The University of Kansas

A Technical Report of ITTC's
Networking and Distributed Systems Laboratory

On the Effects of Pre-Computed Route Caching in Multiple Peer Group ATM-PNNI Networks

Kamalesh S. Kalarickal
and
Douglas Niehaus

ITTC-FY2001-TR-18833-11

August 2000

Project Sponsor:
Sprint Corp.

Abstract

Routing within Asynchronous Transfer Mode (ATM) networks has changed since the introduction of the Private-Network-to-Network Interface (PNNI) protocol by the ATM Forum. Among the many features of PNNI are the facilities it provides for scalable and hierarchical network configurations utilizing the concept of Multiple Peer Groups (MPG). This thesis represents the research conducted using the KUPNNI simulator supporting MPG for PNNI. Specifically a comparison is made between on-demand routing and pre-computed routing strategies within a multiple peer group network context to understand the trade-offs resulting from caching routes.

We describe a simple route caching strategy and investigate its effect on call setup times and call success rates, as a function of peer group size, call load, and pre-computation cost. We then look at the problem of deciding when to initiate a pre-computation of routes to update the route cache. Different heuristics for determining when to start route cache updates are investigated. Finally, we propose a new heuristic, based on the number of topology update messages received for each level of the PNNI hierarchy. We conclude that our *ptse count* heuristic, when used in combination with *crankback initiated invalidation*, is effective in reducing the average call setup time, while achieving accuracy comparable to on-demand route computation.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

"Where shall I begin, please your Majesty?" He asked.
"Begin at the beginning," the King said, very gravely,
"and go on till you come to the end:
then stop."

As networks grow bigger and more interconnected, user demands for reliable, high speed services with Quality-of-Service guarantees increase the constraints on the network. One way of attacking this problem is to provide faster links with increased bandwidth. However, use of effective routing strategies is another means for coaxing better performance from an already stressed network.

Asynchronous Transfer Mode (ATM) is now widely recognized as an important networking technology. ATM specifies quality of service (QoS) guarantees that a network must provide for and manage. In large ATM network clouds, the Private-Network-to-Network Interface (PNNI) protocol provides the infrastructure to efficiently manage customer connections with QoS guarantees. The ATM Forum's PNNI standard [6] specifies a routing protocol for distributing topology and load information throughout the network, and a signalling protocol for processing and forwarding connection establishment requests from the source.

Quality-of-service routing has the potential to optimize the use of network

resources, and increase the success rate of accepting new connection requests, by selecting paths based on existing network load and connection traffic parameters [17, 23, 7]. However, distributing link load information and computing routes for new connections can consume considerable bandwidth, memory, and processing resources [22, 1]. Controlling these overheads in large backbone networks introduces a trade-off between performance and complexity.

1.1 On-Demand Routing

ATM Private Network to Network Interface routing relies on a dynamic link state based routing protocol that computes routes at the source node based on information contained in its topology database. A default model for computing the route is defined in the ATM Forum PNNI specification [8]. This involves *on-demand* routing at every node wherein the following operations occur every time a routing request is made:

- a graph of nodes and links, representing the network topology, is constructed and populated with information from the nodes database.
- a Generic Call Admission Control is carried out on this graph based on the call connection requirements. This process, also called *pruning*, eliminates all those links in the graph that cannot support the call requirements.
- Dijkstra's Single Shortest Path algorithm is run on the pruned graph so as to find a path from the source node to the destination node. This path minimizes a cost metric that is specified in the routing policy for the node and also satisfies the QoS requirements for the call.
- a Designated Transit List (DTL) is created for the route and transported in the setup message that then goes out to the next hop.

1.2 Problem Statement

The problem with this model is that it involves expensive graph manipulations and link state topology database queries for every call request that comes into every switch for source routing. There is scope for amortizing the overhead of route computation over multiple call connection requests, simply by *pre-computing* routes, or portions of a route, to destination nodes in the network.

1.3 Pre-Computed Routing

Pre-computed routes are computed and maintained in advance by the network, independently of subsequent routing requests. This offers better performance in terms of connection setup time since the node just fetches the correct entry from the routing table, without the burden of a costly route computation. Nevertheless, this method has drawbacks since the performance of pre-computed routes is bounded by the reliability and accuracy of information stored in the route cache at the time the route is effectively used. Pre-computation needs to compute routes that are used often, and in the worst case, routes to all possible destinations. The efficiency of pre-computing routes is derived primarily from the trade-off between the number of routes that are computed and the probability of consulting a specific entry.

When a call connection request arrives at the source node, an appropriate choice is made from the pre-computed set of routes. A wrong choice would be signalled by a PNNI crankback event, and on-demand path computation can replace the unsuccessful cache selection. In this way the pre-computed routes can also be categorized as *cache* entries in a cache of previously successful routes.

Given that such a *pre-computed routing* strategy exists, there also exist several tradeoffs where the following decisions need to be made:

- How should the pre-computed route cache be filled ?
- How many routes should be stored in the cache ?
- What is the cost of route cache pre-computation ?
- How should a route be selected from the cache for a given call QoS requirement ?
- What is an appropriate response to a cache miss ?
- When is a cache entry invalidated ?
- When and how often should cache updates be made ?
- What is the cache replacement policy ?

The question of pre-computed routes has been addressed by several other researchers [2, 4, 11, 14, 20]. Most of their work has centered around refining the cache policy decisions listed above. This has been applied to PNNI networking in some cases [14]. Most of the pre-computed routing studies have focussed on a flat, single peer group topology. Only one of the studies make use of the topology update mechanism to measure flux within the network [20]. Further, none of the studies consider the effect of route caching on Multiple Peer Group PNNI networks.

1.4 Our Solution

Our solution defines a pre-computed route caching policy that closely models the on-demand routing procedure described above. This is done intentionally and with the aim of bringing out the improvements due to pre-computed route caching over on-demand routing, without excessively changing the basic routing methodology.

Our study focuses on Multiple Peer Group (MPG) PNNI topologies. The concept of peer groups is intended to enable scaling within a PNNI network. Peer groups are a collection of nodes that are represented as a single logical node within the next higher level of an addressing hierarchy. Aggregated information about the resources in the nodes and links within the group reaches nodes in other peer groups by the process of *topology aggregation* followed by link state flooding. Routing decisions are made using the aggregated information. As the network dynamics change, the nodes decide if there is significant change in the resources, and if so, the new set of resources within the peer group are re-aggregated and flooded.

In a pre-computed routing strategy, updating the entries in the route cache is a way of improving call success. The cache is refreshed by new routes computed based on newer information present in the link state topology database. We attempt to discover a correlation between the *topology re-aggregation* and the pre-computed *route cache updates* based on the following intuitive observation:

> Whenever there is change in network resources, the set of pre-computed routes grows stale. The arrival of 're-aggregated' topology information indicates that there is significant change in the network. Hence updating the cache of pre-computed routes in response to a re-aggregation event would provide appropriate quality control to the pre-computed route cache.

Extending this further, we propose a quantitative method for determining when a set of cached routes need to be updated. By counting the number of new topology elements that arrive due to significant change and flooding, we have a measure of network flux. This is in turn used to initiate an update to the pre-computed route cache.

1.5 KU PNNI Simulator

We conduct our experimentation on the KUPNNI Simulator, which implements ATM Forum PNNI specification version 1.0 and UNI 3.0 [13].

The network that is modeled executes in virtual time and the network entities are abstracted up to but not including actual physical ATM cell transport. Once a node has decided to send out an AAL5 packet, there is a transfer from virtual to real time space. The links that transport this message are simulated within the discrete event simulator engine, and the message is transported with the appropriate virtual time link delay.

The virtual time management within the simulator is very lightweight and simple in its implementation. This allows for minimal delay in processing events. However, it should be noted that the simulated network can scale within the constraints of the host computer, and the simulation itself is still guaranteed to be correct, albeit slower than a small scale network.

Furthermore, we use the *real code* that networks and systems could, and often do, use. The simulations run on ATM signaling support which is the same as that used in an off-board signaling architecture (Q.Port)[3]. Because it uses real system networking code, it does not require the implementation of system code abstractions into a software simulator as is required in BONeS, OPNET, and other commonly used simulation packages.

1.6 Organization

This report is divided into five further chapters.

Chapter 2 discusses related work in the area of source initiated routing in general and pre-computed routing in particular. Chapter 3 describes the implementation of the pre-computed route cache and the quantitative method to determine when pre-computed route cache entries should be updated.

Chapter 4 presents our experimental methodology, the factors influencing our evaluation, and describes the choices we make regarding these factors. Chapter 5 describes the experiments performed. Each experiment establishes a hypothesis. The performance metrics, the parameters affecting those metrics, and the methodology are described for each experiment.

Finally Chapter 6 presents the lessons learned and our conclusions. It also discusses future work that could be carried out with regard to pre-computed route caching.

Chapter 2

Related Work

"Would you tell me, please, which way I ought to go from here?

"That depends a good deal on where you want to get to," said the Cat.

Previous work on pre-computation of routes has mainly focussed on the caching policy, and performance of the caching algorithm itself. There has been research on path caching with respect to storage of routes in efficient data structures, and considering different policies for updating and replacing the pre-computed routes.

Apostolopoulos et al., evaluate the performance and processing overhead of a specific path pre-computation algorithm [2]. The study adopts a Bellman-Ford based algorithm for route computation. It evaluates a purely periodic route cache update scheme. This is done under a variety of traffic and network configurations [10]. The study presents a detailed cost model of route computation to compare the overhead of on-demand and pre-computed strategies. While this study looks at how to improve the performance of the caching mechanism for flat networks, our study looks at a simple caching policy and how changing its parameters affects call performance with respect to Multiple Peer Group hierarchical networks.

Another study proposes a set of route pre-computation policies that optimize various criteria, such as connection blocking and setup latency [14]. The

algorithms try to locate routes that satisfy several QoS requirements through an iterative search of pre-computed paths (optimized for hop-count). This is followed, if necessary, by several on-demand calculations that optimize different additive QoS parameters, one at a time. This study primarily focuses on PNNI networks running on single peer group topologies, while our solution focuses on multiple peer group topologies.

As part of a broader study of QoS routing, Ma et al., evaluate a class-based scheme that pre-computes a set of routes for different bandwidth classes [19]. The evaluation compares the performance of several algorithms for class-based path computation to on-demand computation. This approach is similar to our method of quantizing QoS constraints into equivalence classes. Guillen et al. have proposed only to pre-compute and maintain frequently used routes (equivalent to the caching principle) [11]. But they do not consider how to update the cached routes.

Peyravian et al., introduce a policy that invalidates cache entries based on the number of link-state updates that have arrived for links in the pre-computed route [20]. The proposed algorithms also check the current link-state when selecting a path from the cache and allow re-computation when the cached paths are not suitable. We use a similar method to determine network flux, but while they invalidate the cache entries, we use the information to initiate route cache updates.

The remaining studies consider different ways to pre-compute paths for multiple destination nodes and connection QoS requirements. The work by Guerin et al., proposes a Dijkstra-based algorithm that computes minimum-hop paths for different bandwidth classes [10]. We use a modified version of this algorithm for pre-computing the route cache entries, by using equivalence classes of available bandwidth.

Another algorithm, introduced in Przygienda et al., pre-computes a set of external routes to all destinations such that no other route has both higher bottleneck bandwidth and smaller hop-count [4]. The Bellman-Ford-based algorithm

used by Guerin et al., has a similar optimization criterion for constructing a next-hop routing table with multiple routing entries for each destination [10]. We also optimize the pre-computed routes based on the hop-count, and use the optimized list of routes within our cache replacement policy. The emphasis of these last three studies is on algorithmic issues, such as reducing complexity of cached route computation.

Our solution is unique in that it uses the intuitive result that the arrival of a topology state packet is an indication of network flux. This is particularly important given the nature of information flow within a PNNI hierarchy. The link-state flooding and topology aggregation processes in Multiple Peer Group PNNI distribute routing information at different rates. This information ages at varying rates. The dynamics of where the information is generated and where it is used play an important role in how a route cache policy performs.

We investigate these issues and consider how pre-computation affects performance in Multiple Peer Group PNNI, using call setup time and the call acceptance ratio as the metrics of comparison. We also look at route cache update policies and propose a new heuristic, that reflects most recent link-state information, based on topology message count and cache entry success.

Chapter 3

Implementation

"Well, I'll eat it," said Alice,
"and if it makes me grow larger, I can reach the key;
and if it makes me grow smaller, I can creep under the door;
so either way I'll get into the garden,..."

In this chapter, we focus on our implementation of the pre-computed route caching mechanism. A detailed description of the PNNI protocol, with the signalling and routing components is available elsewhere [21]. Further, the working of the KU-PNNI simulator and its implementation of the PNNI protocol are described in the KU-PNNI User Manual [13].

3.1 Overview

First we give a brief description of PNNI routing for multiple level hierarchical topologies. Then, we describe the route caching policy used in our solution. We intend to bring out the benefits of a *pre-computed* routing strategy over *on-demand* routing. Hence we need to ensure a fair comparison. We retain the routing methodology used in *on-demand* routing, and make minimal changes to the actual route processing invocation within the PNNI router module.

We describe the way route cache updates occur, and then propose a quantitative method for determining when a pre-computed route cache update is required. Finally, we list the limitations and assumptions in our model.

3.2 The PNNI Routing Protocol

In this section, we focus on the PNNI routing mechanism and present a brief overview of how it works. More details and a thorough description of PNNI signalling can be found in the ATM Forum PNNI specifications [6]. In general, routing in an ATM network can be described as a mechanism that occurs on a call-by-call basis. To establish an end-to-end connection, a path comprising a sequence of PNNI-ATM nodes and transmission links has to be selected. The source node chooses a route based on its view of the network and the available resources. A setup message is propagated bearing the route inside it, in the form of a Designated Transit List (DTL). The intermediate nodes decide whether they can support the call with its request QoS parameters, and if they can, they allocate the required resources along the path. A set of supporting protocols and algorithms helps to efficiently perform these activities. The following subsections discuss them in more detail.

3.2.1 Routing and Path Selection

Path selection is usually implemented as a shortest path optimization problem. This can be done on-demand for every call, or the overhead of computation can be amortized over several calls by using some form of route caching, as is described in this thesis. Since ATM networks provide a guaranteed Quality of Service (QoS), each shortest path has to be chosen such that QoS constraints are fulfilled. The constraints can be additive metrics, like the maximum permissible end-to-end delay. On the other hand, they could also be a combined attribute like the maximum

available bandwidth. The metrics are computed over all the links in the path.

Like other dynamic routing protocols, such as Open Shortest Path First [5], PNNI uses a protocol to distribute information required for the path selection process. PNNI nodes exchange messages, called PNNI Topology State Elements (PTSE), to acquire knowledge about network topology and the state of nodes and links.

Nodes run a permanent neighbor detection computation. If a node learns about a new neighbor node it initiates a topology database synchronization in order to achieve a common view within a selected group of nodes, called the peer group (see Section 3.2.2). Then, a flooding process is started which distributes topology information. In case of network failures, nodes are able to recover and, given the physical paths, to remain aware of connectivity to other nodes using the mechanisms described above. Hence, nodes can be automatically removed from or inserted into the network.

In contrast to routing protocols used in the Internet, PNNI routes connections through the network based on source routing; the source node selects a path. However, taking scalability into account, a PNNI node determines only a hierarchically complete source route according to the PNNI hierarchy (see Section 3.2.2). A hierarchically complete source route contains a sequence of nodes, be it switching systems or logical nodes representing a collection of switching systems in a higher level abstraction of the hierarchy.

The accuracy of the route computed at the source depends on the accuracy of the information in the topology database. Further, the representation of peer groups as logical group nodes introduces topology aggregation. While this helps scalability, it comes at the cost of decreasing accuracy of aggregated information. With source routing, however, loops in the path can be avoided easily thus increasing routing stability.

During source routing, the node performs Generic Call Admission Control

(GCAC) to decide whether the call can be supported in the network. Additionally, during call setup, each ATM switching system along the chosen path applies Connection Admission Control (CAC) to enforce the required QoS on the respective links. Here, a switching system has to decide whether the new connection can be accepted or not, depending on the availability of sufficient resources to establish the connection.

In the case of insufficient resources, the crankback mechanism is initiated to inform the source node of a bad route. Crankback in combination with alternate routing is a mechanism to resolve call blocking by clearing the call back to the last node which has added information to the hierarchically complete source route. This node has the chance to decide about alternate paths.

3.2.2 Hierarchy Mechanism

PNNI routing strongly relies on a hierarchical network representation. It is derived from clustering mechanisms and topology aggregation methods which are recursively applied in an arbitrary number of levels. Figure 3.1 shows a three-level hierarchy as an example.

At the bottom, ATM switching systems (nodes) are grouped together forming a set of peer groups. Since peer group members are identified by a common peer group identifier, it is the responsibility of the network administrator to assign nodes to a peer group. At the next level of the hierarchy, the process is repeated, based on the peer group identifier, thus collecting logical nodes in a higher level peer group. At this level, logical nodes are connected via logical links being mapped to virtual connections at the lowest (physical) layer. A peer group is represented by a peer group leader, darkened nodes in Figure 3.1, that mainly performs topology aggregation and advertisement on behalf of the group members, as discussed in Section 3.2.3. In the next higher level of the PNNI hierarchy, the representation of the peer group is called a logical group node.

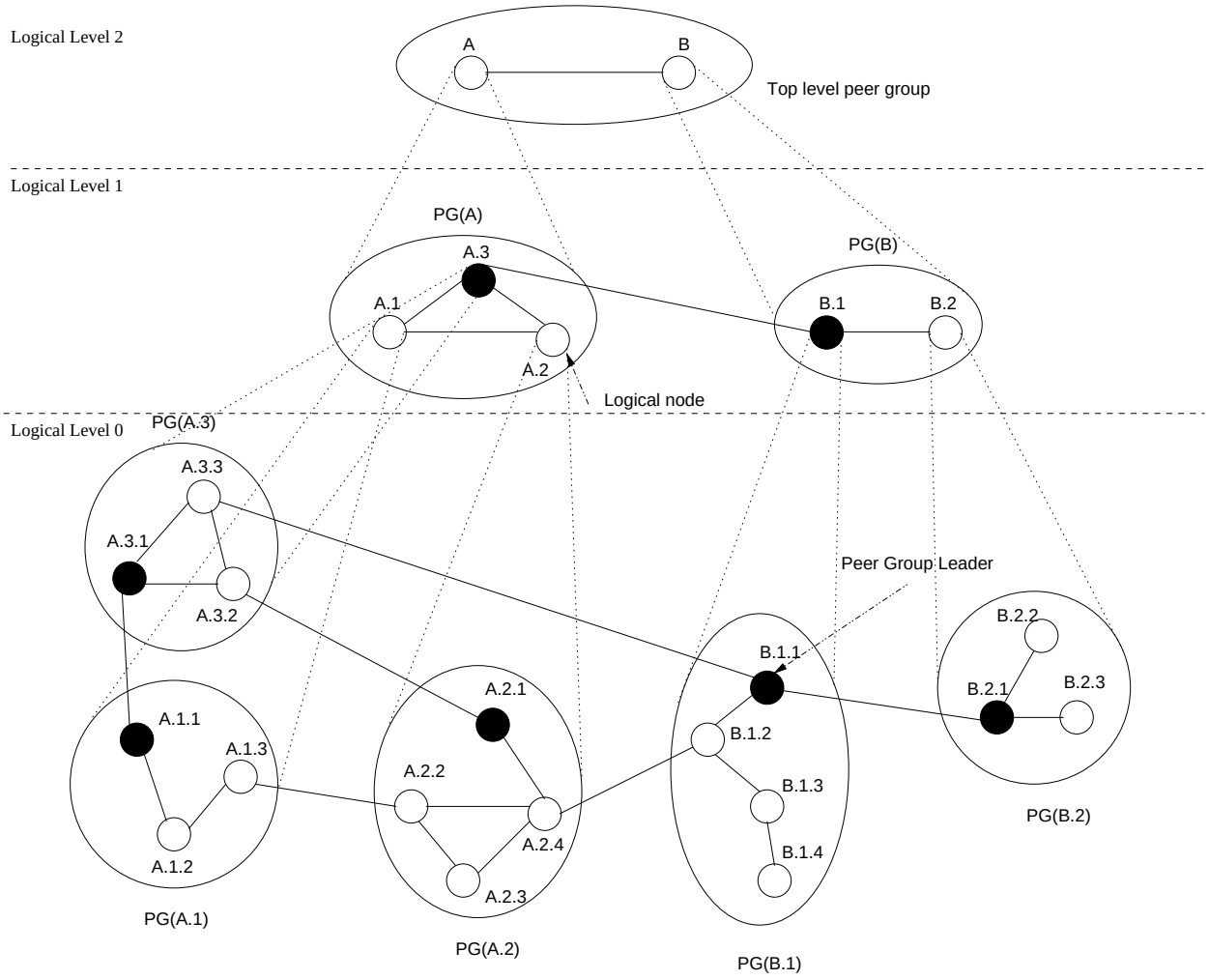

Figure 3.1: Hierarchical PNNI Topology

In each peer group, protocols for flooding link state information and detecting neighbor nodes are run to ensure connectivity and network state information update. Moreover, induced by the hierarchy support mechanisms, routing control channels are setup for information exchange between different nodes at each logical level of the PNNI hierarchy. They in turn flood the information received from within their peer group down to their child peer groups.

3.2.3 Topology Aggregation

A node or link is described by topology state parameters, such as available cell rate or cell loss ratio. To reduce the amount of routing information distributed throughout the network, a complex process of summarizing and compressing topology state information is performed at each hierarchical level. This is called topology aggregation. However, the reduction necessarily induces information loss. Thus, at higher levels of the network hierarchy, topology aggregation cannot represent the actual network state with perfect accuracy. This may result in two effects. On the one hand, calls which are routed according to overly simplified information may be blocked due to insufficient resources during call setup. On the other hand, resources may be underutilized if calls are blocked at the source node due to a lack of available resources being accurately represented to the source node.

3.3 Information Flow and Aging in MPG PNNI

Our main contribution is the way in which we determine when a route cache update is to be carried out. For this we use the intuitive understanding of the quality of topology state information and its longevity within a MPG PNNI hierarchy.

Consider the three level Multiple Peer Group PNNI topology shown in Figure 3.1. For any logical level 0 (physical) node, the information within its database can be categorized into the following based on its quality:

- Category A: This represents the best, most up-to-date information describing the resources of this node and all links starting from or terminating in it.
- Category B: This represents the next best quality information which is the link resource information and nodal information within the node's own peer group.
- Category C: This represents lower quality information which is the link and nodal resource information of other peer groups at the same logical level (level 0).
- Category D: This represents still worse quality information which is of the link and node resources at the next higher level of aggregation, which is logical level 1, in this example.
- Category E: Finally this represents the worst quality information which is of the links and nodes at the highest level of aggregation, logical level 2, in this example.

This can be extended to network topologies having 4, 5, or a higher number of levels. It should be noted that as the number of levels increases, the quality of aggregated information decreases, since the information lost due to topology aggregation increases.

The PNNI Topology State Elements (PTSE) are associated with th nodes that create them. They are flooded to other nodes and inserted into topology databases at each receiving node. Aggregated information from logical group nodes is flooded in their logical level, and flooded down to the children by the peer group leader. The topology databases are used to compute routes. The aging of the information elements depends on:

- When it was created by the owner node;

- The delay experienced by the information element to reach other nodes, which is variable depending on the network topology and conditions; and
- The threshold limit for determining that there is a significant change in the resources to warrant a flooding of the information element by the owner.

The different information elements are thus updated at different rates. Category A information is updated as soon as there is significant change in the local link resources. Category B is updated as and when other nodes in this peer group have significant change in their link and nodal states. Finally, the higher aggregation level information elements (Categories C, D and E) are updated according to the *re-aggregation* policy that is followed. If there is an explicit time interval for re-aggregation, these information elements are updated at that re-aggregation rate.

3.3.1 PTSE Count as a Determinant of Flux

The fact that PTSE's arrive at a node because of change in the resource availability in the network can be used as a measure of flux in the network. Keeping a running count of the number of update PTSEs gives us a quantitative measure of this flux. In addition, this measure is associated with specific network components and is available, essentially, for free. It also provides a quantified method which we can use to initiate route pre-computation for updating the pre-computed route cache.

The use of *PTSE count* to create a heuristic for determining cache updates is evaluated in this thesis. Further, it is shown that this measure enables network performance to reach the levels of on-demand route computation in terms of average call acceptance, while also benefiting from the reduced average call setup times associated with pre-computed route cache lookup.

3.4 Route Caching Policy

The pre-computed route caching strategy is described below in terms of answers to the questions outlined in Section 1.3. The questions help in describing the properties of the cache, the policy decisions used to determine its behavior, and the data structure used in its implementation.

3.4.1 How should the pre-computed route cache be filled ?

We propose to have a cache entry for every unique node in the source node's database. This includes all nodes that are within its own peer group and all higher level logical nodes to which it is connected. For each of these destinations, we compute routes that satisfy the Available Cell Rate (AvCR) QoS constraint.

Routes are stored as a binary tree indexed by the destination node address. Each destination has a route cache entry, which is itself a sorted list of route entries. Any given route entry has a Designated Transit List to reach the destination node, and an associated *effective bandwidth*, which is the maximum bandwidth that the links connecting the nodes in the DTL can support. The different routes are sorted based on the number of hops it takes to reach the destination. The route cache data structure is shown in Figure 3.2.

Understanding the need for a *granularity* measure in determining *equivalence classes* of bandwidth, we divide the range of Maximum AvCR to Minimum AvCR into 'N' quantized levels, corresponding to bandwidth equivalence classes, similar to Guerin et al. [10]. For each of these classes, a route is pre-computed for every destination in the network. The procedure for doing this is as follows:

- A representation of the topology is constructed as a graph of nodes and links and populated with information from the nodes database.
- A Generic Call Admission Control is carried out on this graph for requirements that are constructed depending on each zone. Thus the graph is pruned

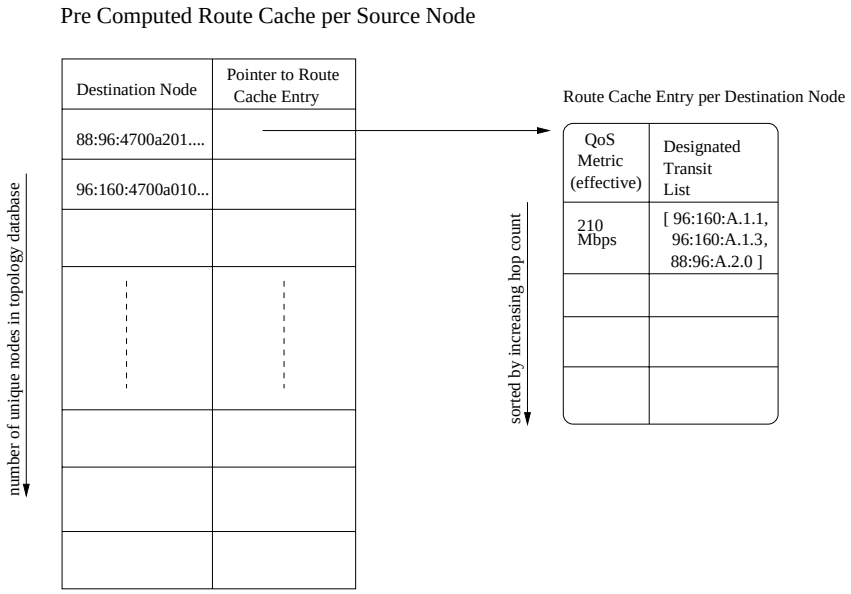

Figure 3.2: Structure of Route Cache

for each equivalence class of bandwidth before the routing algorithm is run.

- Dijkstra's Single Shortest Path algorithm is run on the pruned graph so as to find a path from the source node to the destination node. This path minimizes a cost metric that is specified in the routing policy for the node.
- A Designated Transit List is created for the route to be stored in the pre-computed route cache.

Algorithm 3.1, which fills the route cache, is presented below. It is to be noted that the algorithm chooses to go from a higher bandwidth quantization level to a lower bandwidth quantization level while pre-computing routes. This does not affect the routing decision, because the routing algorithm is called independently for each case. The nature of the network determines which route is available for the quantized bandwidth that is chosen as the constraint. It could even be that for certain network topologies, higher bandwidth ranges do not give any routes, or give very long routes. In fact this is seen in the experiments conducted in Section

5.3.2.

3.4.2 How many routes should be stored in the cache ?

Pre-computation of routes introduces a tradeoff between the processing overhead and the quality of the routes that are stored. We can control the size of the pre-computed cache and study the performance of the network as this size is changed. This is done via the tunable parameter representing the number of quantization levels, 'N'. It might be that, depending on the routing policy and the network topology, there are not enough routes for each of the quantization levels. In the worst case, if there exist unique routes stored for each quantization level in the cache, there will be 'N' routes per destination node.

Thus, depending on the routing policy that is followed, we have *(No. of Nodes) * N* possible routes, all of which minimize the cost specified in the routing policy.

3.4.3 What is the cost of route cache pre-computation ?

Every route cache pre-computation involves some cost. This can be in terms of CPU cycles consumed and call connection requests missed, to name two. In order to model the cost of updating the cache, we use the following algorithm:

1. First, we measure the real execution time it takes to update the route cache on the CPU on which the simulation runs.

2. We introduce a tunable *scaling factor* by which the measure of real execution time is multiplied to get the "equivalent" computation time on a real switch.

3. Assuming that the route cache cannot be used during the time it is being updated, we disable the route cache for a period of virtual time equal to the *scaled* computation time.

Algorithm 3.1 Filling the Pre-Computed Route Cache

```
Condition: Receipt of route cache update request.
Variable: Number of Quantization Levels (N)

% find range of bandwidths available in
% the topology database
for all links in the topology database
    find maximum available bandwidth (max_avcr)
    and minimum available bandwidth (min_avcr)
end

% find unique nodes in the topology, other than myself
for all nodes in the topology database
    if the node is NOT myself or my peer group leader
        add to a list of unique nodes
end

% create quantization levels
for range of bandwidth values [max_avcr min_avcr],
    create N Quantization levels, equally spaced,
    in decreasing order
end

% main loop
for all unique nodes
    % generate route for each quantization level
    for each  Quantization Level Q[n]
         prune all links that cannot
            support Q[n] bandwidth
         call Dijkstra's routing algorithm
         create destination route to each unique node
         calculate effective bandwidth of route

         % invalidate cache before replacing
         delete cache entry for that destination
         store route cache entry for the destination
    end
end
```

4. Disabling the route cache is carried out by turning on the “busy” flag. All subsequent routing requests are rejected, and classified as route cache lookup failures.

5. The cache is re-enabled after the computation time has elapsed, by turning off the “busy” flag. The newly updated cache is now ready for use.

It is noted by Peyravian et al. that network route caching varies substantially from caching used in other contexts (memory systems, multiprocessors, distributed systems, network file systems) in the following ways [20]:

1. The penalty of using an outdated cache entry is greater due to the time it takes for the failure to be discovered.

2. Cache size, which is a constraint in traditional memory system caches, is not as significant a constraint in caching network routes. This is primarily because the number of route entries stored in the route cache would be fewer than say the number of pages or variables in a memory or distributed system.

3. It is easier to determine the validity of a cache entry in memory / multiprocessor caches, via a boolean setting of valid / invalid. However, validity of routes is a much more complex and often impossible decision, which requires a heuristic approaches that is spread over a considerable time period.

However, in our implementation, the cost-benefit tradeoff is better than in a traditional cache system, because the savings in network call setup time is obtained in return for the lower cost of relatively cheap primary memory (DRAM) used in the CPU cache. This comes at a penalty in terms of decreased quality of the pre-computed route, as compared to routes computed on-demand. We believe that we cannot ignore the effect that pre-computation load has with respect to the CPU cycles consumed. Hence we study the effect of pre-computed computation cost on our route caching scheme.

We assume that the pre-computation of routes occurs on a dedicated co-processor, or via other independent mechanisms. These can create the routes without hindering the call processing activities of the switch. However, the route cache is not available for use while it is being updated. This is a pessimistic approach to modeling pre-computation cost, and as such tends to perform *worse* than any real implementation would.

However, by the same argument, if the simulations are not affected much by the changes in the scaling factor or pre-computation cost as defined by this model, then we can safely assume that a real implementation of the algorithm would be affected to a lesser degree by pre-computation costs. This implies that a real implementation should perform better than the simulation results, all other things being equal.

3.4.4 How should a route be selected from the cache ?

The incoming call connection request has a set of QoS requirements. We map the required bandwidth of the call to one of the 'N' quantization levels and determine if a route is stored for that equivalence class. If there is such a route, we have a *cache hit* and that route is selected and the DTL is embedded in the setup message to the next hop. If there is no route meeting the needs of the call, we have a *cache miss*.

In deciding which route cache entry to assign to the requesting call, we follow a *load balancing* policy. Thus, we choose that route cache entry that retains maximum remaining bandwidth. However, this is overridden by the route with minimum hops. Algorithm 3.2 presents how to choose a route in the caching policy that we follow.

3.4.5 What is an appropriate response to a cache miss ?

When we cannot find a route in the pre-computed cache, we decide to undertake on-demand routing for that call. The result of the on-demand routing is then inserted into the route cache so that subsequent calls can make use of the route.

Algorithm 3.2 Choosing a Pre-Computed Route

```
Condition: Receipt of connection request with
destination and QoS requirement.

if the cache contains path to destination
then
    for all candidate routes to the destination
    if candidate route effective bandwidth is
       less than or equal to request bandwidth
    then
        if there is more than one route with
           same effective bandwidth
        then
            return candidate route with minimum hop
        else
            return candidate route
else
    do on demand route computation
    store newly computed route
      with its effective bandwidth
      in route cache for that destination
```

3.4.6 When is a cache entry invalidated ?

When we receive a crankback element in the release message, we know that the call attempt did not succeed. We take this as an indicator that the cache entry is stale and that entry is deleted. This method is called the *crankback initiated invalidation* of route cache entries. It ensures that the routes that proved to be stale are not used in the future. This route cache policy can be turned off to study the effect it has on

call acceptance and call setup times. It should be noted that when a route entry to a particular destination is invalidated, the next time a call needs to be routed to that destination, on-demand routing is carried out and the result populated in the route cache. Thus there is a feedback effect involved: more crankback indications would induce more on-demand route computations. These on-demand computations use better quality topology state information, and hence tend to increase the accuracy of the routes that are computed. This in turn reduces the number of crankbacks. This mechanism is described in detail in Section 3.5.1.

3.4.7 What is the cache replacement policy ?

Whenever we are trying to replace an existing cache entry with a new one, we assume that the replacement is because of a cache update process. Hence we simply replace the entry with the new one. However, during replacement of cache entries, the route is disabled using the mechanism in the cache cost model of Section 3.4.3. Thus, depending on the scaling factor, the result of our replacement is not available until the route cache is re-enabled.

3.5 Route Cache Update Heuristics

Determining the need for updating the cache and the accompanying issue of when to recompute the routes is the problem that we are investigating. We consider the following heuristics for determining when to update the cache.

3.5.1 Crankback Indicator

When a crankback indication arrives for a path that was pre-computed, we know that the route is stale, and we delete the route. This allows us to use the freely available information regarding the failure of a route to discard the outdated route

entry, which caused the failure in the first place. The next time a route is chosen to the same destination, the working of the route cache is such that we will *not* find a valid route in the cache. Hence we are forced to undertake an on-demand computation of the route to the destination. It is expected that the new route will be computed using updated topology state information. The result of this computation is then reinserted back into the route cache. Thus a crankback indication followed by invalidation of the failed route is a means by which the cache gets updated with new routes which are based on fresher topology information.

If alternate routing is enabled, then there would be an on-demand route computation following the crankback indication. This would replace the invalidated cache entry immediately. However, alternate routing is not yet implemented in the MPG version of the KUPNNI simulator. When it is, performance should be no worse, and may even improve. This is a question for future research.

3.5.2 Timer based Method

In this heuristic, we set a timer that will periodically update the cache. This is a simple heuristic that takes its inspiration from timer based aging of topology state elements in the PNNI. It is based on the *ex cathedra* assumption that once the timer expires, the cache entries have aged sufficiently to force a complete refilling of the route cache. In this way, the changes that might have occurred in the topology database are reflected in the routes in the cache. The periodicity with which the cache is forced to update can be changed by changing the update time. The performance increase comes at the cost of increased route pre-computation overhead.

3.5.3 PTSE Count Method

We propose a new *ptse count* heuristic that counts the new topology state messages that arrive at a node and uses that as an indicator for when to recompute the routes. Any node will have an idea of how many PTSEs to expect after its initial

convergence. We define this number as the convergence limit. It is calculated by accounting for:

- One Nodal PTSE for each physical node
- Two Horizontal Link PTSEs for each physical link (one in either direction)
- One Nodal PTSE and one Nodal State PTSE for each Logical Group Node
- Two Horizontal Link PTSEs for each logical link (one in each direction)

When updated PTSEs arrive, the node keeps track of how many new PTSEs have arrived. There is a tunable PTSE Count heuristic parameter, that is specified for the experiment. We set it to be the convergence limit. When this update PTSE count reaches the PTSE Count heuristic parameter value, we initiate a route cache update. The use of the convergence limit as an indicator of significant change is arbitrary. We could experiment with other values for significant change, by changing the PTSE Count heuristic parameter value. This is a question for further research.

3.5.4 Combined Heuristics

We try mixing the three heuristics above, in order to be able to study the relative advantages and disadvantages of using them.

ptse-crankback: As the name suggests, we invalidate entries based on crankback indication, and use ptse count as a measure of flux to initiate route cache updates.

timer-crankback: Here, we combine the invalidation by crankback indication with time based route cache updates.

ptse-timer: In this heuristic, we disregard crankback indication, and update the route cache based on a periodic timer and the ptse count measure.

ptse-timer-crankback: Finally, we try a combined heuristic that invalidates based on crankbacks, and flushes the route cache based on a periodic timer and the ptse count.

3.6 Limitations of the model

Our model makes several important assumptions and has a few limitations. It is hoped that future development will overcome these limitations. However, it is also noted that our results are still valid, though the accuracy with which they depict real networks is challenged by these limitations. In addition the current limitations have been introduced conservatively, so improved models should look better, not worse.

Rechecking quality of the pre-computed route

We do not cross-verify the validity of the pre-computed route entries within the current topology database, as is carried out in other research studies [20]. Thus we are foregoing making a further check regarding the suitability of the route cache entry. However, this is intentional and accounted for in the fact that we are trying to shadow the working of the on-demand algorithm as far as possible. Since the standard on-demand routing function does not undertake this re-checking, we choose to do the same when evaluating the performance improvement due to caching.

Error due to quantization levels

As with any estimate of call bandwidth requests, we divide the available bandwidth into quantized levels for pre-computing multiple routes to the destination nodes. This introduces error since our choice of route tends to have more available

bandwidth. This is because in our route cache policy, we choose the path that can support equal or greater bandwidth compared to the call requirements.

However, this can in turn be an attractive side effective of our caching algorithm. By using equivalence classes of bandwidth, we account for any minor fluctuations in call load. Further, our load balancing scheme helps spread out the utilization of bandwidth in the network, creating fewer bottleneck links.

Crankback Retries

We set the number of crankback retries to be 0. This prevents alternate routing from occurring in our model. We do this primarily because alternate routing functionality is not currently supported in the KU-PNNI simulator for Multiple Peer Groups. It should be noted that, with alternate routing, there can be significant differences in the call acceptance ratio [21]. However, this is an area of future work once the functionality is added to the simulator.

Cost of cache updates

We do not account for how cache updates would affect call processing if there is no dedicated processor doing the pre-computation. In order to do so, we could use the following method:

- First, we measure the real execution time it takes to undertake filling the route cache.
- We determine the equivalent *idle cycles* consumed by the pre-computation. This is done by measuring the virtual time elapsed before another call connection request arrives.
- We can now determine how many *extra cycles* the pre-computation process consumed. This is over the CPU time it would have been able to use if it had not been pre-empted by the call connection request coming in.

- Using the condition that pre-computation would occur only when the routing algorithm is not pre-empted by a call connection request, we could invalidate as many cache entries as were filled during the *extra cycles*.

However, our literature survey shows that most research into route caching has ignored the full cost of route cache computation. The study by Apostolopoulos et al. [2] is an exception. Their model is at a much higher degree of complexity, and has the ability to measure the cost of several elements of the routing process. This is an area for future work in our simulator.

Chapter 4

Evaluation

"Contrariwise", said Tweedledee, "if it was so, it might be;
and if it were so, it would be, but as it isn't, it 'ain't.
That's logic."

This chapter presents our experimental methodology, the factors influencing our evaluation, and the choices we made regarding these factors.

4.1 Verification

To begin with we prove that our model verifies results obtained in other research studies regarding the benefits of pre-computed route caching [2, 20]. This sets the stage for validating the new features that are introduced in our solution. We undertake the following studies:

- In a single peer group, as the network scales, we show that pre-computed routing, when compared to on-demand routing, is beneficial in reducing the call setup times without paying too high a price in terms of increased call rejection.
- In a multiple peer group, we show how changing the peer group size, and the cache size, affects the performance of pre-computed route caching. This

is aimed towards finding optimum values for peer group size and cache size which are then used in the subsequent experiments.

4.2 Performance of Multiple Peer Group PNNI

On the basis of the verification experiments, we look at PNNI networks that are based on Multiple Peer Group hierarchies. We consider how changing parameters of the caching policy as well as the grouping of the network affects performance. We also consider the effect of cache pre-computation cost based on the cost model described in Section 3.4.3.

4.3 Validation

Finally, we look at the issue of route cache updates, and validate the following hypotheses:

- timed pre-computed route updates that are synchronized with the process of re-aggregation show improved performance over any chosen cache update policy.
- the *ptse count* based heuristic that we propose is a valid method for determining when to recompute the cache routes. When used in conjunction with *crankback initiated invalidation*, it gives comparable performance to on-demand route computation.

This represents the validation of our solution, based on the two performance metrics of average call setup time, and average call acceptance ratio. The intent is to show that using the *ptse count - crankback invalidation* heuristic, we can achieve the best combination of lowering average call setup time while increasing the average call acceptance ratio.

4.4 Evaluation Setup

Our evaluation is based on the KUPNNI Simulator, which implements ATM Forum UNI 3.0 and the PNNI specification version 1.0 [13]. These specifcations define the ATM User Network Interface and Private Network-to-Network Interface. The simulator is an event driven system, that models network nodes, links and traffic sources using 98% code shared with *real off board signalling code* [3]. It supports a virtual time model of the experimental space, and uses it to model link level delays, routing latency, etc. The call traffic model supports several arrival, duration, and destination selection distribution. The topology, network parameters, traffic load, and event schedule are specified using an input language, the experiment is run and the results are printed out.

Now we describe the simulation environment, the performance metrics and parameters that affect these metrics of interest.

4.5 Simulation Environment

There are several factors that determine the results of a network experiment. Hence while evaluating the performance of a routing algorithm, we need to be careful in our choice of the following:

4.5.1 Topology

In general we consider large ATM backbone networks, typically characterized by an Edge-Core topology, commonly used to set up private ATM networks. This is created using a topology that has been used in similar studies [21].

Our general Edge Core network has 36 edge nodes and 24 core nodes. The edge nodes are *dual homed* to two core nodes. The core nodes themselves are categorized into large capacity and small capacity nodes to include multi-vendor vari-

ations in the switch performance. The large capacity nodes are connected to other high capacity nodes with higher bandwidth links, and the small capacity nodes are interconnected with lower bandwidth links [21]. For experiments that require scaling, we use another version of the Edge-Core network, and retain the connectivity-per-node while increasing the size of the network.

For Multiple Peer Group experiments, the motivation is to extend the Edge-Core format when grouping the physical network into Peer Groups. Hence, we partition the topology into edge peer groups and core peer groups that connect the edge peer groups. Further, in keeping with the dual homing scheme, each small capacity core switch is connected to two large capacity switches in the core peer group.

As a check on our new heuristic for determining when to update the route cache, we test our solution on the following variations of network topology:

- 3 level Multiple Peer Group network.
- Cluster network containing 8 nodes per cluster, and 5 such clusters.
- 60 node network divided into 5 peer groups (non edge core).

The topologies of primary interest are sketched in Figures 4.1 and 4.2.

4.5.2 Traffic Sets

In order to demonstrate the benefits of QoS routing, we create increased loads on parts of the network. This is done by modeling the traffic as follows.

Some of the experiments involve uniform traffic where a node can request connections to any of the other nodes with uniformly distributed probability. When uniform traffic is used, the mean request arrival rate (λ_U) is the same for all nodes in the network. This is termed Type U. We also create conditions of non-uniform traffic by establishing two sets of non-uniform traffic nodes, Type X and Type Y,

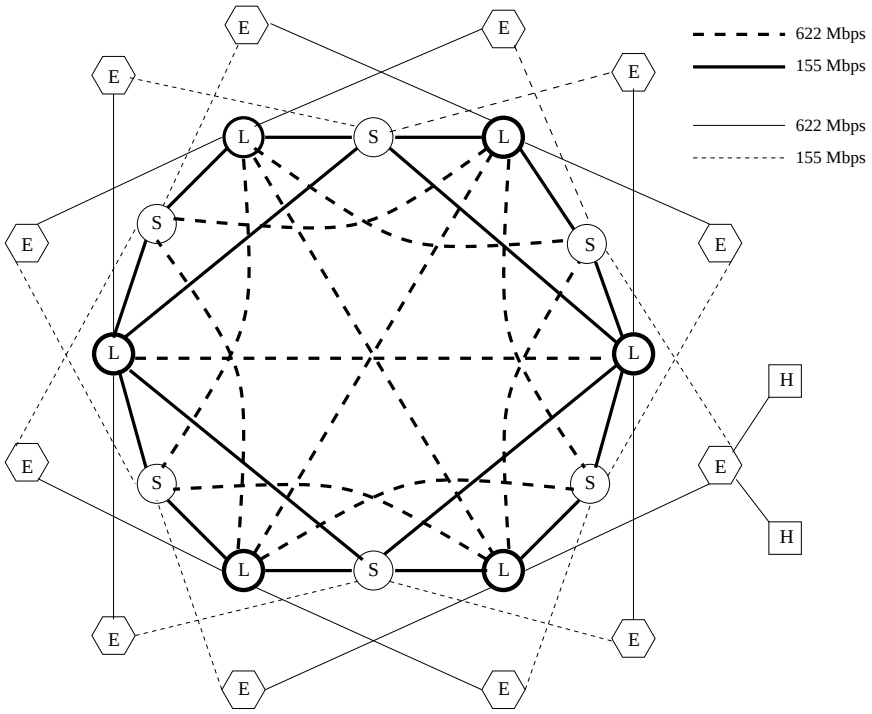

Figure 4.1: Flat Edge Core Topology

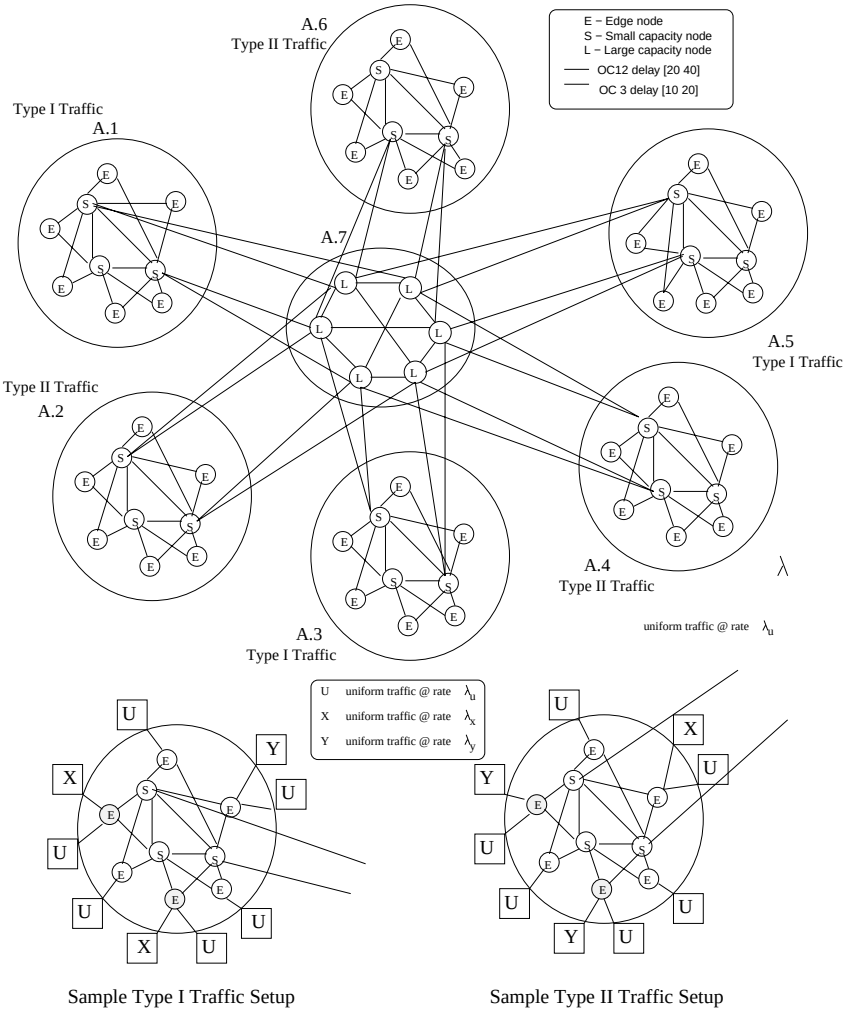

Figure 4.2: MPG Edge Core Topology with Traffic Sets

with mean request arrival rates λ_X and λ_Y respectively. Table 4.1 lists these properties.

Node Type	Destination Choice	Arrival Rate
Type U	*uniform* among 35 other Type U nodes	*Poisson* [λ_U]
Type X	*uniform* among 9 other Type X nodes	*Poisson* [λ_X]
Type Y	*uniform* among 9 other Type Y nodes	*Poisson* [λ_Y]

Table 4.1: Traffic Sets

Nodes of one type can request connections only with nodes of the same type. The mean request arrival rate can be specified for each type. The peer groups have different number of Type U, Type X and Type Y nodes and thus form two distinct categories, named Type I Traffic and Type II Traffic. The peer groups are appropriately marked with the traffic types in Figure 4.2.

In order to vary the load on the network, we increase the arrival rate of the calls at the host / UNI interface, which are assumed to follow a Poisson distribution. In experiments that require fixed arrival rates, the average traffic loads are chosen such that call acceptance ratios are kept in the 70% -80% range.

4.5.3 Traffic Load Characteristics

A basic dimension of comparison between the different routing architectures is their behavior under different call loads. To capture the effects of call duration and requested bandwidth each UNI entity in our network requests a mixture of calls that correspond to the following applications:

- Low Load Calls: characterized by low bandwidth (64Kbps to 1.5Mbps), and last for shorter mean durations of 30 seconds.
- Medium Load Calls: characterized by medium bandwidth requirements that

fluctuate between 1.5Mbps to 3.6 Mbps and last for medium mean durations 45 seconds.

- High Load Calls: need constant high bit rate, from 5Mbps to 15Mbps, and last for comparatively longer mean periods 60 seconds.

Bandwidth requirements are uniformly distributed between the minimum and maximum value shown while call duration is exponentially distributed with the mean shown. This is shown, along with the fraction of calls for each application in Table 4.2.

Application	Bandwidth *uniform*	Call Rate *exponential*	Fraction of Calls
Low	64Kbps - 1.5Mbps	$1/\mu = 1/30s$	30%
Medium	1.5Mbps - 3.6Mbps	$1/\mu = 1/45s$	30%
High	5Mbps - 15Mbps	$1/\mu = 1/60s$	40%

Table 4.2: Traffic Parameters

4.5.4 PNNI Parameters

In order to reduce the effects due to variations in the PNNI parameters, we use a standard switch configuration which uses the following PNNI specific parameters:

- Aggregation Policy: This represents the scheme by which resource information within a peer group is aggregated. We adopt a symmetric star aggregation policy for all our experiments. This policy is simple and scales well [12]. It does not adversely skew comparison results between on-demand and pre-computed routing techniques.

- Routing Policy: We study the performance of the pre-computed route caching by using the WIDEST_MINIMUM_HOP routing policy. This runs a routing algorithm that generates a widest_minhop path. This is defined as [19]:

> ... a path with the minimum hop count among all feasible paths. If there are several such paths, the one with the maximum reservable bandwidth is selected. If there are several such paths with the same bandwidth, one is randomly selected.

It is shown that this routing algorithm gives the best overall performance, as compared to other multiple criteria algorithms [21].

- Significant Threshold: The parameters for determining significant change for flooding are set to :
  - Proportionality Constant: 25
  - Flooding Threshold: 5
- Crankback Retries: It is well known that excessive alternate routing can actually reduce routing performance in conditions of high load, since traffic following alternate routes can interfere with minimum hop traffic competing for the same links [15, 9]. Hence, we choose to set crankback retries to 0. This way, the nodes in our network are informed of crankback events. However, they do not do alternate routing after getting crankback.

4.5.5 System Parameters

Ours is a discrete event simulator. It generates random numbers from a single seed. Thus, if we use the same seed, we should be able to reproduce all the events in repeated experiments. The use of our virtual time model in simulating link delays, routing latency, and other processing delays ensures that multiple runs of an experiment provide strongly similar results.

We do the following to ensure fairness in our simulation studies:

- Change the value of the random number seed: We calculate averages of our performance metrics for each experiment, by changing the random number

seed.

- Use a warm up time to allow the system to stabilize: We ignore the first 10% of all calls when computing the results of any experimental run. This accounts for the traffic to warm up the network. Such calls are not included when determining the routing performance [16].

4.6 Performance Measures

We now describe the performance metrics and the parameters that affect them. These are identified as the properties of interest in our experiments.

4.6.1 Call Acceptance

In most circuit switched routing performance studies, the connection success ratio is used as a measure of routing performance. The connection success ratio is defined as the percentage of connection requests accepted out of the total number of requests. A parallel measure of call performance is the bandwidth acceptance ratio. This is defined as the ratio of the sum of bandwidths of connection requests accepted over the sum of bandwidths of all the connection requests. We gather call statistics that allow us to calculate both these metrics.

4.6.2 Time Measures

All time measurements within our simulation are based on *virtual time* duration, measured within the virtual time model. This represents the time line for events within the experiment. The timing metrics of concern are:

- Setup time: For call connection requests, this is defined as the time between the host / UNI sending out a connection setup request until the time the connection established message returns. Compared to *routing time*, which is the

time required for undertaking a route computation, *setup time* is a more comprehensive measure, since :

1. Setup time includes routing time within it.
2. Setup time incorporates the link delays resulting from traversing links, which are chosen by the routing function.
3. Setup times are the more realistic estimates of how much delay a real user of the network would face.

- Refresh Periods: We conduct experiments that vary the refresh periods for route cache updates and compare it with re_aggregation intervals. For these purposes, the time measure is expressed in virtual seconds and simulated as a timer within the virtual time model.

4.6.3 Counting Failure Cause

This is a way of defining if the call was rejected during route computation, within the first peer group (source peer group), or in a foreign peer group. This helps to determine if the reason for failure was because of inaccuracy in local updates within a peer group, or inaccuracy of aggregated information from other peer groups.

A call can be rejected due to either of two reasons:

- it is rejected because a feasible path with sufficient resources cannot be found by a routing algorithm; or
- the call is refused in an intermediate node. This is because during the call connection period, the resource availability on the selected path has changed since the time when it was used for making the routing decision. Alternately, the update delays causes routing to be out of date.

With on-demand routing, either case is possible. However, in the case of pre-computed paths, if we have a cache hit, we will fail only due to the second

reason, since there is no routing involved when the call request arrives. However, if there is a cache miss and we go ahead and try on-demand routing, we can fail due to the first reason.

In our experimentation, we count each type of call failure with a view to determining the location of inaccurate information.

4.6.4 Cache Hit/Miss Ratio

We measure the accuracy of our caching scheme by counting the number of times that a candidate path is successfully found or missed, in the route cache. Expressing the ratio of this number with the total number of routing requests gives us the cache hit / miss ratio.

4.6.5 Cache Update Cost

We measure the cost of doing cache updates. This is done by measuring the time it takes to fill a route cache in real time. Further, we also measure the size of the route cache, so as to compare the performance of different route cache update heuristics.

Chapter 5

Experimental Results

”There’s more evidence to come yet,
please your Majesty,” said the White Rabbit,
jumping up in a great hurry;
”this paper has just been picked up.”

This chapter describes the experiments that were carried out, the results that were obtained, and the explanation for the behavior of the system on the basis of the metrics of primary interest, which are:

- Average Setup Time
- Average Bandwidth Acceptance Ratio
- Average Call Acceptance Ratio

The experiments can be classified as: verification, validation, and auxiliary experiments. These are described in the following sections. We also describe the statistical analysis conducted on the data obtained from the experiments.

Verification Experiments

Here, we aim to corroborate assertions made by other researchers, which indicate lower average call setup times for a pre-computed route caching strategy compared to an on-demand strategy [4, 20]. This improvement in our performance metric occurs at the expense of decreased call acceptance.

Validation Experiments

We study the problem of determining when to update the pre-computed route cache. We apply different cache update heuristics and evaluate our hypothesis that:

> Updating the route cache based on a *ptse count* heuristic with *crankback based invalidation* gives better overall performance, when compared to the other heuristics.

Auxiliary Experiments

We look at the effect of changing different parameters of the route cache, such as pre-computation cost and cache quantization levels, on call performance in multiple peer group topologies. We also look at how changing peer group size affects the performance of pre-computed routing.

5.1 Data Analysis

We conduct the following analysis on data obtained from the experiments. Multiple runs of each experiment were made, using different values for the random

number seed, to determine the average values of the performance metrics. The mean value of the samples thus obtained was used for the data analysis.

First, the integrity of the data itself was tested to see if the values obtained were close enough to the expected values. This was done by building an interval estimate for the data point obtained using a confidence level of 95 % . For obtaining the interval estimates, we use the t-distribution which is required whenever the sample size is 30 or less and the population standard deviation is not known, as is true in our case. The formula for estimating standard error is:

$\hat{\sigma}_{\bar{x}} = \frac{\hat{\sigma}}{\sqrt{n}}$

where $\hat{\sigma}$ is the standard deviation of the sample, and n is the sample size. Having obtained the standard error, we obtain the interval estimates by the equations:

$\bar{x}_{high} = \bar{x}_{mean} + t * \hat{\sigma}_{\bar{x}}$

$\bar{x}_{low} = \bar{x}_{mean} - t * \hat{\sigma}_{\bar{x}}$

where $\bar{x}_{mean}$ is the average value over all the data samples, t is obtained from standard t-distribution tables for the number of degrees of freedom and the confidence level that is required (95 % in this case).
To quote from Levin and Rubin [18]:

> The probability that we associate with an interval estimate is called the confidence level. This probability indicates how confident we are that the interval estimate will include the population mean. The confidence interval is the range of the estimate we are making.

If we report that we are 95 % confident that the mean of the setup times for a certain topology will lie between 174 and 183 ms, then the range 174 - 183 is our confidence interval. We will, however, express the confidence interval as a standard error of 1.5927 where 183.3783 and 174.5356 are the upper and lower limits of the confidence interval, for a t value of 2.776.

The mean values obtained for on-demand and pre-computation runs are

tested for statistically significant differences between them. This is done using the t-test for hypothesis testing of means (small sample size) [18].

5.2 Effect of Topology Scaling

These experiments are carried out on a Single Peer Group network. The topology follows a basic Edge-Core design, and scaling increases the number of nodes and links. The average number of connections per node is held constant.

5.2.1 Establishing the Value of Route Caching

Performance Metric

The performance metric used for this experiment is average call setup time for call connection requests. We also computed the percentage increase in call setup times for on-demand routing as compared to the pre-computed routing strategy.

Performance Parameters

The number of nodes in the network is increased from 24 to 96. This represents the scaling in the network topology.

Experimental Design

We subject the experimental network to a light load and consider both on-demand computation and pre-computed routing strategies. There are no updates to the route cache, and a single quantization level is used to pre-compute the cache entries. This is a pessimistic choice of cache parameters. We believe that the pre-computed routing strategy will perform better when there are route cache updates, and a greater number of quantization levels. Choosing a pessimistic set of cache

parameters makes it likely that the performance of a real implementation would be much better.

Hypothesis

As the size of the network increases, we expect that the average call setup time will also increase, given a uniform choice of destinations in the call load. However, the average call setup time for the pre-computed strategy would be less than that of an on-demand strategy for a given network size.

Results

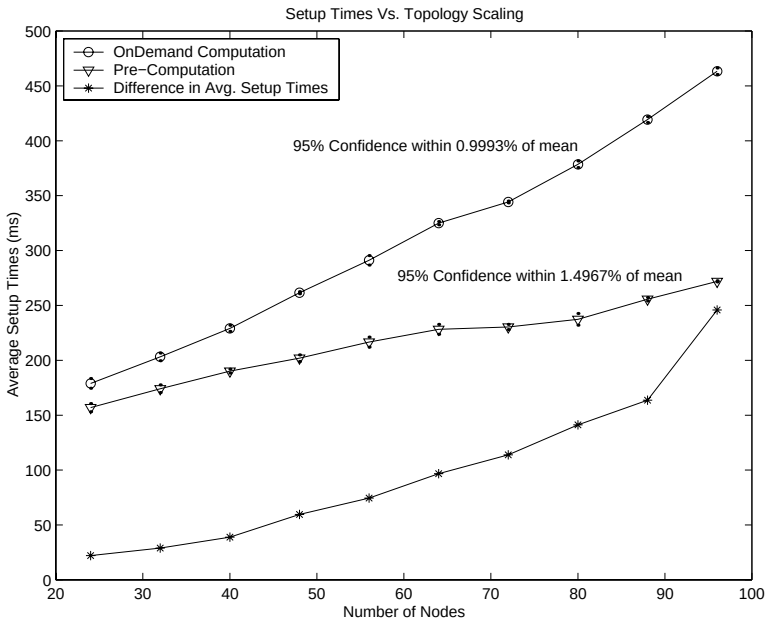

Figure 5.1: Effect of Topology Scaling on Average Call Setup Times

As seen in Figure 5.1, the results support our hypothesis since:

- as the size of the network increases, average call setup times also increase, for both routing strategies.

- pre-computed routing shows lower average call setup time, as compared to the on-demand routing strategy.

Further, it is observed that the difference in average call setup times for on-demand and pre-computed routing follows an exponential curve, that fits the distribution $y = n * e^{mx}$, where $n = 1.8593$ and $m = 0.2573$, with a correlation coefficient of $R^2 = 0.9866$.

Node	24	32	40	48	56	64	72	80	88	96
Run1	184.639	206.086	232.884	262.640	290.479	322.914	344.374	381.029	420.682	462.125
Run2	176.233	198.784	228.342	261.646	289.506	325.691	344.714	377.728	419.189	466.908
Run3	178.939	203.077	229.068	261.170	289.063	325.081	344.907	381.186	420.562	463.130
Run4	179.32	202.54	227.141	260.6	297.17	326.044	343.325	377.001	420.519	463.998
Run5	175.655	205.087	228.149	261.893	289.263	324.937	343.722	375.855	415.647	460.541
Average	178.957	203.115	229.117	261.59	291.096	324.933	344.209	378.56	419.32	463.34
Standard deviation	3.561	2.82	2.216	0.767	3.438	1.215	0.669	2.42	2.141	2.373
Variance	12.684	7.951	4.909	0.588	11.823	1.477	0.447	5.857	4.585	5.63
Stdandard Error	1.593	1.261	0.991	0.343	1.538	0.543	0.299	1.082	0.958	1.061
Upper Range	183.378	206.615	231.868	262.542	295.365	326.442	345.038	381.564	421.978	466.286
Lower Range	174.536	199.614	226.367	260.638	286.827	323.425	343.378	375.555	416.661	460.395
% of mean	2.471	1.723	1.201	0.3641	1.466	0.464	0.241	0.794	0.634	0.636

Table 5.1: Effect of Topology Scaling on Average Call Setup Times (On-Demand Routing)

Table 5.1 shows how analysis is carried out on average call setup time data for an increasing number of nodes using the on-demand routing strategy. The five runs of the experiment are conducted. The average, standard deviation and standard error is computed. Using the Student t-distribution for 4 degrees of freedom, the 95 % confidence interval has a value for t of 2.776. This is used in calculating the Upper and Lower range of values. We are then able to compute what is the deviation in the mean as a percentage of the population mean. It is seen that for the 95 % confidence interval, the data values lie on average within 0.9993 % (rounded off to 4 decimal places) of the population mean.

Similar data analysis was carried out on pre-computed routing time data. The results are summarized in Table 5.2. Here, it is seen that the data values lie on average within 1.4967 % (rounded off to four decimal places) of the population mean for the 95 % confidence interval.

Node	24	32	40	48	56	64	72	80	88	96
Upper Range	160.599	177.665	191.938	204.967	221.183	232.726	232.694	242.689	257.264	272.434
Lower Range	153.204	170.602	188.387	198.956	212.022	223.767	228.004	232.079	253.995	271.395
% of mean	2.356	2.028	0.934	1.488	2.115	1.963	1.018	2.236	0.639	0.191

Table 5.2: Effect of Topology Scaling on Average Call Setup Times (Pre-Computed Routing)

The improvement in average call setup times between pre-computed routing and on-demand routing reaches **53%** when the number of nodes is **96**. Figure 5.2 shows the percentage increase in the average call setup times when a transition is made from pre-computation to on-demand computation.

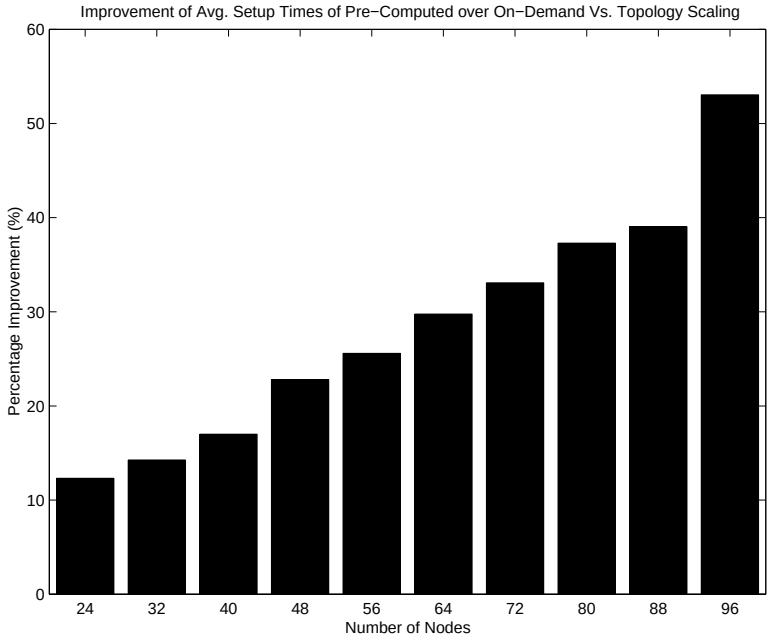

Figure 5.2: Percentage Improvement in Avg. Setup Times (Pre-Computed over On-Demand) with Topology Scaling

5.2.2 Variation of Bandwidth Acceptance Ratio with Load

Performance Metrics

The bandwidth acceptance ratio is used to measure call success. This metric incorporates the fact that the main reason for call rejection within a network is lack of bandwidth. This is especially true for highly stressed or loaded networks. We also gather data about the call acceptance ratio for this experiment.

Performance Parameters

In this experiment, the average call load is increased in stages. This is done by decreasing the mean inter-arrival time of the call traffic from 5 seconds to 0.5 seconds. This corresponds to a call arrival of 0.5 calls per second to 2 calls per second.

Experimental Design

We fix the size of the edge-core network topology at 64 nodes, and by providing a high traffic load, ensure there is appreciable change in the network resources over relatively short periods of time. We then investigate the change in call acceptance rate when network size is increased. This is done for on-demand and pre-computed routing strategies.

Hypothesis

As the load is increased, we expect the average bandwidth acceptance ratio to decrease irrespective of the routing policy. However, for the pre-computed routing strategy, there is a more pronounced effect of rapidly changing network state on the quality of route cache entries which have been computed at the beginning of the experiment. Hence we expect the average bandwidth acceptance ratio for pre-computed routing to decrease faster with increasing load, than for on-demand routing.

Results

Figure 5.3, shows the variation of call bandwidth acceptance ratio with call load for both on-demand and pre-computation strategies. It is seen that as the call load increases, the bandwidth acceptance ratio decreases in both cases. We find that for a **64** node network, there is a statistically significant change in the average bandwidth acceptance ratio between on-demand computation and pre-computed routing for a call arrival rate of **1.0** calls/seconds or more. At lower call arrival rates, the average bandwidth acceptance is not statistically significant.

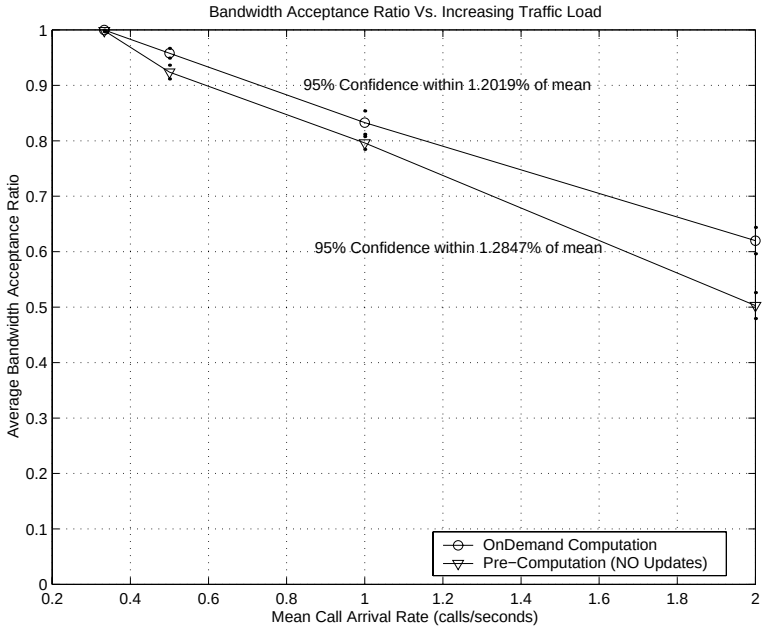

Figure 5.3: Average Bandwidth Acceptance Ratio with Increasing Load

The average bandwidth acceptance for pre-computed routing decreases at a faster rate than for on-demand routing. This is because as the network size increases, network flux affects pre-computed routes much more than routes which are computed by on-demand computation.

We also see in Figure 5.4, that the call acceptance ratio follows almost exactly the same curve as the bandwidth acceptance ratio (both in value and trend).

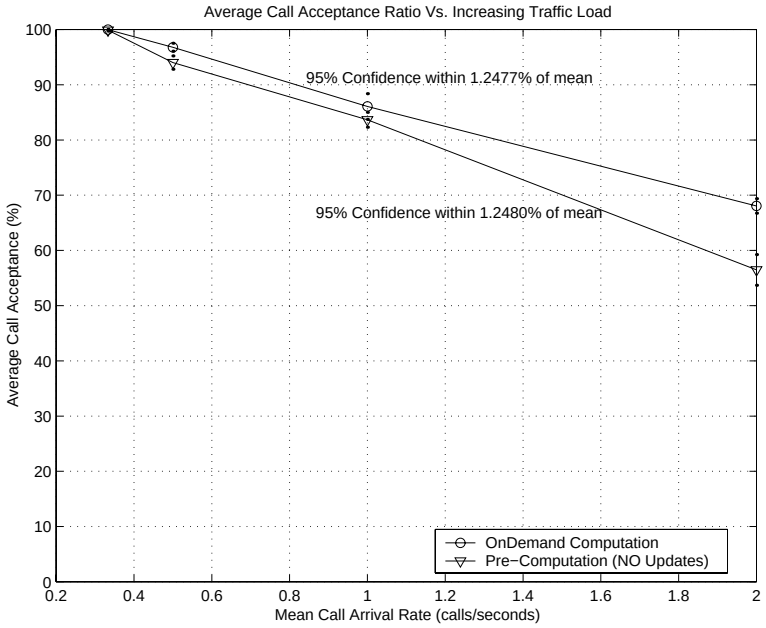

Figure 5.4: Average Call Acceptance Ratio with Increasing Load

Statistical analysis of average bandwidth acceptance and call acceptance ratio data values was carried out, in a manner similar to that described in Section 5.2. It is seen that the data values lie on average within 1.25 % of the population mean for the 95 % confidence interval. We do not repeat the listing of data values obtained and the intermediate results that were calculated.

5.2.3 Conclusion

The results of the verification experiments described in Section 5.2.1, show that the use of a pre-computed routing strategy significantly decreases average call setup times as the topology size increases. Section 5.2.2 shows the tradeoff experienced with using pre-computed routing. We see that the decreased average call setup time comes at the penalty of a decreased average call acceptance ratio. These results indicate trends and trade-offs consistent with those observed by other researchers [2, 4, 20].

5.3 Multiple Peer Group Experiments

We now consider a multiple peer group hierarchical topology, and study the effect of peer group size, pre-computation bandwidth quantization levels, and pre-computation cache update costs on the call performance metrics of interest.

5.3.1 Effect of Changing Number of Peer Groups

Performance Metrics

Average bandwidth acceptance ratio and average call setup time are the primary metrics in this experiment. The secondary metrics are the location of failure and topology database size.

Performance Parameters

We increase the number of peer groups in a network having a fixed total number of nodes. This implies decreasing the number of nodes per peer group. In this manner we can study the effect of changing multiple peer group topology on call setup time and bandwith acceptance for on-demand and pre-computed routing strategies.

Experimental Design

We establish a standard edge-core topology and a standard traffic mix. Next we vary the peer group size, and hence the number of peer groups. Experimentation is done using both on-demand and pre-computed routing strategies.

It is significant to note that the pre-computed route cache is *not* updated in these experiments. This is a pessimistic choice, and we thus tend to experiment with route cache contents that are worse than will be conceivable when we introduce a cache update method in experiments described later in this section.

Hypothesis

Increasing the number of peer groups for a fixed total number of nodes leads to increased aggregation for a larger portion of the network. This results in decreased accuracy of information that is flooded across peer groups, and less accurate information to use to compute the routes. This would indicate a decrease in the average bandwidth acceptance ratio with increasing number of peer groups. Since aggregated information ages fastest, it is expected that there are more failures in foreign peer groups than within source peer group.

Moreover, as the number of peer groups increases, the average number of hops in a hierarchically complete route decreases. This should indicate a decrease in the routing complexity and hence the average call setup times as the number of peer groups increases.

Primary Results

Figure 5.5 shows that increasing the number of peer groups decreases the average call setup time for both on-demand and pre-computed routing strategies. However, for any given peer group size, pre-computed routing shows lower average call setup times as compared to on-demand computation. The improvement in call setup time of pre-computed routing as compared to on-demand computation, is consistent with earlier, single peer group experiments. For all the peer group sizes that were studied, the improvement is in excess of 12 % .

Figure 5.6, shows that average bandwidth acceptance ratio first decreases with increasing number of peer groups, and then starts to increase.

The shape of the curve can be interpreted by looking at the plot of average database size as a function of increasing number of peer groups. This is shown in Figure 5.7. The database is the same for both on-demand and pre-computed routing schemes, since it is a function of the aggregation policy. It is seen that the database size decreases with increasing number of peer groups and then starts to

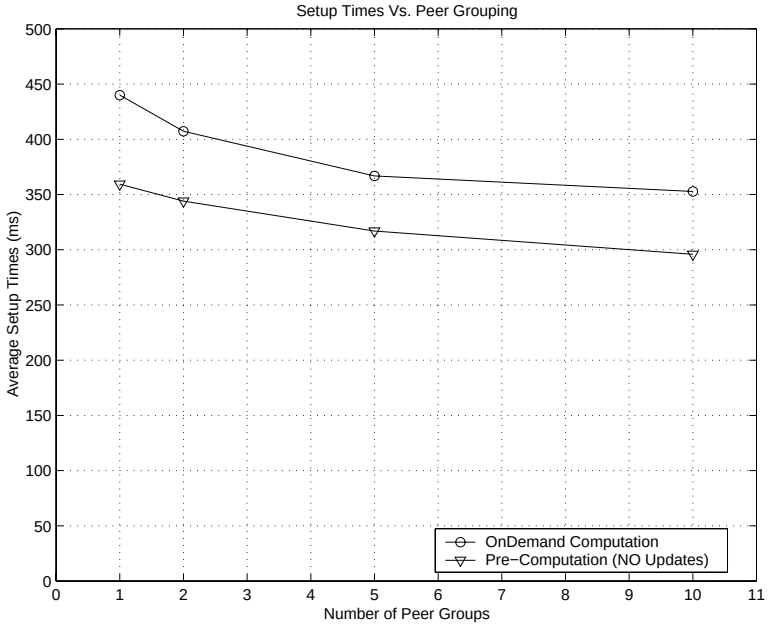

Figure 5.5: Average Setup Time with Peer Group Size (no cache updates)

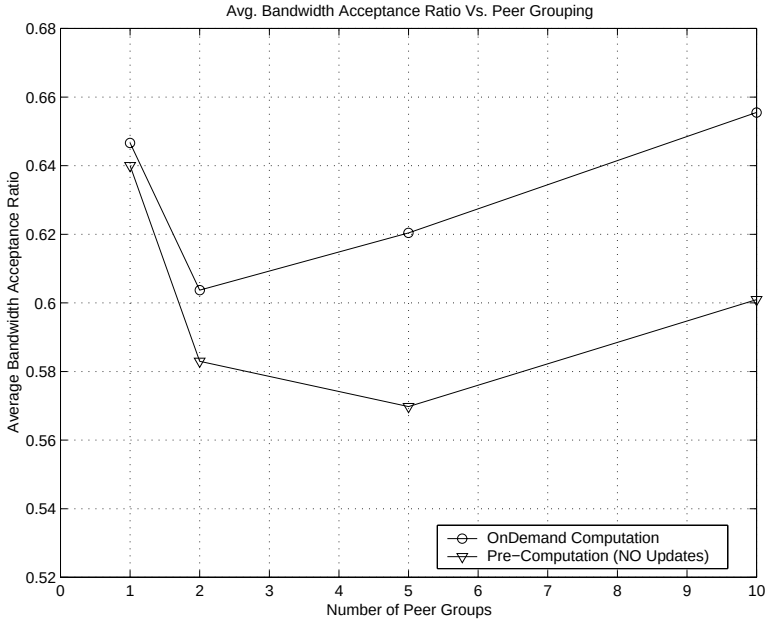

Figure 5.6: Average Bandwidth Acceptance with Peer Group Size (no cache updates)

increase. This occurs when the number of peer groups (10 peer groups) is greater than the number of nodes in each peer group (4 nodes). This change in database size is because of the change in the amount of information that gets flooded within the network.

In our experiment, the size of the database is an indication of the quality of information available to make routing decisions. Since we use the same aggregation policy in all the experiments, a decreasing database size with increasing number of peer groups indicates a decrease in the number of aggregated PTSEs. This decreases the quality of information available to compute routes. But when the number of peer groups exceeds the number of nodes within any peer group, the components of the database that correspond to aggregated information exceeds the locally flooded information. Hence the database size shows an increase.

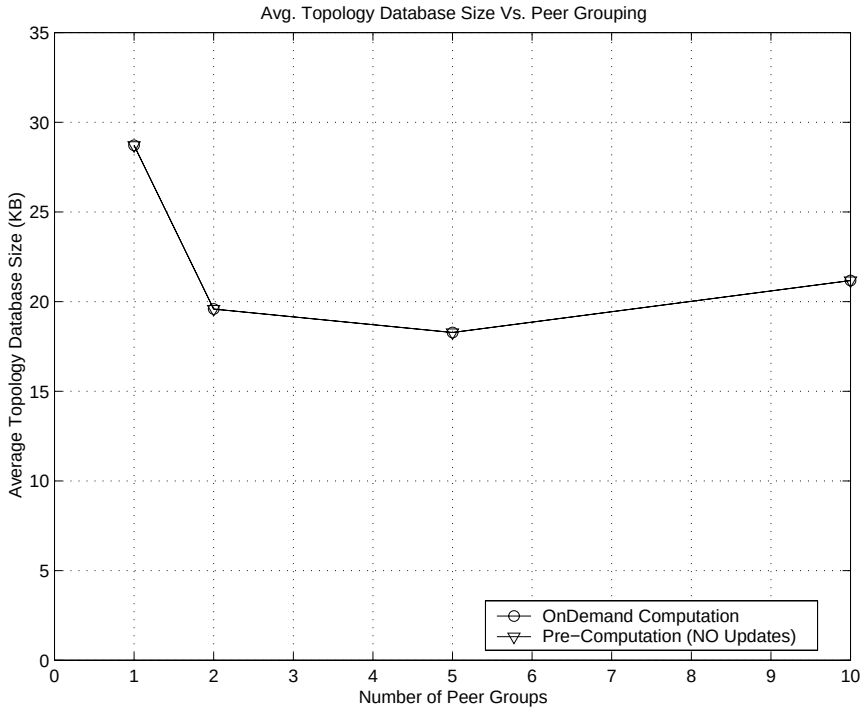

Figure 5.7: Average Database Size with Peer Group Size

Now we can see why the average bandwidth acceptance ratio follows a

curve similar to the change in database size. As the amount of information available for route computation varies, the accuracy of the routes that are computed also varies in a similar manner.

Figure 5.6 also slows that the average bandwidth acceptance ratio of pre-computed routes is lower than that for on-demand computed routes. This is in accordance with the tradeoff experienced with pre-computed route caching in that the decreased average call setup time is obtained at the cost of decreased bandwidth acceptance ratio.

We find that the call acceptance ratio follows almost exactly the same curve as the bandwidth acceptance ratio (both in value and trend). Hence we do not present the essentially identical plot for average call acceptance ratio.

We find as part of our results, that the size of the route cache lies in the moderate range of 500 bytes to 8460 bytes, in our implementation.

Secondary Results

Next we look at the secondary metric of interest: location of call failures. Figure 5.8 plots the case for on-demand computation. We observed that as the number of peer groups increases:

- for on-demand routing, the number of calls failing at the source node, owing to to its inability to obtain a route to the destination, decreases;
- of the number of calls that do get routed out of the source node, those that fail in the source peer group remain essentially the same; and
- the calls that fail in foreign peer groups increases.

These observations can be explained in the following manner. As the number of peer groups increases, the information available to the source node tends to indicates that there are resources in the foreign peer group. Thus the source node

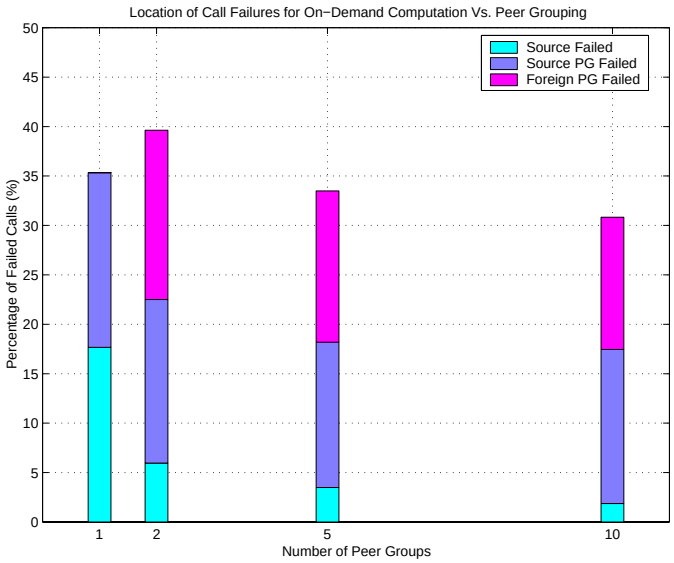

Figure 5.8: Location of Call Failures (On-Demand Routing) with Peer Group Size

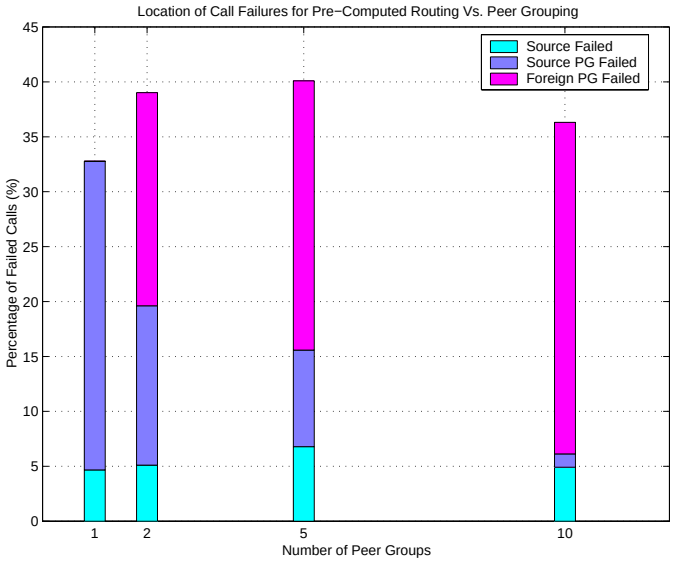

Figure 5.9: Location of Call Failures (Pre-Computed Routing without cache updates) with Peer Group Size

admits the call connection with increased probability as the number of peer groups increases. However these guesses need not be correct. The quality of these routes is determined by where they actually fail in the network.

As observed, the routes that are generated by the source node using on-demand computation fail uniformly within the source peer group as the number of peer groups increases. This is because the source peer group gets flooded with updated information about the availability of its resources.

In the case of foreign peer groups, the on-demand route computed by the source node is based on lower quality information. The quality of this information decreases with increasing number of peer groups. Hence the number of failures in foreign peer groups increases.

Figure 5.9 plots the number of failures total and by location versus peer group size when using a pre-computed routing strategy (without route cache updates). It is seen that:

- for pre-computed routing, the number of calls failing at the source node owing to its inability to obtain a route to the destination varies only slightly;
- of the number of calls that do get routed out of the source node, those that fail in the source peer group decreases as the number of peer groups increases; and
- finally, the calls that fail in foreign peer groups increase as the number of peer groups increases.

These observations can be explained by remembering that pre-computation strategy, without cache updates, builds the initial route cache and routes all subsequent calls based on this initial route cache. The variation in peer group size only affects the route cache inasmuch as it changes the number of cache entries in it. The source node finds routes in the cache with equal probability and as the number of peer groups increases, the source failed calls remain essentially constant.

The decrease in the number of calls failing in the source peer group for pre-computed routing is explained as follows. As the number of peer groups increases, the degree of network flux within the source peer group decreases. Initially computed route entries in the cache remain valid for a longer period of time. Hence the failures based on initially computed routes in the cache decrease with an increasing number of peer groups.

Finally, since routes to foreign peer groups are pre-computed on aggregated information, as the experiment progresses the foreign peer group information ages at a faster rate and contains less information to begin with. This accounts for the increase in the number of calls that fail in foreign peer groups, when the number of peer groups increases.

5.3.2 Effect of Changing Quantization Levels

Performance Metrics

We measure the average bandwidth acceptance ratio for this experiment. We also measure the average call setup times.

Performance Parameters

Our scheme of pre-computation divides the available bandwidth into ‘N’ different equivalence classes, where ‘N’ is a tunable parameter. These represent the quantization levels for which routes are pre-computed. It is expected that the incoming call requests can be mapped into these levels, and an appropriate route can be extracted. This is described in greater detail in Section 3.4.1. We let the number of quantization levels (’N’) vary from 1 to 3.

Experimental Design

For a standard edge-core topology, subjected to a standard traffic mix, we vary the quantization level and measure the average call acceptance ratio and the average call setup times. The use of a standard topology provides a common network to test, while the use of a standard traffic mix eliminates the effect of variance in load on the call rejection.

Hypothesis

When the number of quantization levels increases, we have a better set of possible routes to choose from. A greater number of quantization levels indicates more bandwidth equivalence classes, for each of which there is a possible route in the cache. Hence we would get routes that are at a finer granularity, with respect to the bandwidth that they can accommodate. This provides a way of more accurately selecting the route that best meets the call requested bandwidth requirement. Ideally the trend we should expect is a decrease in blocking rate as the number of quantization levels increases.

Results

As seen in Figure 5.10, the average bandwidth acceptance ratio remains essentially the same, as the number of quantization levels increases.

The absence of increased call acceptance with a higher value of number of quantization levels is accounted for in the following way. As the number of quantization levels increases, it is seen that the average hop length also increases. Of the possible routes in the network, those corresponding to higher bandwidth quantization levels (with larger bandwidth requirement) take longer paths. This is because the pre-computation algorithm recursively chooses from higher bandwidth quantization level to lower bandwidth quantization level. The nature of the

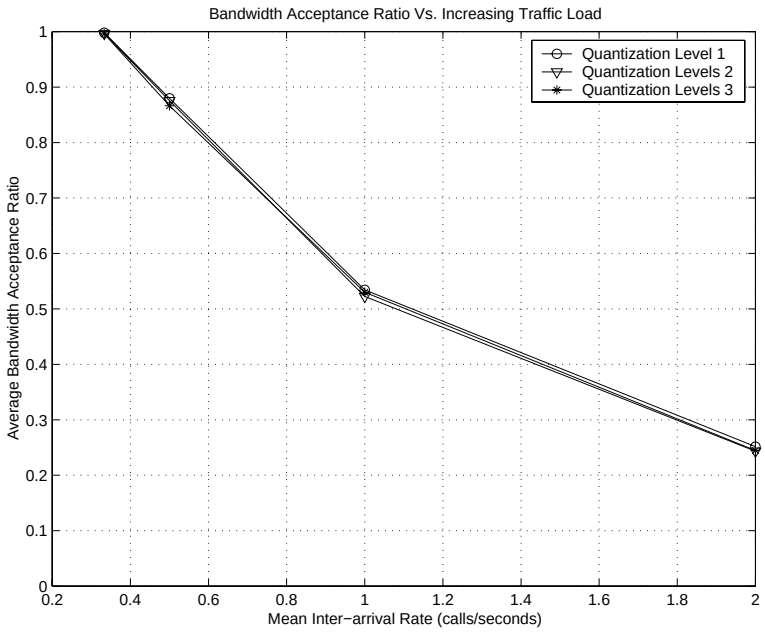

Figure 5.10: Avg. Bandwidth Acceptance with Increasing Call Load for Varying Quantization Levels

network is such that finding a larger bandwidth route tends to use whatever route is available after pruning — even those that take a peculiar path that is long.

As a result, there is an increase in the average hop count of the routes. As the hop count increases, there is greater possibility for the call to get rejected. This offsets the gain obtained, when using a higher number of quantization levels, of using better bandwidth selection from among the routes in the cache.

5.3.3 Effect of Pre-Computation Update Cost

The cost of pre-computation is modeled using a safe or pessimistic approximation, which is described in 3.4.3. The real time for pre-computation of the route cache is measured, and the cache is disabled for a period of time that is a scaled version of the real pre-computation time. The scaling factor is tunable and mimics the computation load on another processor. While the cache is disabled, it cannot be

consulted for routing calls.

Performance Metrics

The metric of interest is the average number of cache misses. The cache cost model disables the route cache while it is being updated, and hence the number of cache misses is the penalty paid due to the cache computation. This metric is chosen since it emphasizes the worst case scenario, of making the cache unusable while it is being updated. However, it should be noted that a cache miss is different from a route failure. When the cache miss occurs, an on-demand computation is carried out and the route is computed.

Performance Parameters

We look at the case where the pre-computation scaling factor takes values: 0, 1, 3, 5, and 10, for fixed values of number of quantization levels (1), and mean inter-arrival time of calls (3 seconds). We also study the effect of increasing number of quantization levels while keeping the scaling factor constant.

Experimental Design

For a standard edge-core topology, subjected to a standard traffic mix, we vary the scaling factor for pre-computation cache update cost, and measure the number of cache misses. The use of a standard topology provides a common network to test, while the use of a standard traffic mix eliminates the effect of variance in load on the call rejection.

Results

The results plotted in Figure 5.11 , show that as the scaling factor increases, there is a corresponding increase in the number of cache misses. This is in accordance with the basic assumption of the simple cost model, since an increase in the scaling

factor implies greater time that the cache is unusable and thus greater number of cache misses.

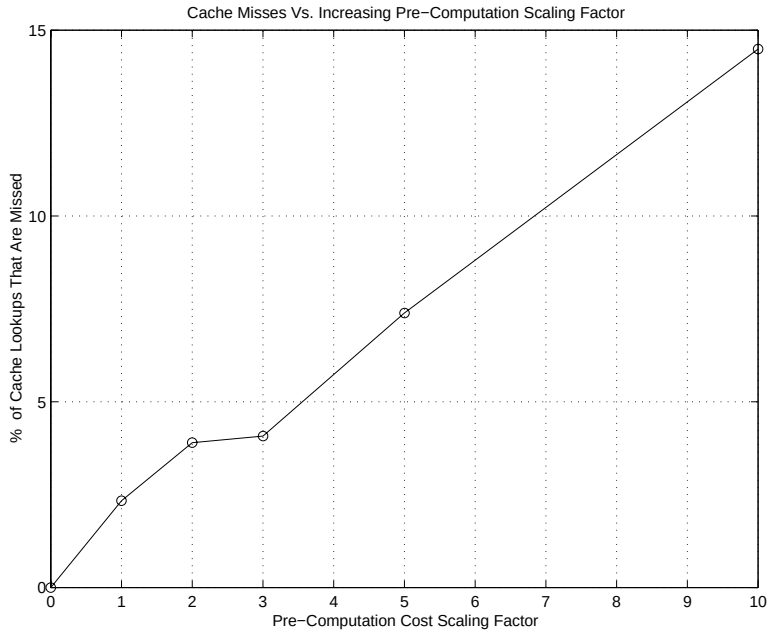

Figure 5.11: Cache Misses with Increasing Pre-Computation Cost Scaling Factor

Further, we see in Figure 5.12 that as the number of quantization levels increases, there is a linear increase in the number of cache misses. Once again, this is in accordance with our cost model. An increase in the number of quantization levels implies a greater number of route entries in the cache and a corresponding increase in the cache update processing time. This means that the cache is rendered unusable for longer periods of time and thus the number of cache misses increases.

5.3.4 Conclusion

We see from the above experiments that the effect of increasing the number of peer groups is an increase in the number of calls failed in foreign peer groups. However, the fact that source peer group failed calls decreases, indicates that pre-computation of route caches is beneficial for routing within the source peer

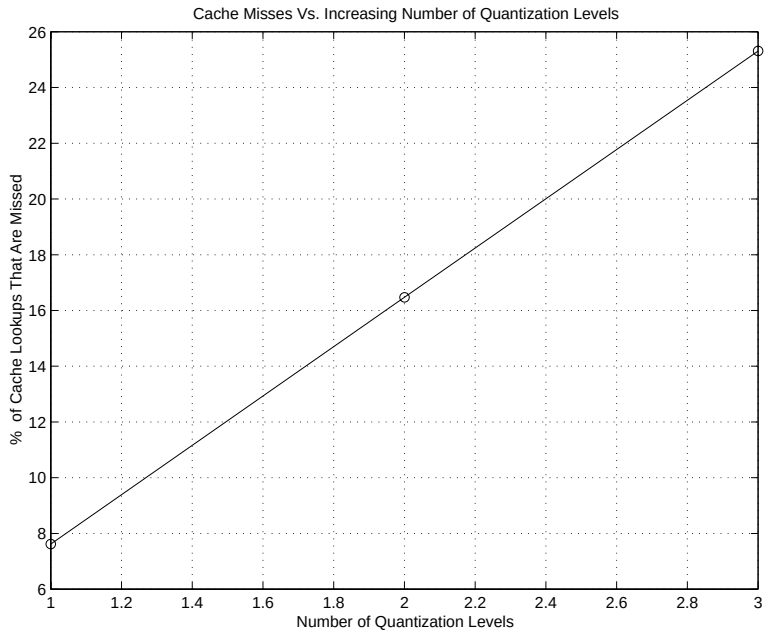

Figure 5.12: Cache Misses with Increasing Number of Quantization Levels

group. The average call setup time decreases as the number of peer groups increase. This decrease is greater for pre-computed routing strategy compared to on-demand routing.

It is observed that increasing the number of quantization levels does not increase the average bandwidth acceptance ratio. Hence the number of quantization levels is not significant in evaluating pre-computed route caching for the chosen network.

Finally, we see that even with an overly pessimistic model for the cost of route cache updates, pre-computation causes only 14 % of cache lookups to fail in finding a cached route.

5.4 Heuristics for Initiating Route Cache Updates

The following experiments consider the problem of "when to update the route cache". It was observed in Section 5.3.1 that without route cache updates, pre-computed routes fail mainly in the foreign peer groups. This is because the information about foreign peer groups has lower quality, and ages faster, when compared to source peer group information. In order to improve the performance of the pre-computed routing strategy, we introduce route cache updates. We look at different heuristics for initiating a route cache update. It should be noted that we now have a re-aggregation timer that initiates re-aggregation and flooding of foreign peer group information. First we study a *periodic timer* based heuristic, followed by a comparison of the other cache update heuristics described in Section 3.5.

5.4.1 Timer Controlled Updates

This experiment is intended to bring out the relationship between re-aggregation period and timer based updates to the pre-computed route cache.

Performance Metrics

Average bandwidth acceptance ratio and average call setup times are the metrics of interest in this experiment.

Performance Parameters

Cache updates are carried out at regular intervals to refresh the route cache so that old routes created using old information can be flushed out and a new set of paths can be computed and filled. The update timer period determines how often this process occurs. We vary the period of the cache update timer, so as to incur from 1 to 5 cache updates during the duration of the experiment.

Experimental Design

We take a standard edge core topology with standard traffic mix. We set the re-aggregation timer value such that there are 3 re-aggregation events in each peer group, during the course of the experiment.

It is important to note that the route caching policy now has the property that when a crankback message arrives for a route that was chosen from the route cache, that particular route is invalidated. The next time a route is requested to the invalidated destination, a cache miss occurs, and an on-demand computation is carried out. The newly computed route is of course reinserted in the route cache.

Hypothesis

Route caching is done in every node that can be the source of a call. It is also done at every border node, which serves as the ingress node for a transit peer group. A good measure of the update condition for the route cache is when there is substantial change in the network resources.

For the transit node, this would be when aggregation occurs. This is because aggregation defines the time when there is enough change in the network to cause network wide flooding. At that time, the topology database is refreshed with information from other peer groups. Hence the cache update performed at this time would have better information about foreign peer groups.

Further, since most of pre-computed route call failures occur in foreign peer groups, it is expected that when the route cache update timer is synchronized with the re-aggregation timer, the overall performance should increase to give lower average call setup time and higher average call acceptance ratio.

Results

Figure 5.13 shows how the average call setup time changes with increasing value of route cache update period. It is seen that at a value of **100** seconds, there is a

marked decrease in the average call setup time. This is because that value of the route cache update period is identical to the re-aggregation timer value. The routes in the cache are recomputed based on the latest information. The resulting increase in successful establishment of routes picked up from the route cache cause fewer crankback events. This implies that fewer entries are invalidated and hence there are fewer invocations of the on-demand computation, resulting in lower average call setup times.

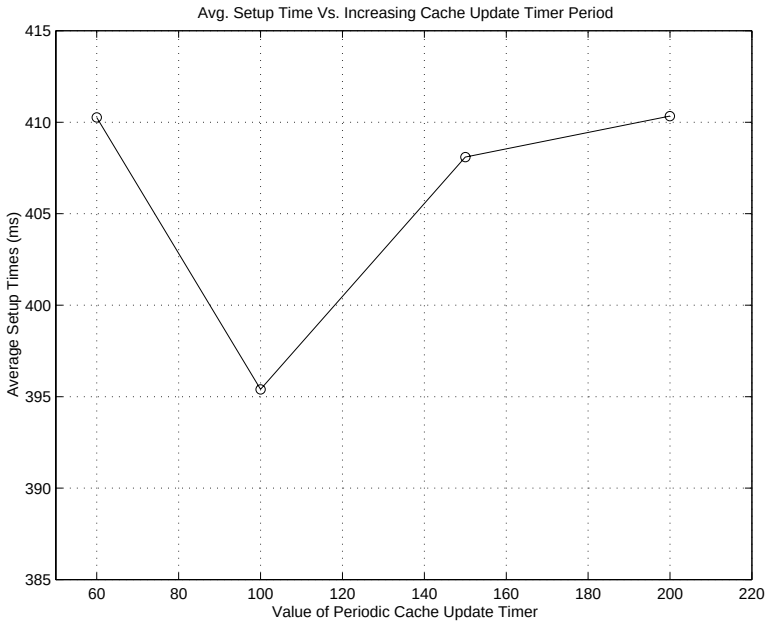

Figure 5.13: Average Setup Time with Increasing Cache Update Timer Period

Figure 5.14 shows how the average bandwidth acceptance ratio changes with increasing value of route cache update period. Once again, it is observed that the average bandwidth acceptance ratio increases sharply when the route cache update period is equal to the re-aggregation period. This indicates that our hypothesis is correct and validates the assertion that:

> updating the cache of pre-computed routes at the same rate as the re-aggregation rate would maximize the quality of the pre-computed route

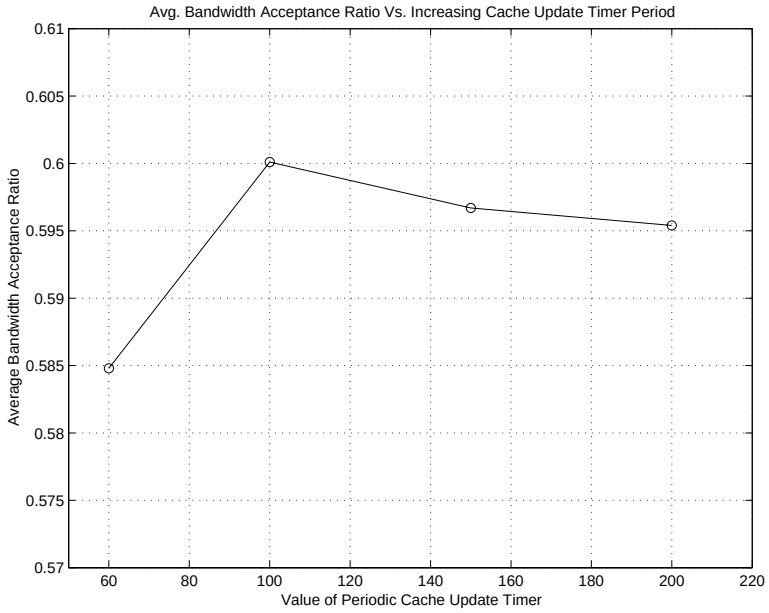

Figure 5.14: Average Bandwidth Acceptance Ratio with Increasing Cache Update Timer Period

cache, and decrease average call setup times while increasing average bandwidth acceptance ratio.

We find that the call acceptance ratio follows almost exactly the same curve as the bandwidth acceptance ratio, both in value and trend. Hence we do not present the essentially identical plot for average call acceptance ratio.

5.4.2 Comparison of Heuristics for Updating the Cache

Performance Metrics

In this experiment, average call acceptance ratio, and average call setup times are the metrics of primary interest. We also need to compare the different heuristics that we experiment with based on the two primary metrics. For that purpose we generate a combined metric. We are influenced by the understanding that a lower value of call setup time is favorable, while a higher value of call acceptance is also

favorable. Hence we take the call rejection, for which a lower value is favorable, and multiply it with the call setup time. This give the combined metric using the following formula:

$$combined\_metric = average\_call\_rejection\_ratio * average\_call\_setup\_time$$

The heuristic with lowest value of the combined metric exhibits the best combination of lowering average call setup times and increasing average call acceptance ratio.

Performance Parameters

We conduct the experiment for on-demand routing, as well as pre-computed routing strategies. We choose the heuristic for determining when to initiate route cache updates from among the following:

- Crankback initiated invalidation (cbk): here, a crankback indication is used to invalidate the entry for that destination from the route cache. This is described in Section 3.5.1.
- Timer based updates (timer): here the timer value is set to the re-aggregation value so as to get best performance for timer based updates, as shown in Section 5.4.1.
- Ptse Count based updates (ptse): here the number of updated PTSE's for each level is tracked and a cache update is initiated for the level that reaches significant change in the PTSE count. The level of significance is set to the convergence number as described in Section 3.5.3.
- Crankback-Ptse Count (cbkptse): here we try a combination of crankback initiated invalidation and ptse count based cache updates.

- Crankback-Timer (cbktimer): here we try a combination of crankback initiated invalidation and timer based cache updates. Once more, the timer is set to the value of re-aggregation timer.

- Ptse Count-Timer (ptsetimer): here we try a combination of ptse count based and timer based update heuristics.

- Ptse Count-Timer-Crankback (ptsetimercbk): here we try a combination of ptse count based, timer based update heuristics, supplemented by crankback initiated invalidation.

- None (none): This is the trivial case having no updates what-so-ever and is used for comparison purposes.

Experimental Design

We take a standard edge core topology with standard traffic mix. The network re-aggregation timer is given a value that will cause 3 re-aggregation events, during the simulated duration of the experiment.

Hypothesis

Since PTSE's are generated by nodes after a "significant change", the *ptse count* for a given node or addressing level, is a good estimator of how much the current cache contents may differ from the current network state. The use of crankback initiated invalidation improves the accuracy of future routes.

Timer based cache update policy does not take into consideration the effect of updated information that occurs due to flooding within the source peer group. Further, depending on the network conditions, cache updates might occur even when there is no change in the network resources.

For these reasons, we believe that the heuristic combining two objectives of cache content relevance and crankback invalidation (Crankback-Ptse Count)

would perform best in terms of the combined metric of comparison, by increasing the average call acceptance ratio, and simultaneously lowering the average call setup time.

Results

Figure 5.15, plots the average call setup time for each of the eight policies described above. It is observed that crankback initiated invalidation alone gives the highest value of call setup time. Further, introducing timer based updates tends to increase the average call setup times. This is seen in the increase of **50** ms in call setup value for Ptse count based updates to the Ptse-Timer based combination. Ptse count based updates, shows average call setup times that are comparable to the case where no updates occur. There is only marginal decrease in call setup time when we add crankback initiated invalidation to the Ptse crankback based updates. The average call setup time ranges from **315** ms for no updates at all, to **425** ms for crankback initiated invalidation heuristic.

Figure 5.16, plots the average call acceptance ratio for each of the eight policies described above. On-demand computation shows best results, and ptse based updates as a family perform better than no updates. However, when we add crankback initiated invalidation to the Ptse count method, we see an increase in the average call acceptance ratio. The average call acceptance ranges from **71** % for no updates at all, to **83** % for on-demand routing.

Considering the combined metric, Figure 5.17 plots the computed values of the combined metric (weighted by a factor of 0.01) versus the different heuristics. The correlation between the observations of average call setup times and average call rejection ratio was **0.025**. This indicates that the two factors, call setup time and call rejection, are marginally related. When one increases, the other increases marginally. This makes the product of the two primary metrics a realistic measure for comparison.

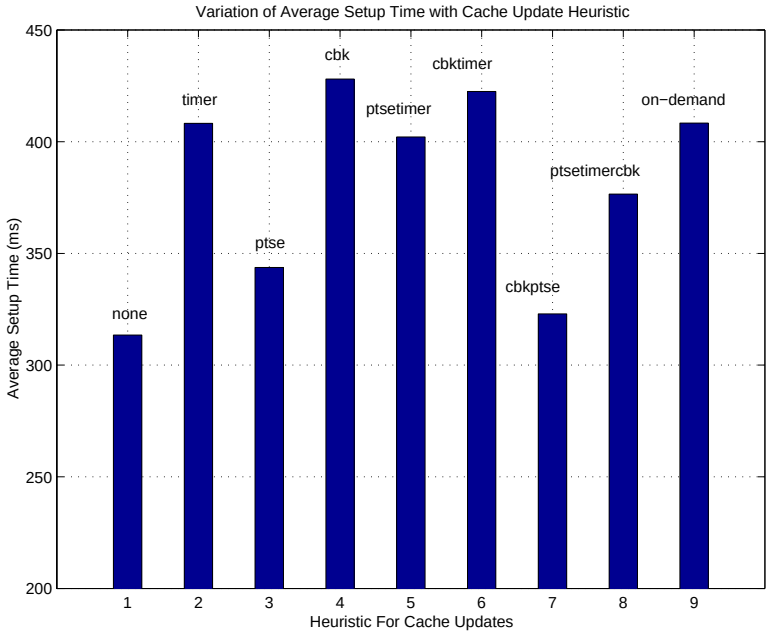

Figure 5.15: Variation of Average Setup Time with Cache Update Heuristic

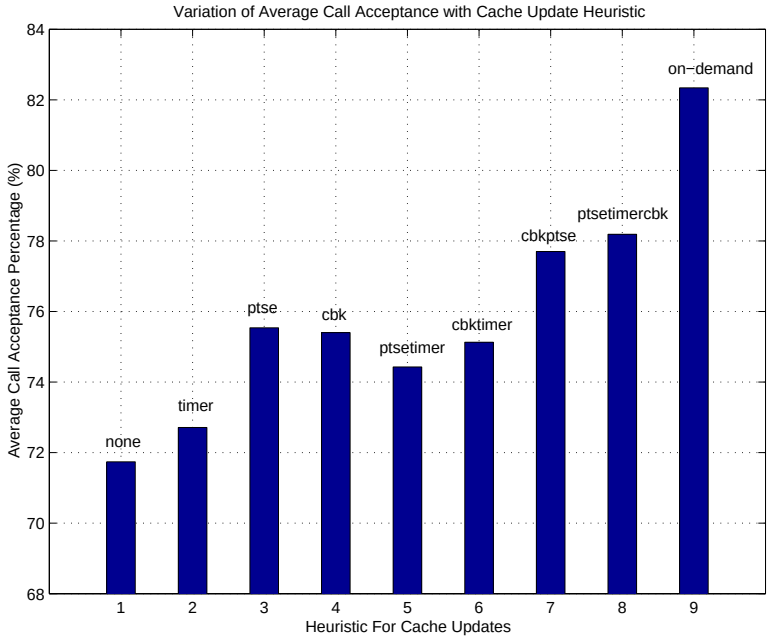

Figure 5.16: Variation of Average Call Acceptance Ratio with Cache Update Heuristic

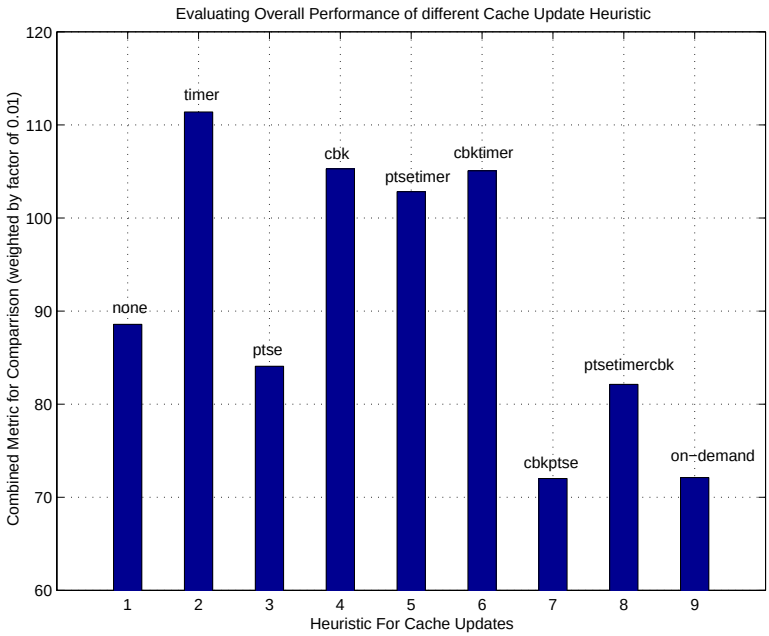

Figure 5.17: Evaluating Overall Performance of Route Cache Update Heuristics

The ptse count based heuristic performs comparably to the case where there are no updates. Having no updates gives lower average call setup times, but it also gives lower average call acceptance ratio. The ptse count based heuristic gives higher average call setup times than the no updates method, but it has higher average call acceptance ratios too. The use of timer based update heuristic tends to increase the combined metric of evaluation.

The heuristic that uses Ptse-Crankback performs better than all other heuristics for the combined metric of comparison. This is because, it exhibits the best combination of lower average call setup times and higher average call acceptance ratio.

5.4.3 Conclusion

When the routing updates are synchronized with the aggregation event, we expect the newly flushed route cache to have routes that are based on latest aggregated

information. Hence the call acceptance increases in comparison to when cache refresh rates are mis-matched with the default aggregation rate.

An improvement in the average call acceptance ratio with a corresponding decrease in the average call setup time, as compared to the other cache update heuristics, show that the *ptse count - crankback initiated invalidation* method is a better heuristic for determining when to update the route cache. This is because the PTSE count is a good measure of network flux for initiating route cache updates, while the crankback initiated invalidation helps to keep the route cache up to date.

5.5 Multiple Peer Group with Route Cache Updates

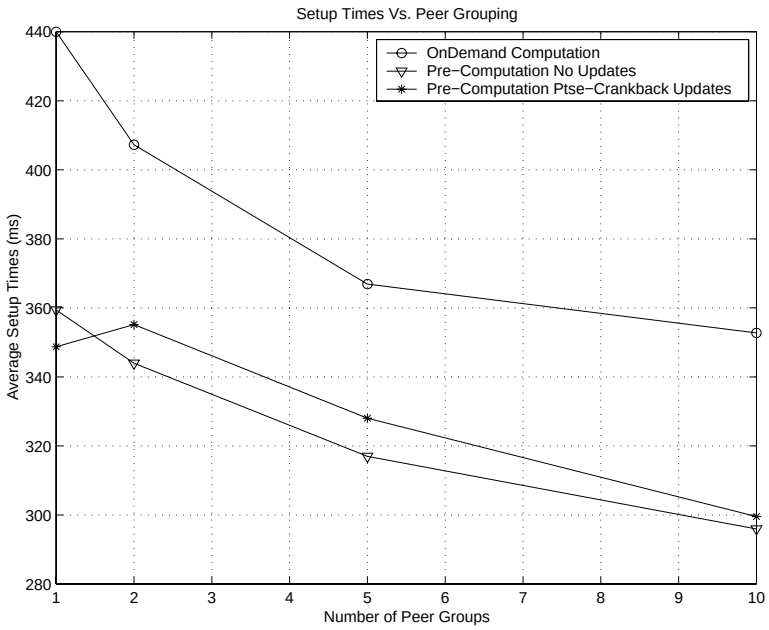

Figure 5.18: Average Setup Time with Peer Group Size (ptse count - crankback update heuristic)

On the basis of our conclusion, we re-evaluate the experiments of varying multiple peer group size (see Section 5.3.1). But this time, we include the *ptse count*

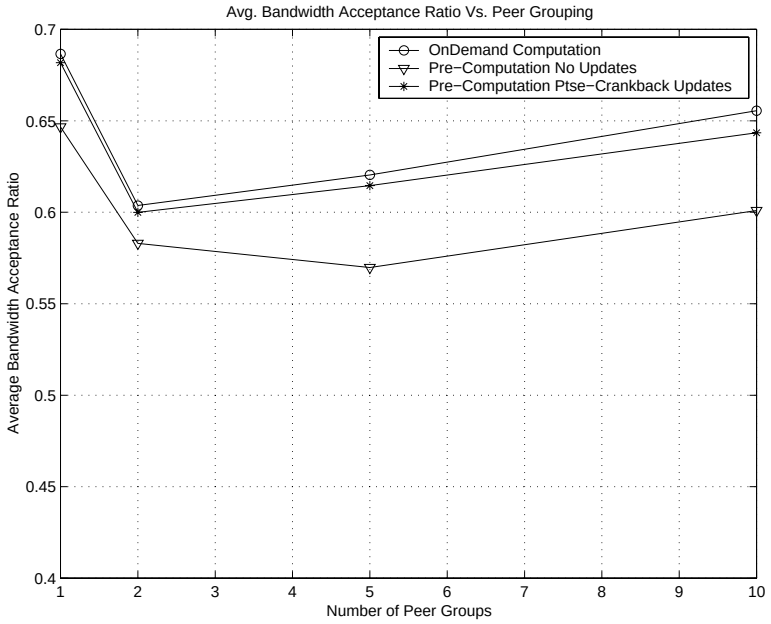

Figure 5.19: Average Bandwidth Acceptance with Peer Group Size (ptse count - crankback update heuristic)

with crankback initiated invalidation cache update policy along with re-aggregation. Figure 5.18 shows that the average call setup time has decreased for Pre-Computed routing strategy with route cache updates. However, Figure 5.19 shows that there is a also an increase in the average bandwidth acceptance ratio. Without updates, the average call setup time is only slightly different from the value with cache updates. Also, when cache updates are introduced, the call acceptance is increased to reach values that are close to those for on-demand computation.

This is primarily due to the increase in the number of calls succeeding in foreign peer groups. As is seen in Figure 5.20, without cache updates, the location of failures is primarily in the foreign peer groups. However, when *ptse count with crankback initiated invalidation* heuristic is applied for route cache updates, Figure 5.21 shows that the failures in foreign nodes drastically decreases.

Thus the use of the *ptse count with crankback initiated invalidation* heuristic

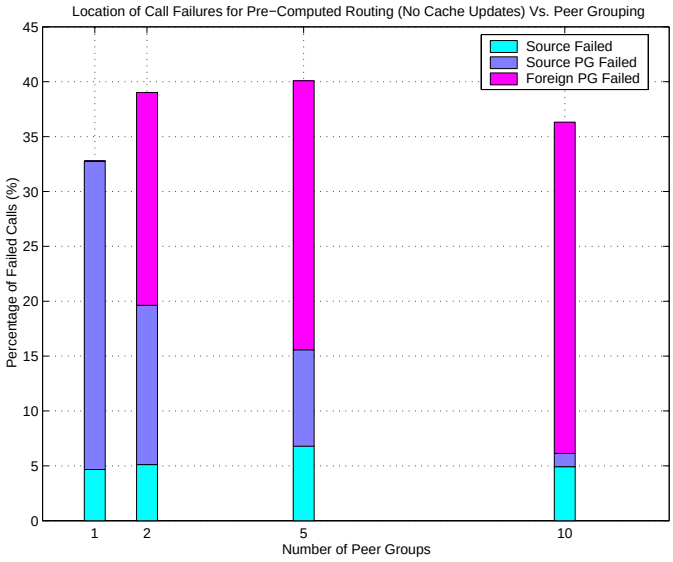

Figure 5.20: Location of Call Failures for Pre-Computed Routing (without cache updates) with Peer Group Size

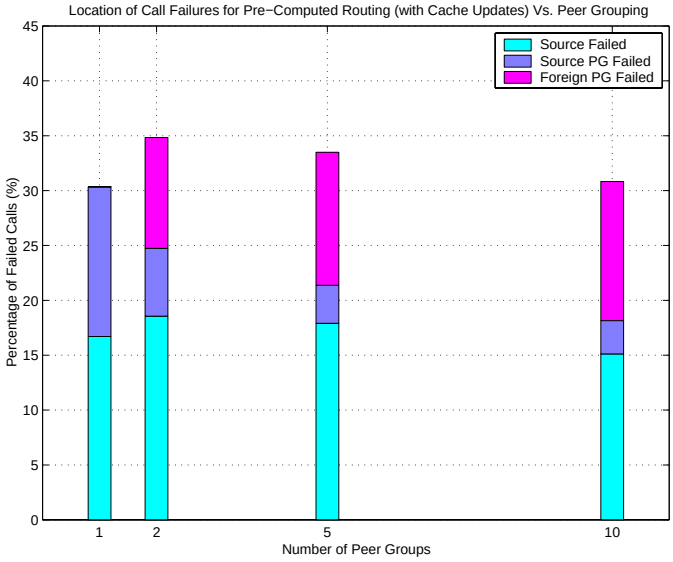

Figure 5.21: Location of Call Failures for Pre-Computed Routing (ptse count - crankback update heuristic) with Peer Group Size

brings out the advantage of avoiding foreign peer group failures, and the accompanying lowering of total signalling load.

The improvement in terms of lowering average call setup time is best represented by using a histogram of route computation times. We consider the case of a **2** level multiple peer group topology, having **5** peer groups at the physical level. Figure 5.22 shows the histogram for on-demand route processing times.

It is seen that there are three peaks corresponding to the following routing decisions:

- lower region peak: this represents the route computation for destinations within the source peer group, or at the terminating peer group.
- middle region peak: this represents the route computation for transit routing occurring at an ingress border node for getting the call across the peer group.
- higher region peak: this represents the high source route computation involving extraction of the entire topology database and routing across foreign peer groups.

Pre-computation involves a search through a list of route entries in the route cache. If there is a miss, an on-demand route computation is carried out. Figure 5.23, shows how the histogram shifts when on-demand routing strategy is replaced with pre-computed routing strategy. The same number of calls are made in both cases. However, with pre-computation, the majority of calls are in the lower end of the histogram, corresponding to the cache hits. The components corresponding to cache misses occur for terminating peer group routing, transit peer group routing, and source node routing. The cache misses are followed by on-demand route computation. These are indicated in the histogram.

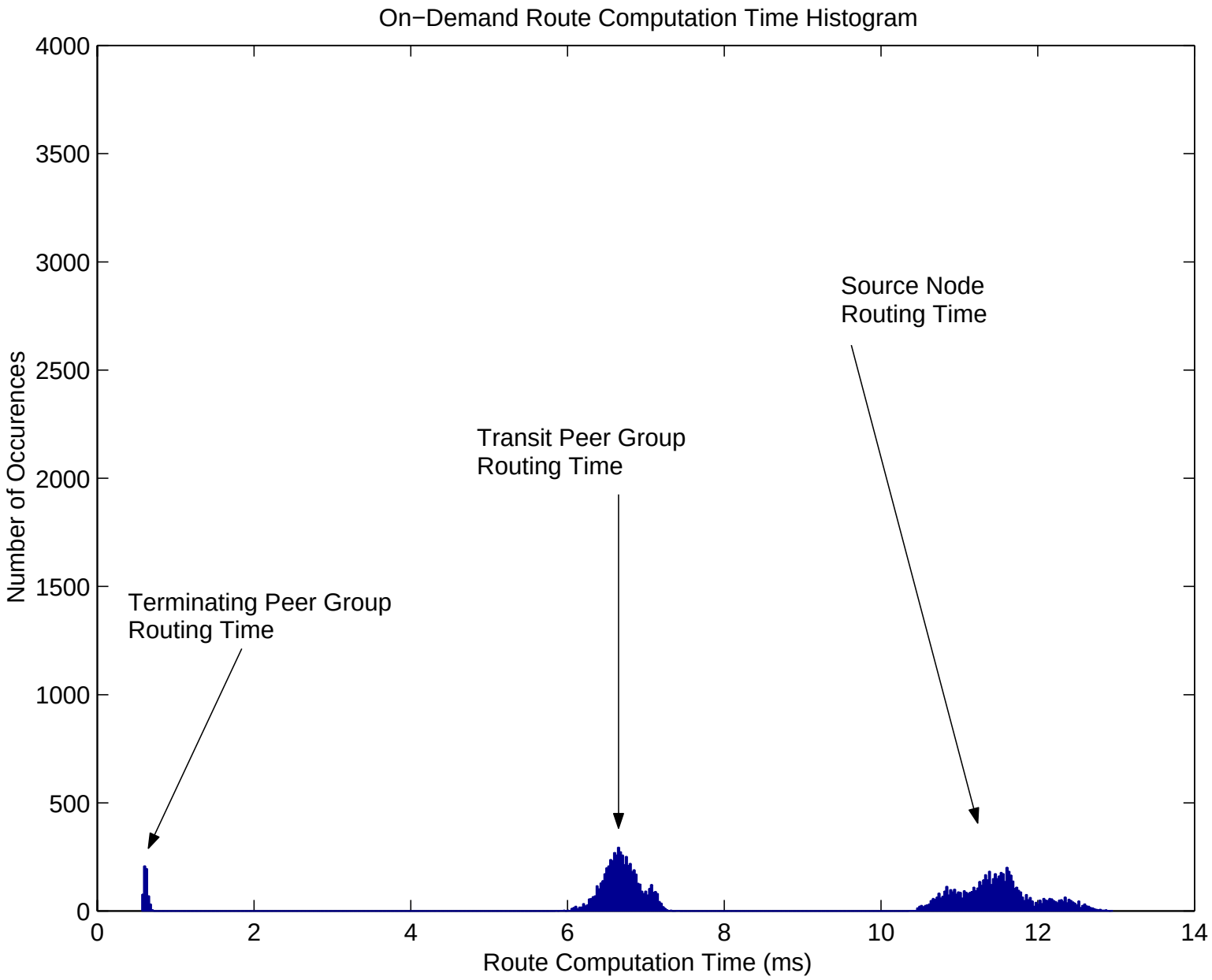

Figure 5.22: Histogram of On-Demand Route Processing Times

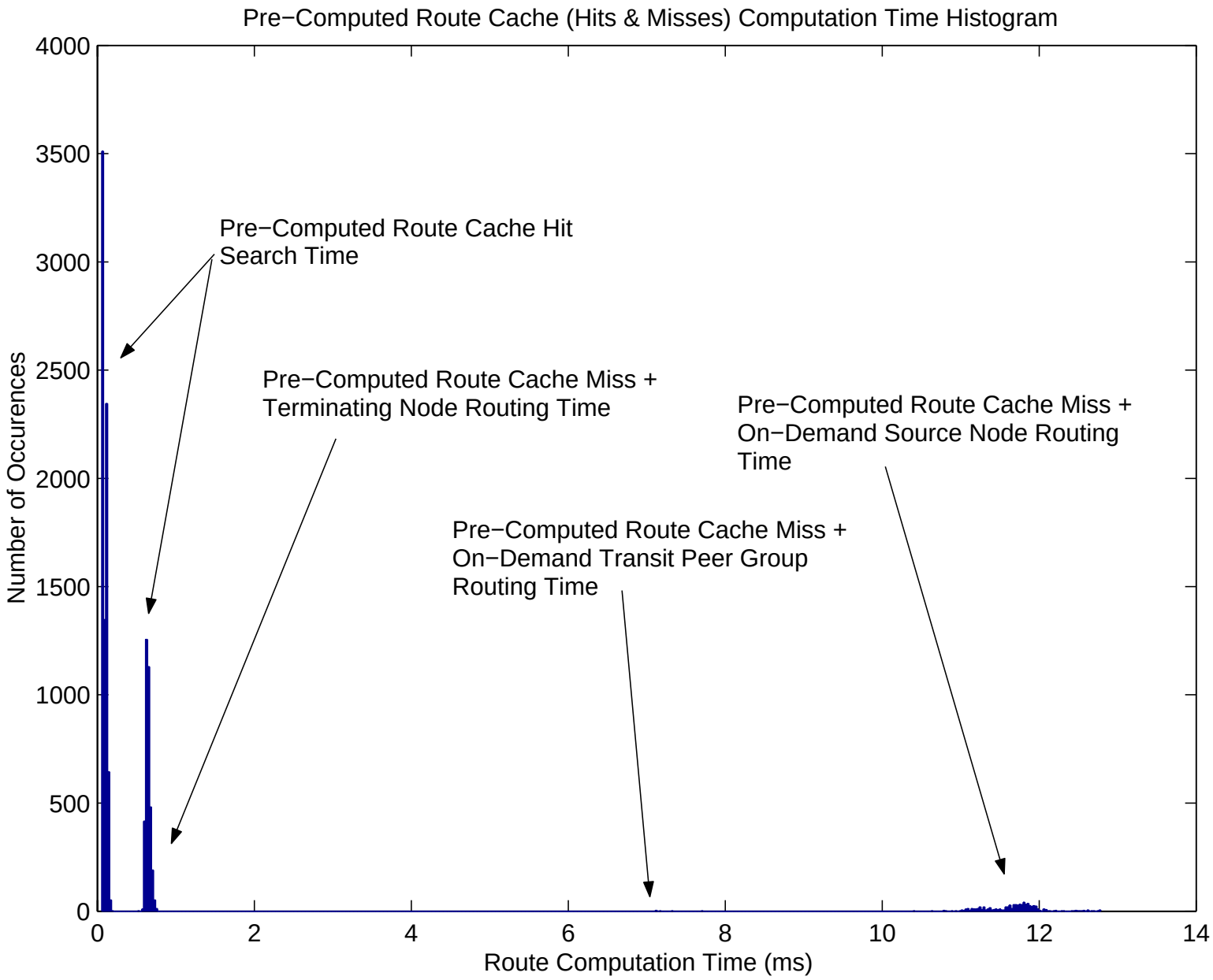

Figure 5.23: Histogram of Pre-Computed Route Processing Times

5.5.1 Other MPG Topologies

We also looked at other non Edge-Core topologies to verify that the *crankback - ptse count* heuristic is not dependent on a specific network topology. Experiments were carried out on:

- 3 level Multiple Peer Group network.
- Cluster network containing 8 nodes per cluster, and 5 such clusters.
- 60 node network divided into 5 peer groups (not conforming to edge core topology).

In all cases, there was an improvement in call setup time with a corresponding increase in the call acceptance was observed for pre-computed route caching strategy with the *crankback - ptse count* update heuristic. This was in comparison to pre-computed route caching without updates. Also, the performance of pre-computed routing with *crankback - ptse count* update heuristic was close to that of on-demand routing strategy. This is consistent with the results obtained with Edge-Core topology.

Chapter 6

Conclusions and Future Work

Thus grew the tale of Wonderland:
Thus slowly, one by one,
Its quaint events were hammered out -
And now the tale is done,
(but then.. research never ends..)

Routing decisions within the PNNI protocol involves topology database queries and expensive graph manipulations, for each incoming call connection request requiring route computation. The research presented in this thesis looks at how to amortize the cost of route computation across several such call requests. We suggest pre-computation of routes that are stored in a *route cache* for future lookup and routing.

Route caching decreases the cost of route computation, and thus decreases the average call setup times. However, this comes at the price of decreased call acceptance, due to the fact that pre-computation uses topology state information that will age by the time the actual call request arrives.

As such, our cache model is relatively simple. Since we wish to illustrate the advantages of route caching over on-demand computations, we use the identical routing algorithm, and minimally change the basic routing methodology. We com-

pute routes to destination nodes present in a node's topology database. Routes are computed on the basis of a set of equivalence classes of bandwidth available in the network. This is done so as to be able to map the bandwidth requirements of the arriving call to the best pre-computed route. Our pre-computed route caching model conducts re-computation of routes so as to update the route cache and increase the average call success. We studied different route cache update heuristics and compared their effectiveness in achieving an optimum combination of decreased average call setup times and increased average call success.

We proposed the *ptse count* method for determining when a cache update needs to be initiated. This method relies on counting the number of PNNI Topology State Elements (PTSE) that are flooded among all nodes within the network, as part of the PNNI protocol. Our solution is based on the fact that since PTSEs are generated by a node because of significant change in its resource availability. The number of PTSEs is a good indicator of network flux. This is better than say, a *timer based* heuristic that forces route computation periodically irrespective of whether the network state has changed or not. While we saw that the *timer based* heuristic performs best, when the update period is synchronized with the topology re-aggregation period, we also found that the use of a timer does not follow network state changes as well as the *ptse count* method. We conclude that the *crankback initiated invalidation* of cache entries, enhanced by *ptse count* based cache updates gives the best combination of decreased average call setup time and increased average call success. The results of our experiments showed that a pre-computed route caching scheme using this combined heuristic achieves performance comparable to on-demand computation of routes.

Our simulation platform, the KU-PNNI Simulator, shares more than 90% of its code with *real* off-board Q.Port signalling code. This makes it a good platform for simulating large ATM networks, especially those running a complex routing protocol like PNNI. The results obtained were statistically tested for data integrity

and consistency. We find that for call setup time, call acceptance ratio and call bandwidth acceptance, the values passed the data integrity test at a confidence level of 95% with the samples lying on average within 2% of the population mean.

The *ptse count* with *crankback invalidation* heuristic was tested under three different network topologies, and the results validated our hypothesis in every case. Finally, our overly pessimistic traffic load model, and the simple structure of our route caching policy indicate that for a real world network, having a smarter route caching method, the results would tend to be even more favorable for pre-computed route caching, as compared to on-demand computation of routes.

For future work, the route cache policy can be enhanced in several areas including: route pre-computation, more efficient storage of routes, verification of a chosen cache entry before it is used for connection establishment, introducing crankback retries and alternate routing. The route computation cost model is overly pessimistic and can be improved to include a more realistic representation of background computation.

Our results indicate that route caching is far better at choosing routes within the source peer group, as compared to foreign peer groups. This suggests a hybrid routing algorithm that does on-demand computations for determining the foreign peer groups to traverse in a route, and then consults a route cache for crossing the source peer group to reach the border node to the first foreign peer group.

Our experiments show that the *ptse count* method is a good estimate for network flux. While we use this as an indication to update a pre-computed route cache, it would not be difficult to apply this to say, the problem of when to re-aggregate logical level topology state information within peer group leaders. As the PNNI protocol improves to encompass inter-networking issues, the use of the *ptse count* method for interacting with IP based routing protocols like OSPF, BGP and so on, may be prospective areas for investigation.

Bibliography

[1] G. Apostolopoulos, R. Guerin, S. Kamat, and S. Tripathi. Quality of Service based Routing: A Performance Perspective. In *Proc. ACM SIGCOMM*, September 1998.

[2] G. Apostolopoulos and S. K. Tripathi. On the Effectiveness of Path Pre-Computation in Reducing the Processing Cost of On-Demand QoS Path Computation. In *Proceedings of IEEE Symposium on Computers and Communication*, June 1998.

[3] Bell Communications Research, Inc. *Q.Port: Portable ATM Signaling Software*, September 1996.

[4] J. Le Boudec and B. Przygienda. A Route Pre-Computation Algorithm for Integrated Services Networks. Technical Report 95/113, DI-EPFL, CH1015 Lausanne, Switzerland, February 1995.

[5] I. Castineyra, N. Chiappa, and M. Steenstrup. The Nimrod Routing Architecture. In *Request for Comments (RFC)*, August 1992.

[6] ATM Forum Technical Committee. *ATM Private Network-Network Interface specification version 1.0 (af-pnni-0055.000)*. ATM Forum, March 1996.

[7] E. Crawley, R. Nair, B. Rajagopalan, and H. Sandick. A Framework for Qos-based Routing in the Internet. *draft-ietf-qosr-framework-04*, April 1998.

[8] ATM Forum. http://www.atmforum.com/.

[9] R.J. Gibbens, F. P. Kelly, and P. B. Key. Dynamic Alternate Routing - Modeling and Behaviour. Technical Report ITC-12, Tele-trafic Science for New Cost-Effective Systems, Networks and Services, Elsevier Science Publishers, 1989.

[10] R. Guerin, A. Orda, and D. Williams. Qos Routing Mechanisms and OSPF Extensions. In *Proceedings of IEEE GLOBECOM*, Phoenix, AZ, November 1997.

[11] A. Guillen, R. Najmabadi Kia, and B. Sales. An Architecture for Virtual Circuit QoS Routing. *Proceeding of First IEEE International Conference on Network Protocol*, pages 80–87, 1993.

[12] Fang Hao and Ellen W. Zegura. Scalability Techniques in QoS Routing. Technical Report GIT-CC-99-04, College of Computing, Georgia Tech, Atlanta, Georgia, 1994.

[13] Information Technology and Telecommunication Center, University of Kansas. *KU PNNI User's Manual*, June 2000.

[14] A. Iwata, R. Izmailov, H. Suzuki, and B. Sengupta. PNNI Routing Algorithms for Multimedia ATM Internet. *NEC Research and Development*, 38, January 1997.

[15] R. S. Krupp. Stabilization of Alternate Routing Networks. In *IEEE International Communication Conference*, Philadelphia, PA, 1982.

[16] Averill M. Law and W. David Kelton. *Simulation Modelling and Analysis, 3ed.* McGraw-Hill, ISBN 0-07-059292-6, 1999.

[17] W. C. Lee, M. G. Hluchyj, and P. A. Humblet. Routing Subject to Quality of Service Constraints in Integrated Communication Networks. *IEEE Network Magazine*, pages 46–55, July / August 1995.

[18] Richard I. Levin and Davis S. Rubin. *Statistics for Management*. Prentice-Hall, December 1998.

[19] Q. Ma and P. Steenkiste. On Path Selection for Traffic with Bandwidth Guarantees. *Proceeding of IEEE International Conference on Network Protocols*, October 1997.

[20] M. Peyravian and A. D. Kshemkalyani. Network Path Caching: Issues, Algorithms and a Simulation Study. *Computer Communications*, 20:605–614, 1997.

[21] Phongsak Prasithsangaree. *Performance Evaluation of Multiple Criteria Routing Algorithms in Large PNNI ATM Networks, M.S. thesis.* Department of Electrical Engineering and Computer Science, University of Kansas, June 2000.

[22] A. Shaikh, J. Rexford, and K. Shin. Dynamics of Quality-of-Service Routing with Inaccurate Link-State Information. Technical Report CSE-TR-350-97, Computer Science and Engineering Division, University of Michigan, Ann Arbor, MI, November 1997.

[23] Z. Wang and J. Crowcroft. Quality of Service Routing for Supporting Multimedia applications. *IEEE J. Selected Areas of Comm*, 14:1228–1234, 1996.