agent checks its cache to decide whether it will send a reply or not, each query incurs computation overhead in neighbor machines. The other problem is that too many neighbors can send replies if the requested domain name is a popular one.

To alleviate these problems, we propose an optional mechanism that divides a subnet into smaller *answernets*. Two nodes belong to the same answernet if

$$(mask_i \ \& \ IP_l = mask_i \ \& \ IP_s) \text{ AND } (mask_e \ \& \ IP_l \neq mask_e \ \& \ IP_s)$$

where $mask_i$ is an including mask and $mask_e$ is an excluding mask. IP_l and IP_s are the

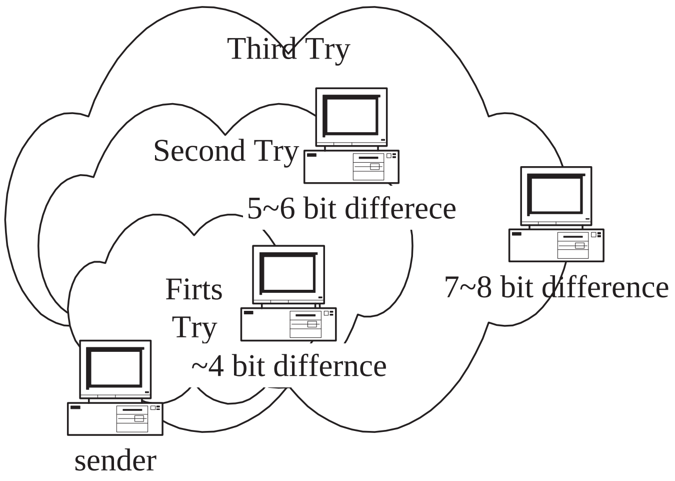


Figure 4.8: An example of expanding *answernet*

IP addresses of a local machine and the sender of the query, respectively.

A query message sent to neighbors includes *answernet mask* (i.e., including and excluding masks). We use an excluding mask to involve neighbor machines in the answernet at most once. When an agent receives a query, it first checks if the agent belongs to the same answernet. Only when the agent belongs to the same answernet as the querying machine, it looks up its cache. If no neighbors reply within a certain time, the agent resends the query with a looser netmask that allows a greater number of bit-differences in IP addresses. When a DNS agent resends the query, the previous $mask_i$ becomes a $mask_e$. This prevents from querying the same neighbors again. Without an excluding mask, an agent of the smallest answernet will receive the same query several times because the answernet is expanding.

In the example shown in Figure 4.8, a DNS agent can look up neighbors' caches up to three times before it sends the query to remote name servers. The bit settings of answernet masks should depend on the number of machines in the subnet.

A DNS agent needs to know at least one root server when it explores the domain name space. This information is called *root hints*. A DNS agent retrieves the hint by querying the `NS` record of the root domain (i.e., .) when the local name servers are alive. When a DNS agent starts up, it sends a query and saves the information in its cache. If a DNS agent starts when the name server is down, the agent cannot get the information. Generally, DNS agents store and share `A` type RRs, but do not do that for `NS`-type RRs. However, the agents do support `NS`-type queries only for the root domain.

Local name servers are often configured as authoritative servers for a domain. If local servers are unavailable, DNS agents cannot resolve the queries for local machines. To cope with this problem, name-address mappings in local machines are distributed periodically and DNS agents store them in a file. However, this approach still has a drawback; machines that are often disconnected from the network may have invalid information about local domain names.

The resolution process of local names is slightly different from that of external names. Like other domain names, DNS agents send queries for the local names to neighbor agents. When a DNS agent receives a query for a local name, it checks the file of local mappings in addition to its cache. If a DNS agent cannot get a reply about the local names from neighbor DNS agents, it does not send the query to external name servers.

4.4.2 Evaluation

To evaluate our system, we built a simulator and used our real DNS traces as input to the simulator. We explicitly set the time when name servers fail. The simulator builds a cache status for each client based on our traces until a failure occurs. Then, it processes each query in a backup mode.

Each query is processed in one of the following four ways:

- **Local Hit**: replied with the help from a local cache
- **Pending**: suspended until a previous query for the same domain name is answered
- **Neighbor**: replied by a neighbor
- **Miss**: replied by a remote name server.

Figure 4.9 shows the overall cache hit ratio. The performance of a DNS agent in a backup mode is compatible with that of name severs in the absence of failures. For the two months of our data collection, the difference is smaller in July than in November. This is because the domains with shorter TTLs is less popular in November than in July as shown in Figure 4.2.

Figure 4.9 also shows the hit ratio when Subnet 4 is excluded. As mentioned in Section 4.3, Subnet 4 has unique characteristics because it includes `SMTP` and `PlanetLab` servers. Because these special servers experience a high cache hit ratio, the overall hit ratio without Subnet 4 is lower than the hit ratio with every subnet included. We expect that the former is more general than the latter.

DNS traffic from all subnets within a domain is targeted at local name servers in a normal mode, whereas traffic only within a subnet can be aggregated in a backup mode. As a result, servers receive queries more frequently in a normal mode than in a backup mode. Although domain names with short TTLs generate many misses in general, the negative effect of short TTLs is less serious in a normal mode than in a backup mode

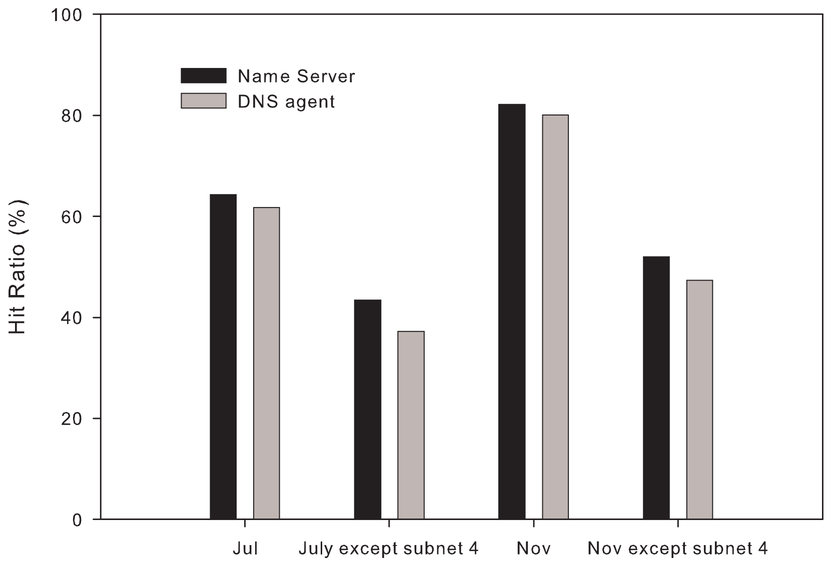
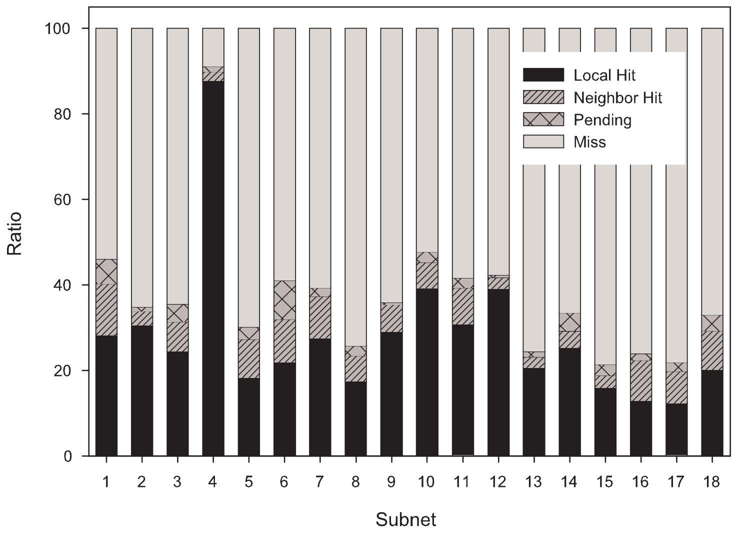


Figure 4.9: Overall cache hit ratio

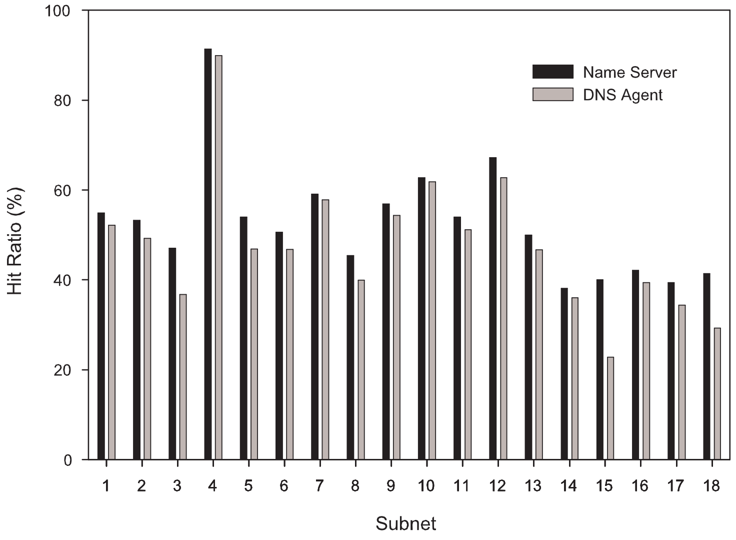
because information is refreshed frequently in a normal mode.

The cache hit ratio varies among subnets. Figure 4.10 shows cache hit ratio and how queries are processed. As shown in Figures 4.10(a) and (b), Subnet 4 exhibits the highest cache hit ratio by a big margin. Figure 4.10(a) shows how queries are processed in each subnet. Subnets 12, 13, 14, and 15 have a low neighbor-hit ratio because they have the smallest number of machines. Subnets with a large number of machines have a higher neighbor hit ratio. In summary, the neighbor hit ratio depends on the number of machines within a subnet.

One surprising result is that Subnet 16 has a very low hit ratio in July. Subnet 16 contains more machines than the average, and generated lots of network traffic. Thus, we expected it to have a high hit ratio. To understand the reason for this, we plotted the cumulative distribution of TTL for subnet 4 and 16 in Figure 4.11. Subnet 4 queried



(a) Cache activities in each subnet in July



(b) Cache hit ratio of each subnet in November

Figure 4.10: Cache characteristics in each subnet

domain names with longer TTLs than subnet 16; more than 50% of queries from subnet 16 are to domain names with TTLs less than 5 minutes.

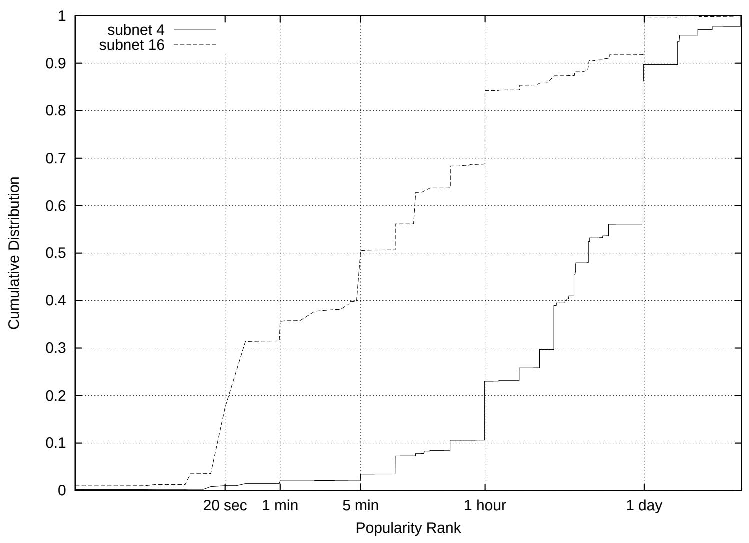


Figure 4.11: Cumulative distribution of TTL of two subnets

4.5 P2P DNS Agent

Although broadcast-based DNS agents are simple to implement, they have two drawbacks. First, because they broadcast a query, the query-reply process involves many clients. Second, broadcasting is not allowed beyond a subnet, which contains only a small number of clients. To deal with these problems, we propose P2P DNS agents that provide the same functions as the local name servers, but in a fully-distributed manner. Since DNS agents are based on a structured P2P system, which relies on unicast messages, the query-reply process involves a small number of clients. These clients can cooperate beyond subnet boundaries.

4.5.1 Operation of a P2P DNS Agent

In a normal mode, the operation of a P2P DNS agent is the same as that of a broadcast-based DNS agent (see Figure 4.6). Figure 4.12 shows the operations of a P2P DNS agent in a backup mode. When a DNS agent receives a query from the stub resolver, it tries to

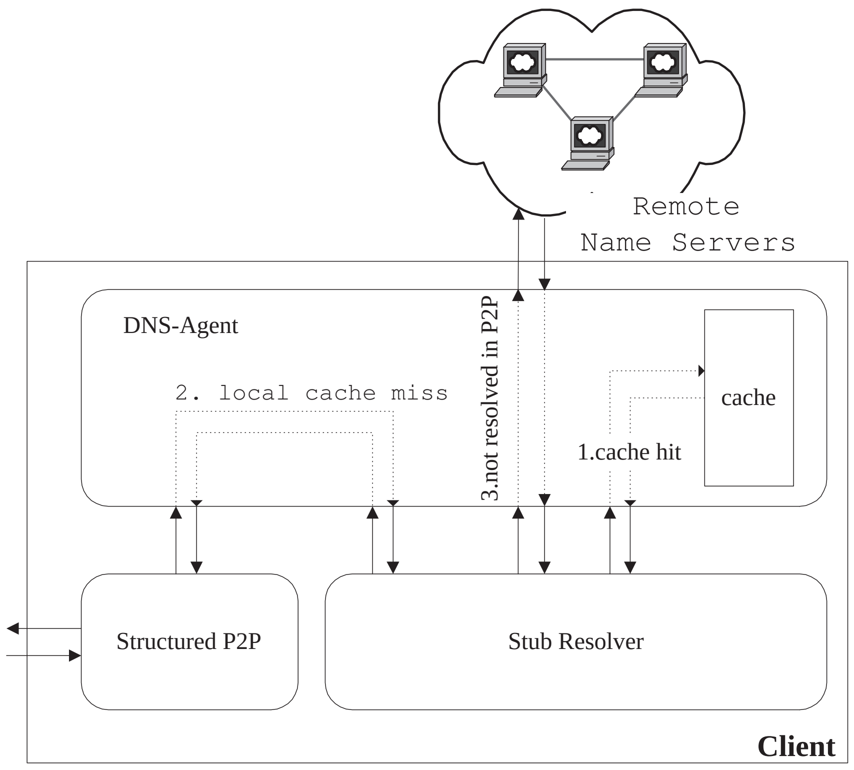


Figure 4.12: Operation of a P2P DNS agent when name servers fail

answer using its cache. However, not all queries can be answered using the cache. Thus, we propose a DNS agent that shares cache contents with other DNS agents over a P2P substrate.

The difference between a broadcast-based DNS agent and a P2P DNS agent is how they share cache contents. A P2P agent shares its cache contents using a structured peer-to-peer system like Chord, Pastry, Tapestry [38, 36, 41]. These systems locate objects within a finite number of hops, typically $O(\log N)$ where N is the number of nodes. A DNS agent sends a DNS query to the home node of the domain name when it does not have valid information in its cache. A home node is the node that has the closest node-ID to the object-ID of the domain name. If the domain name and RR mapping have been inserted into a P2P system and TTL has not yet expired, the home node has valid information about the name and replies to the queries.

If the home node does not have valid information, it sends back a `NOINFO` message to the requester. When a DNS agent receives a `NOINFO`, it contacts remote name servers. When the agent receives RRs of the domain name from a remote server, it stores RRs in its cache and inserts the data into the P2P system to share the data with other nodes. An alternate method is that a home node resolves the requested domain name and replies to the requesting node like a local name server. Even though this alternate method reduces the number of messages in the P2P system, we did not adopt this method because it can burden some nodes with popular domain names.

Our study of DNS traces study shows that a small number of domain names account for a larger portion of the queries. If a node is responsible for a very popular name with a short TTL, it will receive a large number of queries. By allowing a requesting node to resolve the name itself, the home node's burden of resolving the queried name is reduced. However, this home node may still be overloaded simply by the large number of queries and replies.

To alleviate this problem, we propose the *one-hop caching* mechanism: the node, which delivered the query message to the home node directly, will cache the reply information. Because in a P2P system, the reply message goes directly to the node that sent the query message, the last-hop node cannot receive the reply. In our system, when a home node sends back a reply message with valid information, it sends `CACHE-REC` message with the same information to the node from which the home node received the query. When a node receives a CACHE-REC message, it can either cache the object or

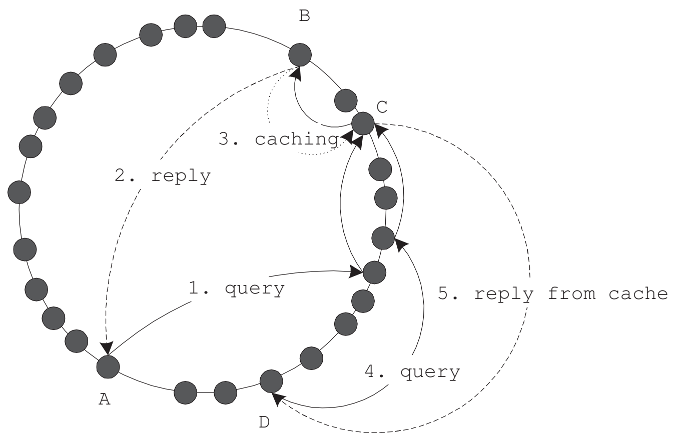


Figure 4.13: An example of replies from home node and caching node.

not. If it decides to cache the object, it stores the object in a cache that is separate from the DNS cache. This separation protects the items in the DNS cache from being replaced with the objects generated by requests from other nodes.

Figure 4.13 shows an example of a reply from a home node and a reply from a caching node. Node A sends a query about a domain name *X*, whose home node is B. The query message is routed through node C to node B. When node B replies to node A, it also sends `CACHE-REC` message to node C. Later, node D sends a query message for the same domain name *X*, and on its way to the home node B, the message is routed through node C. Before relaying the query message to node B, node C checks its cache and finds valid information in the one-hop cache. Thus, node C sends a reply message to node D instead of forwarding the query.

Although the P2P system is not used when the local name servers are alive, the P2P system is prepared in advance. When a DNS agent starts up, it initiates the P2P system. This consumes a little bit of memory and generates network traffic for joining the P2P system and maintaining the routing table. It eliminates the transition delay from a normal

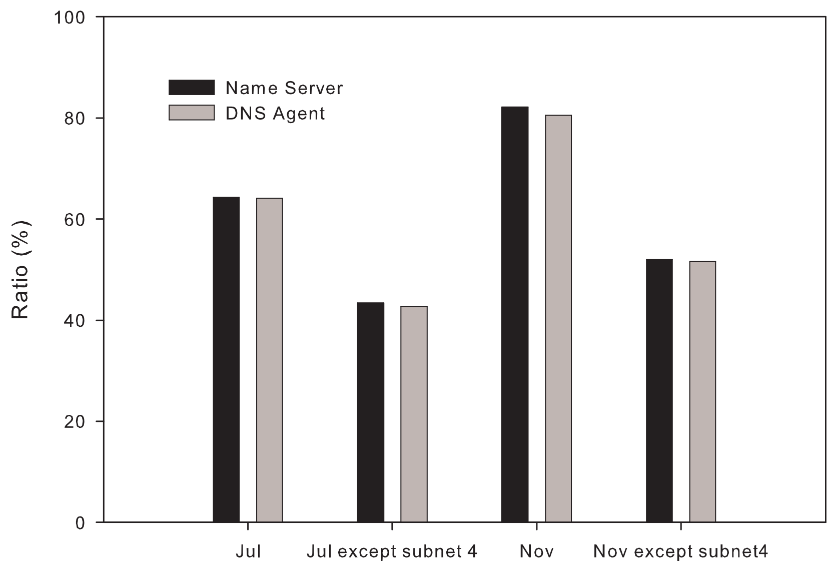


Figure 4.14: P2P: ratio of queries resolved within a domain

mode to a backup mode and prevents the burst of joining and bootstrapping messages of the P2P system when name servers fail.

To traverse a domain name space, a P2P DNS agent needs to know at least one root server. We use *root hints* for P2P DNS agents in the same way as for broadcast-based agents.

Since local name servers are often configured as authoritative servers for a domain, DNS agents cannot resolve the queries for local machines when local servers are unavailable. In our P2P system, every node inserts its name-RR mapping when it starts the P2P system. Since the node can leave P2P system ungracefully, the name-RR mapping can be lost. Thus, every local machine inserts its name-RR mapping periodically.

4.5.2 Evaluation

To evaluate our system, we built a simulator based on FreePastry 1.3 with our real DNS traffic traces as input. Pastry has a couple of configuration parameters, which affect the size of the routing table and the average hop counts. We used the values that Pastry recommends: leaf set size of 24, base of 16 (2^4) and key length of 128 bits.

A simulation starts in a normal mode. In this mode, DNS agents do not use the P2P system to share the cache. While relaying queries and replies between the stub resolver and name servers, DNS agents build up their cache. At the preselected time, name servers fail and DNS agents switch to a backup mode. This is when agents start using the P2P system.

Each query is processed in one of the following three ways:

- **Local Hit**: replied with the help from a local cache,
- **Peer Hit**: replied by a P2P system,
- **Miss**: replied by a remote name server.

Figure 4.14 shows the ratio of queries resolved without contacting remote name servers. A query can be resolved without the help of remote name servers, only when the information is cached in a DNS agent or in the P2P system. The performance shown in the graph can be read as the system-wide cache hit ratio. The performance of a DNS agent system in the backup mode is compatible with that of name severs in the absence of failures.

After name servers fail, every name$\rightarrow$RR mapping newly fetched from remote name servers is stored in the P2P system and used to resolve future queries. This is similar to

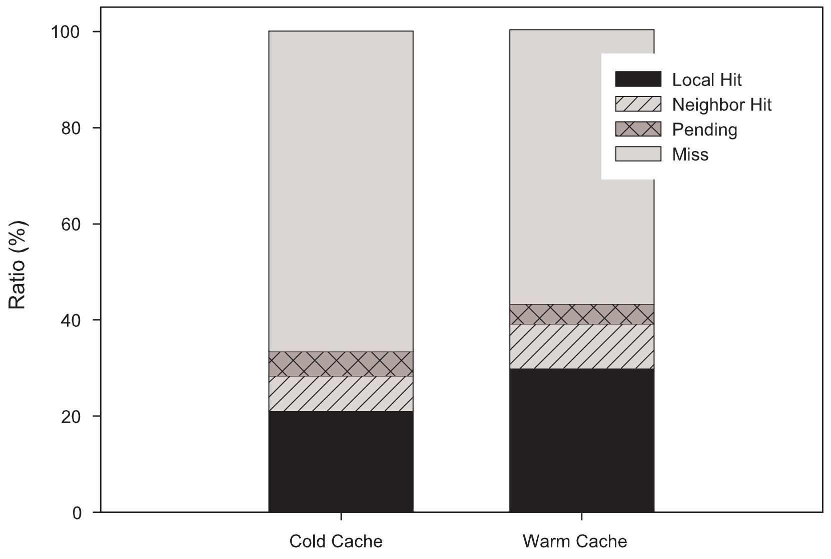
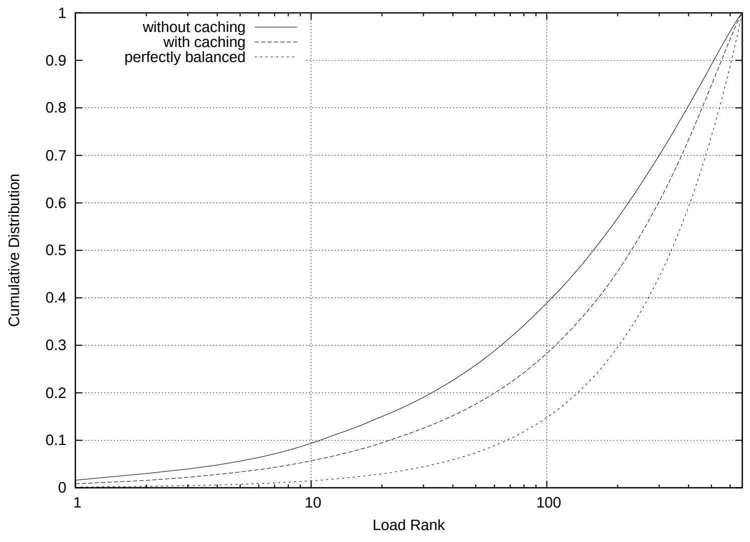


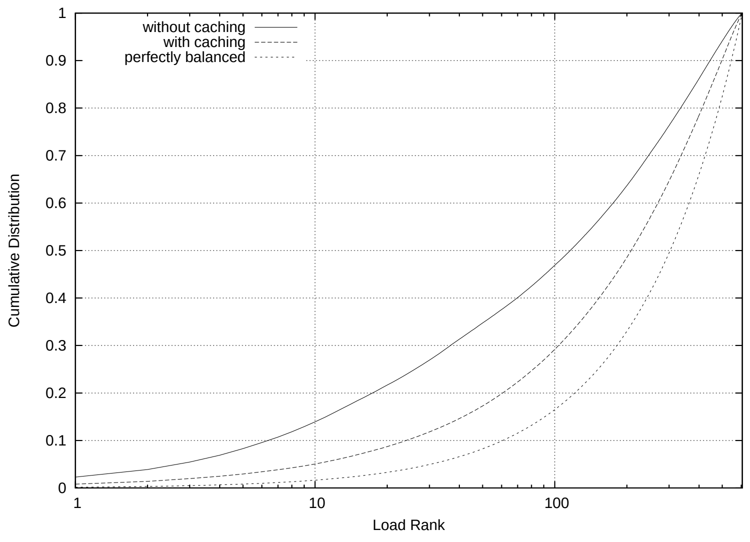
Figure 4.15: Comparison of cold and warm cache for the first one hour

the rebooting of name servers. Because the cache is empty at the startup, name servers generate only cache misses. However, the cache hit ratio improves rapidly since DNS queries show very high reference locality. The DNS agent also shows an about 7% lower cache hit ratio during the first one hour after the failures of name servers. One benefit of our system is that since the cache of a DNS agent is populated already, the lowest cache hit ratio at the moment of failures is about 15 – 20% rather than 0% of the rebooted name servers.

We measured the effect of caching items before name servers fail. For comparison, we disabled caching until local name servers fail. As a result, the cache was empty (or *cold*) when DNS agents switch to the backup mode. We compare this cold-cache policy with the warm-cache policy in which caching is not disabled. We measured the cache performance during one hour after name servers failed. After one hour, the cache was



(a) July



(b) November

Figure 4.16: Cumulative distribution of load of each node

warmed up enough, so the difference between cold and warm caches became negligible. Figure 4.15 shows the performance of two policies in subnet 12. *Cold cache* denotes the case when the caching is not disabled while *warm cache* represents the case when the caching is disabled until local name servers fail. For the first one hour, *warm cache* shows an about 11% higher cache hit ratio than *cold cache*.

In a P2P system, it is important to avoid overloading a node. People share their computing resource in a P2P system only when the system does not hinder other tasks and provides some benefits. We use one-hop caching to distribute load from the home node of a popular name to other nodes.

To measure the effectiveness of one-hop caching for load balancing, we tested the system with and without the one-hop caching mechanism. Figure 4.16 shows the cumulative distribution of the load imposed on each node. The solid line represents cache-disable system. The dotted line shows the distribution of perfectly balanced situation where each node is responsible for the same amount of load. The dashed line is the DNS agent system with one-hop caching. In November, ten most loaded nodes are responsible for about 14% queries when the one-hop caching is disabled. With the one-hop caching enabled, the load reduces to one third. Without the one-hop caching, the most loaded node processes 100 times more queries than the least loaded node. With one-hop caching, the ratio shrinks to 12.

Though one-hop caching alleviates load imbalance, some nodes are still more loaded than others. Because an intermediate node replies instead of relaying a query to the home node only when it can find valid information in its cache, queries, which cannot be resolved in the P2P system, always reach the home node. So, nodes, which are responsible for popular names with short TTLs, may be more loaded than other nodes.

In addition to load balancing, one-hop caching also shortens the average hop counts of query routing. Figure 4.17 shows the distribution of hop counts that a query message travels. The average hop counts of one-hop caching enabled and disabled systems are 1.87

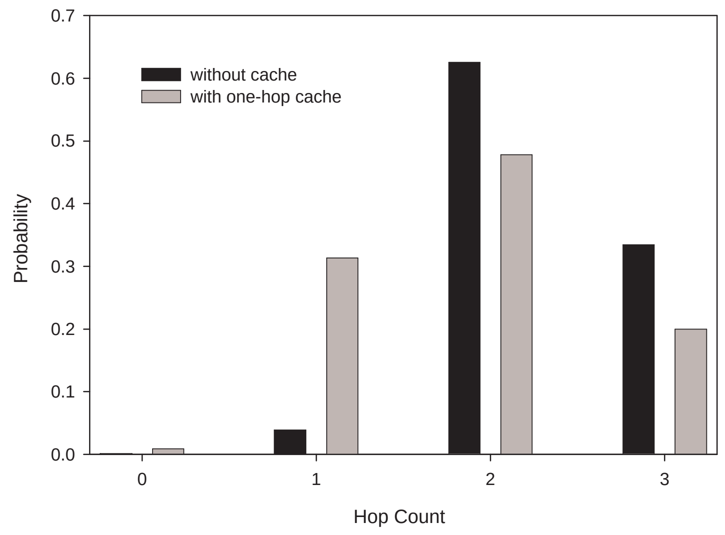


Figure 4.17: One hop: ratio of queries resolved within a domain

and 2.29, respectively. A smaller number of hop counts decrease the resolution latency and reduce the number of messages.

4.6 Conclusion

In the Internet, the name servers provide DNS service, mapping the human readable host names to IP addresses. Although the replication of name servers provide some fault-tolerance, it cannot guarantee the servers' availability against intentional attacks. If all the name servers within a domain fail, client machines within that domain cannot access the outside world.

Our backup plan enables clients to continue network accesses even when all local name servers are unavailable. This is done by sharing DNS information among neighbor client machines using a structured peer-to-peer system. Our system has two operational modes: normal and backup modes. Clients use name servers under the normal mode and use DNS agents under the backup mode.

We collected traces of DNS traffic in our department for two months. We chose the data of a week from each month and analyzed them. The overall cache hit ratio is very high, but there is a large variance in cache access patterns among subnets due to different usages.

To evaluate our backup plan, we built a simulator and used our traces as an input. The result shows that the performance of DNS service using our DNS agents is comparable to that of using name servers. In terms of overheads, the preparation cost for the backup plan is low; the backup plan uses only a small amount of memory at client machines. The simulation based on real traces shows that the DNS agent system improves the availability of DNS sigmificantly.

CHAPTER 5

Conclusion and Future Work

Many applications require performance and dependable guarantees, but the current Internet is unable to provide such guarantees. Computer networks fail due to component failures or malicious attacks. Applications that require real-time delay guarantees have not been deployed widely because they require timely recovery from component failures, which is not supported by the current Internet.

The methods proposed in this thesis improve the network dependability by preparing *a priori* backup plans with extra resources. When primary channels fail to improve the dependability of real-time channels, the proactive schemes prepare backup channels in advance. Using this proactive approach, we developed three schemes for routing backup channels. Two schemes are based on link-state routing and exchange the routing information for backups in link-state messages. The third routing scheme uses bounded flooding and does not require the distribution of link-state information. We evaluated the three routing schemes with simulation and showed that the link-state-based routing schemes discover backup routes with a higher level of dependability than the bounded flooding scheme. Link-state-based schemes, however, incur overheads of maintaining additional information in the expanded link-state database.

Although proactive schemes provide high dependability, they incur the overhead for maintaining per-channel-based routing information. To address this problem, we developed a hybrid scheme that pre-selects backup routes without reserving bandwidth for each backup channel. Because a certain amount of spare bandwidth is set aside *a priori* for backups, the hybrid scheme does not require global routing messages for the spare bandwidth, eliminating one of the main drawbacks of proactive schemes. We also devised a novel distributed routing algorithm that does not require nodes to keep information for each D-connection. The simulation results show that our hybrid scheme provides dependability equivalent to that of proactive schemes while lowering the routing overhead.

We applied the pre-planned backup method to the DNS service to improve its dependability. To better understand the DNS service, we collected DNS traces at our department. The trace shows that a small number of domain names account for most of queries and there are a large number of unpopular domain names. These results suggest that caching DNS lookup results would be effective, but it should be done in a cooperative manner to increase the hit ratio, especially for unpopular domain names.

We developed DNS agents that enable clients to resolve DNS queries when local name servers are not available. DNS agents are based on cooperative cache sharing of DNS information, which shows high reference locality. DNS agents provide continuous DNS service without increasing the load of outside machines. We evaluated DNS agents using the trace-driven simulation. The results show that the performance of DNS service using our DNS agents in case of local name server failures is comparable to using that of local name servers when they are alive.

In this thesis, we aimed to improve the dependability of the Internet. Our schemes provide compatible performance in case of failures without inducing high preparation costs and overhead of other components of the Internet when activated.

We explored a limited part of the dependability of the Internet. To improve the overall dependability, we need further research into several related issues. First, the deployment of dependable real-time connections should be addressed. Since it is difficult to equip existing routers with new features, an overlay network can be used for DR-connections. In an overlay network, links are not homogeneous because each link is composed of a different number of low-level physical links. Also, two apparently disjoint links in an overlay network can share the underlying links and routers. To deploy DR-connections in an overlay network, it is important to measure the characteristics of each link, such as end-to-end delay, available bandwidth, and reliability.

We also like to extend the DR-connection protocol to support multicast. Multicast is devised to minimize resource usage when same contents are delivered to multiple receivers at the same time. Typically, a multicast network is implemented as a tree topology rooted on the source. Because one link failure can disrupt several receivers, the failure recovery is more important in multicast than in unicast. Another difference is that in multicast, the impact of one link failure varies with the link's location. When a link closer to the source fails, more receivers are disrupted. Thus, the level of redundancy should be different according to the importance of each link.

It is also worthwhile to investigate the dependability of P2P networks. Since most P2P networks are distributed systems without centralized severs, P2P networks do not have a single point of failure. However, a P2P network is not stable because a P2P network changes continuously as nodes join and leave the network. Even though P2P networks have mechanisms to adapt to topology changes, they are not sufficient to overcome certain network failures which cause the disconnection of a large number of nodes. It will be interesting to measure the robustness of P2P networks and develop methods for fast recovery from a large-scale topology change.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Akamai. http://www.akamai.com.

[2] P. Albitz and C. Liu. *DNS and BIND*. O'Reilly, 2001.

[3] A. Banerjea. Simulation study of the capacity effects of dispersity routing for fault tolerance real-time channels. In *Proceedings of ACM SIGCOMM'96*, pages 192–205, Stanford, CA, 1996.

[4] A. Banerjea. Fault recovery for guaranteed performance communications connections. *IEEE Transactions on Computer Systems*, 7(5):653–668, Oct. 1999.

[5] A. Banerjea, C. Parris, and D. Ferrari. Recovering guaranteed performance service connections from single and multiple faults. In *Proceedings of IEEE GLOBECOM'94*, pages 162–168, San Francisco, CA, 1994.

[6] A. Basu, L. Ong, B. Shepherd, A. Rasala, and G. Wilfong. Route oscillations in I-BGP with route reflection. In *Proceedings of the ACM SIGCOMM'02.*, 2002.

[7] K. Calvert and E. Zegura. Gt-itm: Georgia tech internetworkr topology models. http://www.cc.gatech.edu/fac/ellen.zegura/gt-itm/gt-itm.tar.gz., 1996.

[8] C. Canali, V. Cardellini, M. Colajanni, R. Lancellotti, and P. S. Yu. Cooperative architectures and algorthms for discovery and transcoding of multi-version contents. In *Proceedings of 8th Web Caching Workshop (WCW 2003)*, 2003.

[9] Cisco Systems. Endless BGP convergence problem in Cisco IOS software releases. Cisco Systems Inc. Field Notice, Oct. 2000.

[10] E. Cohen and H. Kaplan. Proactive caching of dns records: Addressing a performance bottleneck. In *Proceedings of the 2001 Symposium on Applications and the Internet (SAINT).*, pages 85–94, 2001.

[11] C. Dovrolis and P. Ramanathan. Resource aggregation for fault tolerance in integrated services networks. *Computer Communication Review*, 28(2):39–53, Apr. 1998.

[12] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM Transactions on Networking*, 8(3):281–293, 2000.

[13] S. Han and K. G. Shin. Efficient spare resource allocation for fast restoration of real-time channels from network component failures. In *Proceedings of IEEE RTSS'97*, pages 99–108, 1997.

[14] S. Han and K. G. Shin. Fast restoration of real-time communication service from component failures in multihop networks. In *Proceedings of ACM SIGCOMM'97*, pages 77–88, 1997.

[15] S. Han and K. G. Shin. A primary-backup channel approach to dependable real-time communication in multi-hop networks. *IEEE Transactions on Computers*, 47(1), 1998.

[16] C. Huitema. *Routing in the Internet*. Prentice Hall, Mar. 1995.

[17] V. Jacobson, C. Leres, and S. McCanne. `tcpdump`. available via anonymous `ftp` to `ftp.ee.lbl.gov`, June 1989.

[18] J. Jung, A. W. Berger, and H. Balakrishnan. Modeling TTL-based Internet caches. In *Proceedings of IEEE Infocom 2003*, San Francisco, CA, Apr. 2003.

[19] J. Jung, E. Sit, H. Balakrishnan, and R. Morris. Dns performance and the effectiveness of caching. In *Proceedings of the ACM SIGCOMM Internet Measurement Workshop '01*, San Francisco, California, Nov. 2001.

[20] J. Kangasharju and K. W. Ross. A replicated architecture for the domain name system. In *Proceedings of IEEE INFOCOM*, pages 660–669, 2000.

[21] B. Kao, H. Garcia-Molina, and D. Barbara. Aggressive transmissions of short messages over redundant paths. *IEEE Transactions on Parallel and Distributed Systems*, 5(10):1044–1056, Jan. 1994.

[22] M. Kodialam and T. V. Lakshman. Dynamic routing of bandwidth guaranteed tunnels with restoration. In *Proceedings of INFOCOM 2000*, pages 902–911, 2000.

[23] S. K. Kweon and K. G. Shin. Distributed QoS routing using bounded flooding. Technical Report CSE-TR-388-99, University of Michigan, 1999.

[24] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian. Delayed internet routing convergence. In *Proceedings of ACM SIGCOMM'00*, pages 175–187, 2000.

[25] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian. Delayed internet routing convergence. *IEEE/ACM Transactions on Networking*, 9(3):293–306, June 2001.

[26] C. Labovitz, A. Ahuja, and F. Jahanian. Experimental study of internet stability and wide-area network failures. In *Proceedings of IEEE FTCS'99*, pages 278–285, 1999.

[27] C. Labovitz, G. R. Malan, and F. Jahanian. Origins of internet routing instability. In *Proceedings of INFOCOM99*, pages 218–26, New York, NY, Mar. 1999.

[28] G. Li, D. Wang, C. Kalmanek, and R. Doverspike. Efficient distributed path selection for shared restoration connectiosn. In *Proceedings of INFOCOM 2002*, pages 140–149, 2002.

[29] D. McPherson, V. Gill, and D. W. A. Retana. Persistent route oscillation condition. ietf internet draft draft-ietf-idr-route-oscillation-00.txt, Mar. 2001.

[30] P. Mockapetris. RFC 1034 domain names - concepts and facilities, Nov. 1987.

[31] P. Mockapetris. RFC 1035 domain names - implementation and specification, Nov. 1987.

[32] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. Inside the slammer worm. *IEEE Security and Privacy*, 1(4):33–39, July 2003.

[33] D. Moore, C. Shannon, and J. Brown. Code-red: a case study on the spread and victims of an internet worm. In *Proceedings of the Internet Measurement Workshop (IMW)*, 2002.

[34] V. Paxon. End-to-end routing behavior in the internet. *IEEE Transactio on Networking*, 5(5):601–615, Oct. 1997.

[35] P. Ramanathan and K. G. Shin. Delivery of time-critical messages using a multiple copy approach. *IEEE Transactions on Computer Systems*, 10(2):144–166, May 1992.

[36] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, Nov. 2001.

[37] H. Shang and C. Wills. Using related domain names to improve dns performance. Technical Report WPI-CS-TR-03-35, Worcester Polytechnic Institute, Dec. 2003.

[38] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.

[39] J. Wang. A survey of web caching schemes for the internet. *ACM Computer Communication Review*, 29(5), 1999.

[40] B. M. Waxman. Routing of multipoint connection. *IEEE Journal on Selected Areas in Communications*, 6(9):1617–1622, Dec. 1988.

[41] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and. Technical report, University of California at Berkeley, 2001.

[42] Q. Zheng and K. G. Shin. Fault-tolerant real-time communication in distributed computing systems. In *Proceedings of IEEE FTCS'92*, pages 86–93, 1992.

[43] Q. Zheng and K. G. Shin. Establishment of isolated failure immune real-time channels in harts. *IEEE Transactions on Parallel and Distributed Systems*, 6(2):113–119, Feb. 1995.

[44] C. Zou, W. Gong, and D. Towsley. Worm propagation modeling and analysis under dynamic quarantine defense. In *Proceedings of the ACM CCS Workshop on Rapid Malcode (WORM'03)*, 2003.

ABSTRACT

Improving the Dependability of Computer Networks

by

Songkuk Kim

Chair: Kang G. Shin

Current network protocols deal with network failures. Reliable transport protocols such as TCP can handle transient packet losses. Routing protocols such as OSPF can detect persistent failures and adapt to a new network topology. However, the time these mechanisms need to recover from failures is unpredictable, making them inadequate for real-time communications that require QoS (Quality-of-Service) guarantees.

A persistent failure can result in a long service disruption. To provide timely and successful recovery in case of persistent failures, we need to reserve spare resources *a priori*. This thesis investigates several ways to use spare resources to improve the dependability of real-time communications. First, we developed an algorithm for choosing backup routes. Spare resources, such as bandwidth, are limited and, thus, multiple channels can contend over these limited resources. To prevent contention among backup channels, we need to carefully select backup routes. We preselect backup routes to achieve fast recovery, and steer the real-time traffic disrupted by a link failure through the backup routes. Second,

we address the problem of managing spare resources. By reserving spare resources for each real-time channel, routers incur high overhead because they need to maintain routing information for each backup channel. As the number of backup channels increases, the cost of maintaining information may become unacceptably high. To cope with this problem, we reserve a pool of spare resources for all backup channels, and defer allocation of resources to a specific channel until they are needed.

In addition to failures of links or routers, the disruption of Domain Name System (DNS) affects the dependability of computer networks. The current DNS replicates name servers to improve its availability. Although the replication can overcome sporadic failures such as OS crashes and power outages, it is not effective for malicious attacks, which may cause all local servers to fail. We developed a distributed approach for DNS: each client machine maintains a small DNS cache and shares the cache contents with other client machines. Our approach provides a continuous DNS service even when all local name servers fail.