MAXIMIZING NETWORK RESOURCE UTILIZATION
THROUGH DYNAMIC DELAY ALLOCATION ADJUSTMENT

A THESIS SUBMITTED TO THE GRADUATED DIVISION OF THE UNIVERSITY OF HAWAI'I IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
ELECTRICAL ENGINEERING

AUGUST 2007

By
Xiaojiang Liu

Thesis Committee:
Yingfei Dong, Chairperson
Galen Sasaki
Tep Dobry
We certify that we have read this thesis and that, in our opinion, it is satisfactory in scope and quality as a thesis for the degree of Master of Science in Electrical Engineering.

THESIS COMMITTEE

[Signatures]
Chairperson

[Date]
Acknowledgement

This material is based upon work supported by the National Science Foundation under Grant No. 0649950. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

I would like to thank my advisor, Dr. Yingfei Dong, for his guidance, patience, and help throughout my research and study at Department of Electrical Engineering, University of Hawai‘i. I also would like to thank the members of my thesis committee: Dr. Galen Sasaki and Dr. Tep Dobry. Thanks for your advice and support.
Abstract

Quality of Service (QoS) has been an important topic in network research, and many solutions have been proposed to address QoS related issues. In this project, we focus on the delay requirement partition issue for maximizing the network utilization. We first survey conventional schemes, and point out their limitations: they all perform static allocations based on instant load situations, which may cause imbalanced reservations and bottleneck links. We then propose a novel Dynamic Allocation Adjustment (DAA) algorithm to address these problems by dynamically adjusting the existing reservations for earlier admitted flows. DAA not only spreads traffic evenly onto intermediate links but also balances link loads in a broader range. As a result, link congestion on a flow path is alleviated and its total reservation is reduced. On the other hand, DAA addresses the bottleneck link problem. We conducted our simulations on both of symmetric and asymmetric topologies, with uniformly distributed traffic and several types of imbalanced traffic. The results show that the improvement of system utilization is over 30%. 
# Table of Contents

Acknowledgement ..................................................................................................... iii
Abstract ...................................................................................................................... iv
List of Tables ............................................................................................................... vii
List of Figures ............................................................................................................ viii
Chapter 1: Introduction ................................................................................................. 1
Chapter 2: Background ................................................................................................. 7
  2.1 Typical QoS Models ............................................................................................ 7
  2.2 QoS Routing ........................................................................................................ 8
  2.3 Resource Allocation ........................................................................................... 11
     2.2.1 Per-Hop Resource Allocation ....................................................................... 11
     2.2.2 Per-Path Resource Allocation .................................................................... 14
Chapter 3: Problem Formulation ................................................................................ 18
  3.1 Problem to be addressed .................................................................................... 18
  3.2 Motivations ......................................................................................................... 20
  3.3 Overview of Scheme & Adjustment .................................................................... 22
Chapter 4: Dynamic Allocation Adjustment ............................................................... 24
  4.1 Definitions ......................................................................................................... 24
     4.1.1 Flow Specification ....................................................................................... 24
     4.1.2 Routing and Admission Control ................................................................. 25
  4.2 LSS Review ....................................................................................................... 26
     4.2.1 Smoothing Delay and Delay Allocation ...................................................... 26
     4.2.2 Review of LSS algorithm .......................................................................... 27
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.3</td>
<td>DAA Overview</td>
<td>29</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Global Goal</td>
<td>29</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Critical Link</td>
<td>30</td>
</tr>
<tr>
<td>4.3.3</td>
<td>DAA Algorithm</td>
<td>30</td>
</tr>
<tr>
<td>4.4</td>
<td>Important Issues</td>
<td>31</td>
</tr>
<tr>
<td>4.4.1</td>
<td>How often to adjust</td>
<td>31</td>
</tr>
<tr>
<td>4.4.2</td>
<td>How to choose links and associated flows</td>
<td>32</td>
</tr>
<tr>
<td>4.4.3</td>
<td>How adjustment works</td>
<td>33</td>
</tr>
<tr>
<td>4.5</td>
<td>Global Balance Adjustment (GBA)</td>
<td>36</td>
</tr>
<tr>
<td>4.6</td>
<td>Local Dynamic Adjustment (LDA)</td>
<td>38</td>
</tr>
<tr>
<td>4.7</td>
<td>Algorithm Complexity</td>
<td>40</td>
</tr>
</tbody>
</table>

Chapter 5: Simulations                                           | 43   |
| 5.1     | Simulation Settings                                               | 43   |
| 5.2     | Evaluation Criteria                                               | 45   |
| 5.3     | Simulation Results by Cases                                       | 45   |
| 5.4     | Simulation Conclusion                                             | 64   |

Chapter 6: Conclusion and Future work                             | 66   |

REFERENCES                                                        | 69   |
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE 1. Example of Imbalanced Reservation on 5*5 mesh topology</td>
<td>21</td>
</tr>
</tbody>
</table>
# List of Figures

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1. Example of how MPLS works (a) in a network [29], where (b), (c) and (d) are Routing tables for R1, R2, R3, respectively.</td>
<td>10</td>
</tr>
<tr>
<td>Figure 2. A flow with a smoother.</td>
<td>13</td>
</tr>
<tr>
<td>Figure 3. LSS algorithm.</td>
<td>27</td>
</tr>
<tr>
<td>Figure 4. DAA algorithm.</td>
<td>31</td>
</tr>
<tr>
<td>Figure 5. General Balance Adjustment Algorithm.</td>
<td>36</td>
</tr>
<tr>
<td>Figure 6. Local Dynamic Adjustment Algorithm.</td>
<td>39</td>
</tr>
<tr>
<td>Figure 7. Topology 1.</td>
<td>44</td>
</tr>
<tr>
<td>Figure 8. Comparison of DAA over LSS on topology 1 and 5*5 mesh topology.</td>
<td>46</td>
</tr>
<tr>
<td>Figure 9. Improvement of DAA over LSS on topology 1 with hops=4.</td>
<td>47</td>
</tr>
<tr>
<td>Figure 10. Even and uneven topologies.</td>
<td>48</td>
</tr>
<tr>
<td>Figure 11. Improvement of DAA over LSS with hops=3 on (a) topology 2, (b) topology 3, (c) topology 4 and (d) topology 5, respectively.</td>
<td>48</td>
</tr>
<tr>
<td>Figure 12. Performance Comparison of DAA and LSS with and without LDA on (a) topology 1, (b) topology 3 and (c) topology 4.</td>
<td>50</td>
</tr>
<tr>
<td>Figure 13. Comparison of Improvement with different path length on (a) 5<em>5 mesh topology and (b) 8</em>8 mesh topology.</td>
<td>51</td>
</tr>
<tr>
<td>Figure 14. Improvement of DAA over LSS on topology 1 with (a) hops=5, (b) hops=4, (c) hops=3.</td>
<td>52</td>
</tr>
<tr>
<td>Figure 15. Topologies with heavy nodes.</td>
<td>53</td>
</tr>
</tbody>
</table>
Figure 16. Comparison of DAA over LSS with twice active heavy nodes on... 53
Figure 17. Comparison of DAA and LSS with heavy nodes on topology 1, Hops = 4.............................................................................................................. 54
Figure 18. Topologies with active sub-domains ........................................ 55
Figure 19. Comparison of DAA and LSS with bursty congestion on topology 6, hops=4.............................................................................................................. 56
Figure 20. Comparison of DAA and LSS with bursty congestion on modified topology 1, hops=4 ................................................................................ 56
Figure 21. Comparison of DAA and LSS with long congestion on topology 6, hops=4.............................................................................................................. 57
Figure 22. Comparison of DAA and LSS with long period congestion on modified topology 1, hops=4 ................................................................. 58
Figure 23. Comparison of DAA and LSS with two half-long period congestion on modified topology 6, hops=4 ........................................................................ 59
Figure 24. Topology 1 with heavy nodes and active sub-domains.............. 60
Figure 25. Comparison of DAA and LSS with twice active nodes and active sub-domains on topology 1, hops=4 ...................................................... 61
Figure 26. Comparison of DAA and LSS with three times active nodes and active sub-domains on topology 1, hops=4...................................................... 62
Figure 27. Comparison of DAA and LSS with four times active nodes and active sub-domains on topology 1, hops=4...................................................... 63
Chapter 1

Introduction

Delay and bandwidth guarantees are critical to many performance sensitive applications. To accommodate these applications, a network is required to provide performance guarantees, also known as Quality of Service (QoS). Many solutions have been proposed to support QoS either at a network level of selecting a proper route\(^1\) for each flow, or at a link level of ensuring QoS at each hop. Since a delay QoS requirement is additive among multiple hops along the route, how to partition it affects the system efficiency. However, there is limited research about partitioning the end-to-end delay requirement. Therefore, we investigate this delay allocation problem at a path level in order to maximize the overall network resource utilization in this project. Conventional approaches usually *statically* divide an end-to-end delay requirement into hop delay requirements. Such *static* allocations may cause imbalanced resource usage and "hot spots" in a network. To address this issue, we propose a dynamic allocation adjustment (DAA) approach to evenly spread traffic over paths and related links, by adjusting the existing reservations properly, to improve the system utilization. The simulation results

---

\(^1\) In this thesis, the term "route" is interchangeable to the term "path", representing a particular virtual connection with multiple hops, between a source and a destination.
show that DAA is able to admit 30% or more flows for a given network, compared with the best existing solution.

- What is the problem

Generally, we can divide internet applications into two categories: non-real-time and real-time. The former applications work well with the conventional best-effort service model without the guarantees of timely delivery of data. However, real-time applications such as Voice over IP (VoIP), Internet Protocol Television (IPTV), and industrial control systems, are sensitive to performance assurance, such as delay, loss rate or jitter. This implies that the network should be able to treat real-time packets differently from others, which is often said to support end-to-end quality of service [21]. There are two types of QoS: deterministic guarantee and statistical guarantee. A deterministic guarantee provides an absolute guarantee for certain performance parameters even in the worst case. On the other hand, a statistical guarantee allows minor violation of performance parameters in order to utilize the resource more efficiently.

Fine-grained and coarse-grained approaches are two broad types of approaches to support QoS. Fine-grained approaches such as Integrated Services (Intserv) [4][20] associated with Resource Reservation Protocol (RSVP) [5][17] provides QoS guarantees to individual applications, while coarse-grained approach such as Differentiated Services (Diffserv) [3][19] provides QoS to aggregate traffic. In this thesis, we use the term “flow” to loosely represent the traffic sessions requiring QoS: it could either be an individual
session in fine-grained approaches or an aggregated traffic session in coarse-grained approaches.

To provide the end-to-end QoS, the network service mechanism contains three major steps: QoS routing, Admission Control (AC), and Resource Allocation. A QoS routing mechanism determines the proper route for a flow. Given the route information, Admission Control checks whether the flow could be admitted in terms of network resources or other criteria. Then AC returns the admission result together with the corresponding route information, if the system decides to admit the flow. For each admitted flow, a Resource Allocation mechanism works on two levels to ensure an end-to-end QoS guarantee: a path-level resource allocation divides the end-to-end delay requirement into per-hop delay requirements, and assigns them to hops on the path; a hop-level resource allocation ensures the QoS on each intermediate hop with the use of certain single-hop scheduling scheme including GPS, WFQ [7][28] and EDF [23].

In this thesis, flows have the deterministic end-to-end delay requirements. Additionally, we assume that the path between the source and the destination of a flow is already uniquely chosen by routing algorithms. For instance, the path of a flow probably represents a set of IPTV sessions or a group of control/data channels for high-volume data exchange in financial networks or industrial control systems. Once a flow is admitted, its connection stays for a fairly long period, e.g., a connection between financial institutions may last eight hours or an entire working day. Therefore, it is
reasonable to consider each flow as a long-lived aggregated network connection, rather than an individual session.

Consequently, when a flow request arrives, an AC mechanism checks whether there is enough resource to accommodate it. If the flow is admitted, the service mechanism comes to the resource allocation part. As we see, the allocation on path level significantly impacts the hop-level bandwidth allocation. Thus, we focus our work on the resource allocation mechanism in partitioning the end-to-end delay requirement such that: the delay requirement can be satisfied at each hop and eventually along the path; the system can admit more long-lived traffic flows to get a higher resource utilization.

- What are existing techniques

Several path delay allocation schemes have been proposed, including Equal Allocation (EA) [25], Proportional Allocation (PA) [11], Load-based Slack Sharing (LSS) [14] and cost function based optimization algorithms. EA equally partitions the end-to-end delay requirement and PA proportionally partitions the end-to-end delay requirement in terms of the link loads. LSS defines “slack” to quantitatively measure the resource flexibility available in partitioning an end-to-end delay requirement and allocates the slack proportionally in terms of the link loads in order to minimize the deviation of the link loads. Another category of partitioning algorithms such as [9][21][22][30] try to minimize the global cost function as the sum of local costs which increases as QoS
requirements become tight.

- Why our solution

Those conventional schemes provide different ways to solve the partitioning problem of end-to-end delay requirements. Though each of them has its particular goal and own benefits, their static allocations is the key limitation of all these approaches. Once the flow gets admitted and the partition of end-to-end QoS is fixed, the per-hop QoS is determined and the router reserves the fixed amount of bandwidth, as long as the flow is alive. This may result in two issues: first, some schemes can not address the bottleneck link problem, in which one or more links are exhausted more quickly and become "dead" links; second, since the allocation is solely based on the instant system loads condition, request arrival patterns significantly affect the link reservations, and the allocations are unaware of future link loads. When overlapped requests arrive in a short period, the network may be partially congested. As a result, some incoming requests may be rejected while there are still sufficient resources to accommodate them.

To address this issue, we propose the Dynamic Allocation Adjustment (DAA) approach. DAA uses a general scheme to dynamically adjust the delay allocations of existing flows to balance system loads before the allocation of end-to-end delay requirements, in order to avoid the imbalanced reservations. It also provides a way to solve the bottleneck link problem.
This thesis gives contributions in the following aspects. First, we present a brief survey on the existing solutions about the problem of partitioning end-to-end delay requirements. Second, in order to increase the system utilization efficiency and to address the problems caused by the conventional schemes, a novel concept of dynamic adjustment to existing reservations of existing flows is introduced. Then, we propose an algorithm DAA with two separate approaches to address the potential problems. Besides the basic comparison tests, we consider several typical imbalanced input traffic and test our scheme on different types of topologies. Simulations results show that DAA outperforms existing solutions on both symmetric and asymmetric topologies, under either uniformly distributed traffic or imbalanced input traffic. The improvement is over 30%. With a large heterogeneity of link capacity, the improvement can be over 60%. Considering the complexity and computational costs, DAA is efficient in maximizing system utilization.

The remainder of this thesis is organized as follows. In Chapter 2, we first introduce the background knowledge of QoS models and QoS routing and then review the related work on per-hop and per-path resource allocation, respectively. In Chapter 3, we specify the network system, discuss the motivations, and present the idea of dynamic adjustments. Our scheme DAA will be introduced in Chapter 4, followed by simulation results in Chapter 5. The conclusion is presented in Chapter 6.
Chapter 2

Background

In this chapter, we will briefly review the background related to our work. First, we will introduce the common QoS models. Second, based on the study of QoS routing protocols, we address the assumption of MPLS implementation associated with proper routing protocols in our project. Furthermore, due to the inter-dependency, link level resource allocation including the regulation of traffic flows and service discipline will be reviewed, and we are able to get the explicit relationship between bandwidth reservation and delay bound. Moreover, several conventional resource allocation schemes at the path level will be introduced.

2.1 Typical QoS Models

Integrated Services (Intserv) [4][20] and Differentiated Services (Diffserv) [3][19] are two common types of architectures that support QoS. Intserv specifies the flow from two aspects. One is Traffic SPECification (TSPEC) used to regulate the traffic shape and the other one is Request SPECification (RSPEC) used to specify what the requirement is: either Guaranteed Service [33] which provides an absolute bounded service, or Controlled-load service [39] which is used for slightly loaded network and allows for
minor violations of requirements. RSVP as in [5][17], initially designed for Intserv, is a network layer protocol that enables the application to obtain QoS. It maintains soft states of each resource reservation, hence supporting dynamic adaptation to network changes. Therefore, Intserv/RSVP model is able to provide QoS guarantee on a per-flow basis. However, it suffers from heavy costs in storage and management of states and specifications of flows and requires every router to support it.

To overcome the scalability problem of Intserv, Diffserv [3][19] classifies individual flows and aggregates them into a finite number of traffic classes and provide QoS at the class-based granularity. Diffserv is more scalable because the states maintenance only deals with a finite number of classes; all the classification and marking work is done at the boundaries between domains and the rest of routers only work on forwarding packets. However, the shortcomings of Diffserv is that it does not provide the absolute guarantees to individual flow and the agreement of classification of traffic among multiple domains is hard to get, then some solutions such as [1] extends RSVP to Diffserv network as well.

2.2 QoS Routing

As introduced, RSVP is not a routing protocol but works with current protocols including two main classes of routing protocols: distance vector and link state protocols. Routing Information Protocol (RIP) [16] is a widely used distance-vector based routing protocol in IP network. Open Shortest Path First (OSPF) [24] is another common used
link-state routing protocol which contains two steps: reliable flooding and route calculation. IETF also extends RSVP to RSVP-TE [1], where TE stands for Traffic Engineering. RSVP-TE sets up the Label Switched Path (LSP), which contains the information about network resource parameters such as available bandwidth and explicit links. Besides those conventional routing protocols initially designed for best-effort networks, many QoS routing protocols have been developed such as improved shortest path scheme like Widest Shortest Path (WSP) algorithm [15], flooding based algorithms [18][34] and some bandwidth-delay constrained route selection algorithms like [6][37].

In my thesis, we assume the routes are already chosen and the flows are considered to be LSPs [8]. The routing part is supported by Multi Protocol Label Switching (MPLS) [32]. MPLS is used to provide high-performance, multi-service switching in packet data networks. Destination-based forwarding and explicit routing are two main features of MPLS.

With MPLS, the routers forward the packets based on the prefixes instead of normal IP address. As showed in Figure 1, the number associated with each link is used to specify the routes/interfaces. H1 and H2 are two destination hosts with prefixes 1.0.0/3 and 1.3.3/6. R3 and R4 are two routers connected to H1 and H2 respectively; R1 and R2 have their routing tables. Consider the packets firstly entering R1 with the destination H1, R1 is called Label Edge Router (LER) and it adds the label 2 to the packets after the lookup of the routing table and a longest match search for the prefix. When the packets
arrive at R2, based on the label 2, R2 switches the label to 3 and forwards them to R3.

Similar switching from label 2 to 3 occurs at router R3 as well until the packets finally arrive at H1. Therefore, destination-based forwarding enables the implementation of Virtual Private Network other than actual IP routing network; explicit routing solves the routing issue.

![Diagram of MPLS operation]

**Figure 1.** Example of how MPLS works (a) in a network [29], where (b), (c) and (d) are Routing tables for R1, R2, R3, respectively.

In my thesis, MPLS is assumed to be implemented and the routers are considered to be either LER or Label Switching Routers (LSRs) that the route will be determined at LER and all the intermediate routers only do the forwarding packets in terms of the prefixes.
2.3 Resource Allocation

2.2.1 Per-Hop Resource Allocation

- Traffic regulator and Service discipline

To derive the relationship between an upper delay bound and the corresponding reserved bandwidth, we assume that all the traffic traverses a traffic regulator at an ingress node before it enters the network. One popular traffic regulator is leaky bucket. A flow $i$ after the token bucket can be represented with a traffic envelope $(\sigma(i), \rho_{avg}(i))$, where $\sigma(i)$, $\rho_{avg}(i)$ are the maximum burst size and the long term average rate, respectively, such that during any interval of length $t$, the number of arrival bits in that interval is less than $(\sigma(i) + \rho_{avg}(i) \cdot t)$.

The link service discipline can be classified as either work-conserving or non-work-conserving. In my thesis, we use work-conserving service discipline which means the server is never idle as long as there is a packet to send. One common rate-based discipline is Generalized Processor Sharing (GPS) [28], which provides an efficient and flexible way to manage the multiplexed flows sharing a link. For each flow $F_i$ that shares the link $l$, let $\phi_l(i)$ denote the weight assigned to it and $W_l(i; \tau)$ denote the amount of traffic serviced in time interval $\tau$ respectively. In every round, a server serves each flow in proportion to its weight. Then for any two flows, GPS simply assumes that all the traffic flows are fluid flows $F_i$ and $F_j$ sharing link $l$, the following relation holds.
Although studies have shown that many analyses can be derived based on the GPS model, actual flows, are not fluid but consist of discrete packets. Weighted Fair Queuing (WFQ) [28] is a good approximation of GPS for packetized flows. WFQ is a generalization of Fair Queuing (FQ) [7] that each data flow has a separate FIFO queue with its own service weight, which is similar to the weight in GPS. It has been proven that when the WFQ is used and the incoming flows comply with the token bucket model, its end-to-end delay bound even in the worst-case could be guaranteed. Suppose at link \( l \), the flow \( F_i \) is token bucket constrained with the parameters \((\sigma_l(i), \rho_{avg}(i))\), where \( \sigma_l(i) \) and \( \rho_{avg}(i) \) are its maximum burst size and average flow rate respectively, \( \rho_l(i) \) is the actual bandwidth reservation at link \( l \), \( C_l \) is the link capacity and \( L_{\text{max}} \) is the largest packet size, then [28] shows that the worst-case queuing delay \( D_l(i) \) has the upper bound as follows.

\[
D_l(i) = \frac{\sigma_l(i)}{\rho_l(i)} + \frac{L_{\text{max}}}{C_l} + \frac{L_{\text{max}}}{\rho_l(i)}
\]

(2)

where the first component of the delay is fair-queuing delay, the second component is the packetization delay and the last component is router’s transmitting delay.

Therefore, based on the above introduction, we get the explicit relationship between the worst-case delay bound and the amount of reserved resource at one link, which helps us to deal with the local path level of resource allocation later.
Implementation of Smoother

To simplify the analysis of flow bursts and queuing delays, we also assume that, as shown in Figure 2, a smoother at a source node is used to simplify the estimation of an upper bound of queuing delay. A flow is smoothed at its ingress point of the network, and then serviced by rate-based local schedulers at intermediate hops [12][29]. We adopt this idea in our system. The smoother eases the burstiness of a flow at its minimum rate on the path, which is $\rho_{min}(i) = \min\{\rho_l(i)\}$ [12] and $\rho_l(i)$ is the reserved bandwidth on link $l$ along the path. Since on each intermediate hop, we have to reserve bandwidth larger than the flow's average rate, which ensures that $\rho_l(i) > \rho_{avg}(i)$ for any link $l$; otherwise, the queue may become infinitely large in the worse case. Thus the router scheduler's service rate is no smaller than the flow's average rate and the flow after the smoother will not result in any burst. In that case, we can consider the flow having a constant rate. Accordingly, the smoothing delay is determined as follows. $\sigma(l)/\rho_{min}(l)$.

$$D_{\text{smoothing}}(i) = \frac{\sigma(i)}{\rho_{\text{min}}(i)}$$  \hspace{1cm} (3)

Resource-Delay Relationship

At each hop, we use WFQ as a local scheduler to ensure local delays according to the
previous discussion, then the worst-case queuing delay at link $l$ of flow $i$, denoted as $D_l(i)$, is given by (2.3). Because we smooth the flow at the ingress router, $\sigma_l(i) = 0$. Therefore, the worst queuing delay at link $l$ becomes $D_l(i) = L_{\text{max}}/\rho_l(i) + L_{\text{max}}/C_l$. As shown in [26][27], since the delay bounds at each hop of the path are additive, the end-to-end delay bound for flow $i$, denoted as $D(i)$, is the sum of worst queuing delay of each link plus the smoothing delay incurred at the smoother, denoted as

$$D(i) = \sigma_l(i) / \rho_{\text{min}}(i) + \sum_{l=1}^{k} (L_{\text{max}}/\rho_l(i) + L_{\text{max}}/C_l)$$

(4)

where the first term is the smoothing delay, $\rho_{\text{min}}(i)$ is the minimum bandwidth reserved along the path, and $k$ is the total number of links on the path. We name the link with the minimum bandwidth reservation as the critical link of the flow.

### 2.2.2 Per-Path Resource Allocation

- **Equal Allocation and Proportional Allocation**

  Equal Allocation (EA) [25] is a straightforward approach that partitions the end-to-end delay requirement equally among all the intermediate links. Proportional Allocation (PA) [11] tries to balance the loads over hops on a flow path by dividing an end-to-end (delay) requirement proportionally to the current load of each constituent link. EA is very easy to implement but it ignores any local information about links’ available resource and does not perform well when flows have more hops. PA outperforms EA because it
considers current link loads. Suppose the local QoS requirement is \( D_l(i) \) for flow \( i \) at link \( l \), additionally, for flow \( i \), \( U_l(i) \) denotes the load ratio at link \( l \) and \( \mathcal{P}(i) \) denotes the set of all links along the path of flow \( i \), then PA partitions the end-to-end QoS requirement \( D(i) \) as follows [11].

\[
D_l(i) = D(i) \cdot \frac{U_l^+(i)}{\sum_{s \in \mathcal{P}(i)} U_s^+(i)}
\]  

(5)

where \( U_l^+(i) = \max(U_l(i), a) \) and \( a \) is a small constant, such as 0.001, in order to ensure that the local delay requirement is positive and finite.

However, PA has a major drawback that it does not protect those links that already have heavy loads, having the risk of being exhausted of resource. For instance, if an estimated local delay requirement is smaller than the minimum delay \( d \) that a link \( j \) can support, PA simply assigns \( d \) to the link as its local delay requirement, and partitions the rest of end-to-end delay requirement to other hops proportionally to other links on the path. Because this allocation uses up to all available bandwidth of the link \( j \), it becomes a bottleneck link: no flow in future is able to pass through it until any existing flow terminates and the reservation resource is released.

- **Cost function based Solutions**

Many solutions such as [9][21][22][30] have been proposed to optimally solve the resource allocation problem. At each link, a cost function, which increases as the QoS requirement gets tight, is associated. The global cost function is simply the sum of local
costs. Then, the problem becomes to partition the end-to-end QoS requirement in the way that the global cost is minimized. Some polynomial solutions are presented such as OPQ in [22]. However, there is no explicit relationship between the cost function and the actual performance. The problems of how to define a cost function and lacking a general measuring criterion for the whole network still remain.

- Load-based Slack Sharing algorithm

Different from PA, which directly partitions the end-to-end delay requirement based on current link loads, Load-based Slack Sharing (LSS) [14] uses a heuristic to minimize the deviation of link loads to get higher utilization. LSS defines a slack in an end-to-end performance requirement to quantify the flexibility in balancing link loads over a path. A minimum local delay budget $D_{li}^{\text{min}}$ at a hop $l$ is defined as the delay if all residual bandwidth of the link is allocated to service the new flow $i$. Then, an end-to-end slack on the path is defined as $\Delta D_i = D_i - \sum_{l=1}^{k} D_{li}^{\text{min}}$, where $D_i$ is the end-to-end delay requirement of the flow, $1<i<k$, and $k$ is the total number of hops on the path. As long as the slack is positive, it means that the path is able to guarantee a tighter end-to-end delay bound. In other words, for the current flow $i$, there will be some flexibility. As we know, if we distribute some amount of this slack to any link $l$, that link will get a looser local delay bound which is larger than $D_{li}^{\text{min}}$ and thus can save some bandwidth. Therefore, LSS allocates slack to all the links along the path based on load, with the purpose of
minimizing the loads deviation, to get the higher utilization. Moreover, since any individual link along the path will get an amount of slack, LSS avoids the problem of bottleneck links.
Chapter 3

Problem Formulation

Based on the introduction in last chapter, we will summarize the topic, present the motivations, and finally come up with the adjustment idea in this chapter. In section 3.1, we focus our work in partitioning the end-to-end deterministic delay QoS for Unicast and long-lived flows. Then in section 3.2, we compare the performance of conventional schemes to address the imbalanced reservation problem caused by static reservations. Moreover, we put forward the expectations for the partitioning scheme, and briefly introduce the concept of adjustment and our dynamic allocation algorithm.

3.1 Problem to be addressed

We consider a network with the link set $\mathcal{L}$, incoming request set $F$ and a set of routers $R$. The objective is to avoid imbalanced reservation through adjustments in order to allocate the resource more efficiently and assure all the admitted flows meet their end-to-end QoS guarantees. Before introducing our scheme, we summarize and specify the characteristics of this problem as follows.

(a) Deterministic Delay QoS: in my thesis, we only deal with deterministic delays that a network should provide absolute guarantees to end-to-end delay requirements.
Meanwhile, delay requirement is additive so that how to partition it to local delay bounds becomes the focus of our project.

(b) **Resource allocation focus**: as introduced earlier, our framework only performs basic admission control and it focuses on resource allocation. The routing is supported by pre-fixed MPLS with proper routing protocols. Thus, for each flow, a route is determined *a priori* and is used as given information.

(c) **Unicast and Aggregated long-lived flows**: in my thesis, each flow has exactly one source and one destination; a flow is referred to an aggregated traffic session of a number of homologous applications which share the same route; meanwhile, the lifetime of the flow could be several hours, which is considered to be fairly long, compared to the interarrival time such as couple of minutes, thus, we consider “static” requests that once the flow arrives and get admitted, it stays there for a fairly long time.

(d) **Sequential arrival of traffic**: request flows arrive in succession, and we perform the admission control and resource allocation one by one.

(e) **Up-to-date knowledge**: For each new request, we are fully aware of the current situation of the network and the existing reservations, but have no prediction about traffic in future.

(f) **Evaluation criterion**: since we consider “static” reservations, generally, the more flows from request set $F$ get admitted, the higher the resource utilization efficiency is.
3.2 Motivations

Given the above description, we are interested in how conventional schemes work. Based on the introduction in chapter 2, we briefly summarize their performance as follows. EA is the easiest to implement but it does not take link load information into consideration. PA tracks the current load situation in each round and proportionally partitions the end-to-end delay requirement based on link loads that it has a clear local goal for each admitted flow. However, PA lacks a concrete global goal, and it does not care about the remaining bandwidth of links that it easily exhausts all available bandwidth of a link and generate bottleneck links. OPQ [22] is the typical algorithm based on a clear global goal as minimizing the global cost. However, there is no explicit clue about how to define a good cost function and the relationship between the global cost and global goal is not clear. As showed in [14], LSS outperforms the above schemes because it has a clear goal which is to partition the end-to-end delay requirement as well as the flexibility such that the deviation in the loads on related links is as small as possible. After each partitioning by LSS, the links along the request flow have the same loads and thus eventually the links in the network are more likely to have similar loads.

In summary, all the conventional schemes are “current” and “static”. “Current” means an allocation is made based on that particular and local situation. No matter what scheme is employed or what local optimization goal is set, allocation may not be optimal since it is unaware of future input situations. “Static” means once the allocation is done,
the reservation will not be changed until the flow terminates, regardless of how improper it is upon the situation later.

Therefore, since the flow arrives sequentially and we do the allocation one by one, "static" conventional schemes might result in some problems.

(1) *Imbalanced Reservation.* When a series of overlapped requests arrive, LSS may also result in uneven reservations as the dramatic uneven bandwidth reservations as shown in TABLE 1. It shows an example of allocations by LSS for flows with a length of six hops on a 5*5 mesh, with an average request rate of 100kbps, a burst size of 5kbits, the largest packet size of 1kbits and link capacity of 5Mbps. The first two flows have equal allocations and each link reserves 193 kbps. However, flow 3 and flow 4 show significant differences among link reservations: 186 kbps vs. 1182 kbps for flow 3, and 172 kbps vs. 693 kbps for flow 4. Such uneven reservations obviously consume much more bandwidth as showed in the last column.

<table>
<thead>
<tr>
<th>Flow #</th>
<th>Reservation per hop (kbps)</th>
<th>hop 1 (kbps)</th>
<th>hop 2 (kbps)</th>
<th>hop 3 (kbps)</th>
<th>hop 4 (kbps)</th>
<th>hop 5 (kbps)</th>
<th>hop 6 (kbps)</th>
<th>sum of reservation (kbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>193</td>
<td>193</td>
<td>193</td>
<td>193</td>
<td>193</td>
<td>193</td>
<td>193</td>
<td>1158</td>
</tr>
<tr>
<td>2</td>
<td>193</td>
<td>193</td>
<td>193</td>
<td>193</td>
<td>193</td>
<td>193</td>
<td>193</td>
<td>1158</td>
</tr>
<tr>
<td>3</td>
<td>186</td>
<td>186</td>
<td>186</td>
<td>1182</td>
<td>622</td>
<td>622</td>
<td>2984</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>172</td>
<td>172</td>
<td>172</td>
<td>172</td>
<td>314</td>
<td>693</td>
<td>1695</td>
<td></td>
</tr>
</tbody>
</table>

(2) *Bottleneck Dead link and request rejection ratio.* With existing static allocations, if most of intermediate hops through the route have sufficient bandwidth but only one link
is exhausted, we have to reject that request. Even worse, if a link’s available bandwidth is smaller than a given threshold, say, the average rate of request flow, that link is considered dead and all future requests passing link will be rejected. To address this issue while maintaining high system utilization, we propose to adjust the existing reservations of previously admitted flows on a dead link to alleviate the bottleneck problem and accommodate incoming flows.

3.3 Overview of Scheme & Adjustment

We investigate a novel dynamic allocation scheme that meets the following requirements.

(a) Clear global goal: the scheme should have a clear global goal—to maximize the overall resource utilization, which is to accommodate more incoming flows.

(b) Fully automatic: the allocation algorithm is applicable to general topologies without any manual specification of cost function, etc.

(c) Able to modify reservations: the scheme is able to adjust the previous imbalanced reservation or avoid the bottleneck link problem in order to maximize the utilization of the network.

LSS is a good candidate to meet the requirements (a) and (b). Thus in our scheme, we adopt the idea of LSS to partition the slack to all the links along the route. Additionally, we bring in a new idea of “adjustment” into our algorithm that it provides the way to
change the existing reservation before we partition the end-to-end QoS.

We discussed several dynamic adjustment schemes in this project. Two key differences between the proposed approaches and the existing approaches are: (1) we consider not only to partition the requirement of a new flow over the links on its path to balance link loads (as the previous approaches), but also to adjust existing allocations on the links to further distribute link loads more evenly. (2) We consider adjusting not only the link loads along the path of the new flow, but also the link loads of previous flows, which are partially overlapped with the new flow. Solely based on the current status of links on a flow path, PA directly allocates static reservations on the path and hopes to achieve balance eventually, while LSS tried make residual link bandwidth on the path similar. DAA integrates their goals, evenly balances link loads, smoothes bottlenecks, while also makes their residual bandwidth similar at the same time. Consequently, DAA is able to deal with severe imbalanced cases, quickly balance link loads, and significantly improve system utilization. Apparently, dynamic adjustments involve more costs. If it is not done carefully, the performance gains may be overwhelmed by the cost. We will address this challenge and develop efficient and effective dynamic adjustment approaches in next chapter.
Chapter 4

Dynamic Allocation Adjustment

Dynamic Allocation Adjustment (DAA) scheme will be introduced in this chapter. DAA adopts the idea of LSS to allocate slack and bring in a novel way to adjust the existing reservations. Some important questions are addressed such as when and how to do the adjustments. DAA gives preliminary thought that adjustments will not violate the requirements of existing flows and it helps to achieve the target: to increase the utilization efficiency. DAA performs in two ways: one general way is to alleviate the imbalance loads along the path to reduce the total bandwidth reservation for that flow; another way is to address the bottleneck link problem.

4.1 Definitions

4.1.1 Flow Specification

Each flow is considered to be a unicast and aggregate flow of several traffic sessions. For flow $i$, a traffic vector $(\sigma(i), \rho_{avg}(i), L_{max}(i), k, D(i), S(i), Dest(i))$ is used to denote its parameters where $\sigma(i)$ is its burst size, $\rho_{avg}(i)$ is the average flow rate, $L_{max}(i)$ is the largest packet size, $k$ is the number of hops, $D(i)$ is the end-to-end delay requirement, and $S(i)$ and $Dest(i)$ are the source and destination addresses, respectively. Before the flow
enters the network, it is regulated by the smoother and the potential burst will be smoothed. Then the flow becomes a constant rate flow with an incurring smoothing delay. Consequently, the traffic vector becomes as \((\rho_{\text{avg}}(i), D_{\text{smoothing}}(i), L_{\text{max}}(i), k, D(i), S(i), Dest(i))\), where \(D_{\text{smoothing}}(i)\) is the smoothing delay, as introduced in section 2.2.1.

### 4.1.2 Routing and Admission Control

As we have discussed earlier, we assume that all the routers are supported by MPLS associated with some QoS routing protocols. For flow \(i\), the path connecting source \(S(i)\) and destination \(Dest(i)\) is selected and unique, denoted as \(Path(i)\). Meanwhile, the Admission Control mechanism has to be specified. The first and basic condition of AC is that, each link \(l \in Path(i)\) must have an available bandwidth larger than \(\rho_{\text{avg}}(i)\), the average rate of the flow. Let \(C_l\) and \(L_l\) denote the capacity and already reserved/allocated bandwidth for flow \(i\) respectively.

\[
C_l - L_l > \rho_{\text{avg}}(i) \quad \text{where} \quad l \in Path(i)
\]  

(6)

Then, we will check whether the links along the path have enough bandwidth to meet the end-to-end delay requirement. Since the delay bound along the path with multiple hops is additive, we address the condition as follows.

\[
D_{\text{smoothing}}(i) + \sum_{l=1}^{k} D_l(C_l - L_l) \leq D(i) \quad \text{where} \quad l \in Path(i)
\]  

(7)

In the above condition, \((C_l - L_l)\) represents all of the available bandwidth and the function \(D_l()\) is the hop delay bound as introduced earlier. Remember from chapter 2 that
we define the flexibility of available bandwidth as a slack and

$$D_{\text{smoothing}}(i) = \sigma(i)/\rho_{\text{min}}(i) = \sigma(i)/\min_i(C_i - L_i)$$, we have

$$\text{slack} = D(i) - D_{\text{smoothing}}(i) - \sum_{l=1}^{k} D_l (C_l - L_l) = D(i) - \frac{\sigma(i)}{\min_l (C_l - L_l)} - \sum_{l=1}^{k} D_l (C_l - L_l)$$

where \( l \in \text{Path}(i) \) (8)

Thus, the flow would be admitted if slack is positive which means a positive available flexibility.

4.2 LSS Review

4.2.1 Smoothing Delay and Delay Allocation

While in Admission Control procedure, we can exactly calculate slacks because we assume that each link reserves all its available bandwidth such that a slack is determined by the link with smallest available bandwidth. However, if the flow gets admitted and when we start to run the allocation scheme to partition the end-to-end delay, since the delay allocation has not yet solved and the link with least actual reservation is unknown, the smoothing delay could not be calculated explicitly. Therefore, the smoothing delay and the partition of delay requirement are dependent: the partition of delay affects the link with the smallest reservation which determines the smoothing delay.
4.2.2 Review of LSS algorithm

To maximize the efficiency of the network, which is represented as the number of flows admitted in future, LSS employs a good heuristic to allocate the slack such that the load across each intermediate link remains balanced. That is, LSS tries to minimize the load variation after admitting each new flow. Because of the dependency between smoothing delay and delay allocation, LSS uses a dichotomy algorithm to get a delay partition as showed in Figure 3.

\begin{verbatim}
LSS()
1. b=0.5
2. n=50
3. while(n ≤ 50)
   4. for each link l∈Path(i)
      5. \( p_1(i) = \max\{ (C_l - L_l - \beta * C_l * b), p_{avg}(i) \} \)
      6. \( D_1(i) = D_1(p_1(i)) \)
      7. \( D_{\text{smoothing}}(i) = \sigma(i)/\min_1(p_1(i)) \)
      8. \( \text{slack}=D(i) - D_{\text{smoothing}}(i) - \sum D_1(i) \)
      9. if \( 0 ≤ \text{slack} ≤ D_{\text{threshold}}, \) return \( D_1(i), b \) and terminates
     10. if \( \text{slack} > 0, \) \( b=b+b/2 \)
     11. else \( b=b-b/2 \)
\end{verbatim}

Figure 3. LSS algorithm.

This dichotomy algorithm starts with a stepping parameter \( b \) equals 0.5. Since value of \( b \) converges quickly in simulations, we set a maximum number of iterations as 50. Because LSS tries to balance the link loads after partitioning, each link has a remaining bandwidth \( \beta * C_l * b \), where the value of \( \beta \) is set to be 1 when the optimization objective is to ensure that a link’s remaining capacity is proportional to its capacity. Thus, \( b \) becomes the residual link load after each partitioning. As long as each link has the available bandwidth
larger than the average flow rate indicated by line 5, all the intermediate links would have a same load afterwards. In terms of the value of \( b \) in iteration, the link with smallest reserved bandwidth is found and the smoothing delay is calculated in line 7. Because in line 5, we do not use all the available bandwidth at each link and leave some flexibility, the corresponding slack value calculated in line 8 is actually a *difference slack* of the total slack minus the sum of allocated slack. If this slack is negative when the sum of allocated slack is larger than the total slack that the flow could provide, it means that the end-to-end delay requirement can not be met and more bandwidth at each link has to be reserved, then we reduce the value of \( b \) and repeat the iterations. LSS terminates only if the slack is positive and less than a threshold or is positive after the maximum number of iterations. In simulations, since this dichotomy algorithm converges quickly, a proper \( n \) such as 50 performs well. Otherwise, either the request will be rejected or a larger threshold will be set. The smaller the value of \( DThreshold \), the closer LSS can get to the optimization objective that the slack has been allocated properly across those links, and the resulting load variation is minimized.

4.3 DAA Overview

4.3.1 Global Goal

Reservations made by LSS and other conventional schemes are static based on
current situation. The reason that LSS outperforms than other schemes is that it allocates slack in such a manner that the resulting load variation of intermediate links is minimized.

DAA algorithm not only ensures balanced loads along the path of a flow after each partitioning, but also provides a way to adjust the existing reservations to balance the loads of some selected links before each partitioning, in order to reduce the reservation costs for each flow, and therefore increase the total network resource usage efficiency. In a word, the first goal is to balance the link loads of related links to avoid imbalanced bandwidth reservation as indicated in chapter 3. Generally, the smaller load variation, the less bandwidth reservation costs. The main reason is that the smoothing delay is solely determined by the link with smallest reserved bandwidth, thus reserving more bandwidth at that hop can reduce the smoothing delay so that the sum of reserved bandwidth can be greatly reduced. We propose a function as General Balance Adjustment (GBA) function to achieve this goal. Additionally, once the bandwidth of a link is less than some values such as the average flow rate, it becomes the bottleneck link that any flows in future going across that link will be rejected. Thus, our algorithm DAA uses another function Local Dynamic Adjustment (LDA) to solve this problem.

4.3.2 Critical Link

We name the link with the minimum bandwidth reservation as the critical link of a flow. Recall that the queuing delay for each link \( l \) is determined by the function
\[ D_i(i) = L_{\text{max}}/\rho_i(i) + L_{\text{max}}/C_i \] from (2) when the burst \( \sigma(i) \) has been erased to be zero. Suppose link \( i \) is the critical link which brings in the smoothing delay \( \sigma(i)/\rho_i(i) \) from (3), we can easily see that the smoothing delay should be taken into consideration as long as the burst size is comparable to the largest packet size. Further, the burst size is larger than the largest packet size, and the smoothing delay will be significantly large. Therefore, if we want to reduce the total bandwidth reservation of all intermediate links along the flow, the critical link(s) will be a good candidate(s).

### 4.3.3 DAA Algorithm

DAA is designed to address imbalanced link reservations and remove bottleneck links to achieve high system efficiency, by adjusting existing reservations without violating their end-to-end delay requirements, to evenly distribute loads not only on flow paths but also on other related links. For a given network, we have a set of routers \( R \) and a set of links, \( E \); for a link \( l \in E \), it has a link capacity of \( C_l \) bps. We use \( F \) to denote a request set. DAA processes each request as follows in Figure 4.
DAA Algorithm \((F,E,R)\)

1. For each flow request \(i \in F\)
2.    Admission Test();
3.    if need_balance_path
4.        General_Balance_Adjustment();
5.    if admitted
6.        Allocation based on its slack;
7.    else // not admitted
8.        Local_Dynamic_Adjustment();
9.    if (can accommodate the request)
10.       Adjust existing reservation;
11.       Partition its path delay requirement;
12.    else // still can not accommodate
13.       Reject the request;

Figure 4. DAA algorithm.

GBA is to balance the link loads of the request and related links, and address imbalanced reservations. GBA is called if its condition, which is specified later, is met. When the request is admitted, DAA partitions the end-to-end delay into local delay requirements based on its slack. If the request is rejected, it means that there exist bottleneck links; then, LDA is applied. LDA firstly checks if adjustment will help accommodate the request. If so, adjustments are conducted; then the request is admitted, and its path delay is partitioned and allocated to hops. Otherwise, the request is rejected.

4.4 Important Issues

4.4.1 How often to adjust

Since GBA is to balance loads for related links, GBA is currently called when the load variation of request links meets the condition which will be introduced later. LDA is
called only if an incoming flow has been rejected.

4.4.2 How to choose links and associated flows

Since DAA performs link-by-link adjustments for a selected set of related links, and the bandwidth reservations at one link impacts the reservations at other links, we need carefully to determine which links to adjust in order to control adjustment costs. Suppose link $I$ is the target link that we want to either increase or decrease its load, we have to figure out which link and its corresponding existing flow reservation to adjust.

First, we determine the set of qualified and related links, denoted as $\mathcal{P}(I)$ via a procedure called $\text{Find\_Link\_Coverage()}$. Currently, in terms of computational complexity, we only select the links that belong to an existing flow going across link $I$ and do not consider other links in the network. That is, we will pick link $k$ only if there is at least one existing flow going across both of link $I$ and $k$. There are several methods to select $\mathcal{P}(I)$: (a) selecting all other links excluding link $I$ on the request flow path and their related links, (b) selecting only related links but excluding the links on the request path, or (c) selecting other links on the flow path only.

For instance, considering a target link $I$, when the objective is to lower its load to distribute extra delays to other links, we determine $\mathcal{P}(I)$ by selecting links not overlapped with the current request, because it will not make other links for the same request become bottlenecks. When the objective is to raise the load of link $I$ to absorb extra delays from
other links, we determine $\mathcal{F}(l)$ by selecting links that host the same flows as link $l$. This helps lower the link loads of the current request, and balance global reservations.

Since each link is only adjusted once, we have to pick up one existing flow for link $k$, when there are multiple flows going across link $l$ and $k$. The simple way is to always randomly choose one; or we can choose the flow on which link $k$ has the smallest reservation load. Because with the smallest reservation load, link $k$ is more likely to be a critical link on that flow, thus the adjustment is thought to be more effective to a critical link since it brings in a large smoothing delay.

4.4.3 How adjustment works

The larger bandwidth reserved at a link, the smaller delay bound is at this hop; and vice versa. Suppose we need to adjust link $j$ of flow $i$. After the delay allocation, the delay bounds for flow $i$ should meet the end-to-end delay bound $D(i)$. Let $S$ denote the set of all links of flow $i$ excluding link $j$. We rewrite (2.3) as follows.

$$D(i) = \sigma(i) / \rho_{\text{min}}(i) + L \max / \rho_j(i) + L \max / C_j + \sum_{s \in S} (L \max / \rho_s(i) + L \max / C_s)$$

(9)

Given the earlier bandwidth reservation vector $\rho(i)$ of the hops of the flow $i$, and the amount of bandwidth adjustment $\Delta\rho_j(i)$, we derive a function $f()$ to decide the corresponding adjustment of delay bound, denoted as $D'(i)$, and $D'(i) = f(\rho(i), j, \Delta\rho_j(i))$. Since the adjustment may affect which link will determine the smoothing delay, for the
reservation vector \( \rho(i) \), we use \( \rho_{\text{min}}(i) \) and \( \rho_{2\text{nd}\_\text{min}}(i) \) denote the minimum and the second minimum bandwidth reserved along the flow \( i \), respectively.

(1) When \( \Delta \rho_j(i) > 0 \), we increase the bandwidth reserved for link \( j \). Comparing the new reservation bandwidth \( (\rho_j(i) + \Delta \rho_j(i)) \) with \( \rho_{\text{min}}(i) \) and \( \rho_{2\text{nd}\_\text{min}}(i) \), we have three cases. First, if \( \rho_j(i) > \rho_{\text{min}}(i) \), then \( (\rho_j(i) + \Delta \rho_j(i)) > \rho_{\text{min}}(i) \). It means link \( j \) is not the link that determines smoothing delay. We have:

\[
D(i) = \sigma(i) / \rho_{\text{min}}(i) + L \max(\rho_j(i) + \Delta \rho_j(i)) + L \max(C_j) + \sum_{k=0}^{\text{slots}} (L \max(\rho_k(i)) + L \max(C_k)) + D'(i) 
\]  

(10)

Second, if \( \rho_j(i) = \rho_{\text{min}}(i) \) and \( (\rho_j(i) + \Delta \rho_j(i)) > \rho_{2\text{nd}\_\text{min}}(i) \), it means link \( j \) is no longer the link with minimum reserved bandwidth. Then we have:

\[
D(i) = \sigma(i) / \rho_{2\text{nd}\_\text{min}}(i) + L \max(\rho_j(i) + \Delta \rho_j(i)) + L \max(C_j) + \sum_{k=0}^{\text{slots}} (L \max(\rho_k(i)) + L \max(C_k)) + D'(i) 
\]  

(11)

Third, if \( \rho_j(i) = \rho_{\text{min}}(i) \) and \( (\rho_j(i) + \Delta \rho_j(i)) < \rho_{2\text{nd}\_\text{min}}(i) \), it means even \( \Delta \rho_j(i) \) more bandwidth has reserved, link \( j \) is still the link with minimum reserved bandwidth. Then we have:

\[
D(i) = \sigma(i) / (\rho_j(i) + \Delta \rho_j(i)) + L \max(\rho_j(i) + \Delta \rho_j(i)) + L \max(C_j) + \sum_{k=0}^{\text{slots}} (L \max(\rho_k(i)) + L \max(C_k)) + D'(i) 
\]  

(12)

Consider these three cases with its initial allocation, we have:
\[ D'(i) = f(\rho(i), j, \Delta \rho_j(i)) \]

\[
D'(i) = \begin{cases} 
L \max / \rho_j(i) - L \max / (\rho_j(i) + \Delta \rho_j(i)) & \text{if } \rho_j(i) > \rho_{\min}(i) \\
\sigma / \rho_{\min}(i) + L \max / \rho_j(i) - \sigma / (\rho_j(i) + \Delta \rho_j(i)) - L \max / (\rho_j(i) + \Delta \rho_j(i)) & \text{if } \rho_j(i) = \rho_{\min}(i) \text{ and } \rho_j(i) + \Delta \rho_j(i) > \rho_{2nd_{-\min}}(i) \\
\sigma / \rho_{\min}(i) + L \max / \rho_j(i) - \sigma / \rho_{2nd_{-\min}}(i) - L \max / (\rho_j(i) + \Delta \rho_j(i)) & \text{if } \rho_j(i) = \rho_{\min}(i) \text{ and } \rho_j(i) + \Delta \rho_j(i) < \rho_{2nd_{-\min}}(i) 
\end{cases}
\] (13)

(2) When \( \Delta \rho_j(i) < 0 \), we reduce the bandwidth reserved. We also have three cases and the function \( f() \) is defined as follows.

\[ D'(i) = f(\rho(i), j, \Delta \rho_j(i)) \]

\[
D'(i) = \begin{cases} 
(\sigma + L \max \rho_j(i) - (\sigma + L \max)(\rho_j(i) + \Delta \rho_j(i)) & \text{if } \rho_j(i) = \rho_{\min}(i) \\
L \max / (\rho_j(i) + \Delta \rho_j(i)) - L \max / \rho_j(i) & \text{if } (\rho_j(i) + \Delta \rho_j(i)) > \rho_{\min}(i) \\
(\sigma + L \max \rho_j(i) + \Delta \rho_j(i)) - \sigma / \rho_{\min}(i) - L \max / \rho_j(i) & \text{if } (\rho_j(i) + \Delta \rho_j(i)) < \rho_{\min}(i) 
\end{cases}
\] (14)

Notice that function \( f() \) is piecewise linear, we can find its inverse function \( \Delta \rho_j(i) = f^{-1}(\rho(i), j, D'(i)) \), for different cases to calculate the bandwidth adjustment in terms of the delay bound adjustment. Then, we have two sub-procedures Cal_Delay() and Cal_Bandwidth(). Cal_Delay() applies function \( f() \) to calculate the adjustment of delay bound in terms of modified bandwidth reservation; Cal_Bandwidth() applies the inverse function \( f^{-1} \) to get the adjustment of bandwidth reservation if delay bound has changed.

Using these two procedures, DAA is able to calculate adjustments link by link. For each individual adjustment, both of the two sub-procedures are used once. For example, for flow \( i \), in order to decrease the load at one link \( l \), we decide to increase the amount of
bandwidth reserved for previous flows at two related links $j$ and $k$. First, for link $j$, we use function $f()$ to calculate the reduced delay of link $j$ in terms of the increased amount of reserved bandwidth. After updating the new reservation of link $j$, we apply function $f^{-1}()$ to calculate the bandwidth saved for link $i$ in terms of the saved delay bound which is exactly the reduced delay shifted from link $j$. Then, we update the modified flow reservation and repeat the same steps for link $k$.

4.5 Global Balance Adjustment (GBA)

GBA is used to balance the link loads of a current request $i$ and its related links as shown in Figure 5. Due to the cost of adjustment, we consider performing GBA when a new request flow arrives. If the condition in line 2 is satisfied, GBA will distribute some bandwidth from the lightest load links to heaviest load ones.
General_Balance_Adjustment()

1. find r_max, r_min, link_min, link_max;
2. if (r_max - r_min) > Threshold
3. r_t = (r_max + r_min)/2;
4. for link l ∈ link_min
5. Benefit = Capacity(l)*(r_t - r_min)
6. ∀(l) = get link coverage(l);
7. Exclude links with load < r_t from ∀(l)
8. for each link m ∈ ∀(l)
9. Higher_Load(l, ∀(l), r_t);
10. Bene(m) = Benefit*r(m)/Σr(m);
11. D_Benefit = Cal_Delay(Bene(m));
12. Cal_Bandwidth(D_Benefit);
13. for link l ∈ link_max
14. ∀(l) = get link coverage(l);
15. Exclude links with load > r_t from ∀(l)
16. for each link m ∈ ∀(l)
17. Lower_Load(l, ∀(l), r_t);
18. Bene(m) = capacity(m)*(r_t - r(m));
19. D_Benefit = DAA_Cal_Delay(Bene(m));
20. Cal_Bandwidth(D_Benefit);

Figure 5. General Balance Adjustment Algorithm.

For request i, GBA firstly checks the link loads for all intermediate links, finds the maximum link load \( r_{\text{max}} \) and all intermediate links with load of \( r_{\text{max}} \), which is grouped in a set \( \text{link}_\text{max} \); it also finds the minimum link load \( r_{\text{min}} \) and all links with load \( r_{\text{min}} \), grouped in a set \( \text{link}_\text{min} \). Reason of focusing on these links is because links in \( \text{link}_\text{max} \) are more likely to be critical link so reducing their loads will give them more flexibility in later partitioning; links in \( \text{link}_\text{min} \) have the most flexibility now, so taking bandwidth from them will not make their load balance worse from the overall aspect. Currently, we perform GBA to adjust link loads as long as the difference of \( r_{\text{max}} \) and \( r_{\text{min}} \) is larger than a threshold, e.g., zero. The idea of adjustment is to make link loads
close to a target load, denoted by $r_t$. There are several ways to define $r_t$: if the request flows are distributed evenly like the uniform distribution, $r_t$ could be the mean of the loads of all links in the network; if the input is not uniform distributed and we are more interested in reducing the reservation cost for each flow, we set $r_t = (r_{max} + r_{min}) / 2$. For each link in set $\text{link}_{min}$, procedure Higher_Load() allocates a proper amount of its bandwidth, which is $C_i(r_t - r_{min})$, to its related links such that its load will be raised to $r_t$. The related links are chosen by Find_Link_Coverage() procedure and those links with a load lower than $r_t$ will be excluded, because their loads are sufficiently low. Then, the bandwidth requirement is allocated to the related links proportionally to their loads by procedure Cal_delay() and Cal_Bandwidth(). Furthermore, each link $m$ in set $\text{link}_{max}$ gains extra bandwidth from links in $\Psi(m)$; Lower_Load() is conducted similarly as Higher_Load().

### 4.6 Local Dynamic Adjustment (LDA)

Currently, LDA is only called when an initial admission test is failed. Because it tries to accommodate a request, its adjustment may cause minor load imbalance in the system. LDA chooses the most-used link $l^*$ to adjust, which is the one with the smallest available bandwidth. We choose such a link for two reasons. First, this link is more likely to be the critical link of the flow. Adjusting its load helps reduce the smoothing delay for the flow,
because the burst size determines its smoothing delay, and the burst size of a flow is usually much larger than its largest packet size. Therefore, finding the critical link and reserving more bandwidth to it will help the request to meet the delay requirement.

Second, a link with the smallest available bandwidth is more likely to be a bottleneck link as discussed earlier. Adjusting this link is helpful to avoid potential bottlenecks.

Local_Dynamic_Adjustment()

1. Find most-used link $l^*$
2. Find link coverage $W(l^*)$
3. $r = 1$; // use all available bandwidth of those links in $W(l^*)$ for checking
4. Check = DAA_CHECK($l^*$, $W(l^*)$, $r$);
5. for link $l \in W(l^*)$
6. Assign ratio $r$ of available bandwidth of link $l$ to reduce the bandwidth needed for $l^*$;
7. Calculate bandwidth benefit for $l$;
8. Admission_flag = Admission Test();
9. if Admission_flag == 1 // adjust works
10. Local_Adjustment();
11. while (termination != 1 and iterations are less than k times)
12. $r = 0.5$; // use dichotomy to find $r$, not to exhaust link available bandwidth
13. Check = DAA_CHECK($l^*$, $W(l^*)$, $r$);
14. if Check == 1 // adjustment works
15. Allocation based on a flow slack;
16. Check if links $\in W(l^*)$ and links in flow i have about the same link load after adjustment and allocation;
17. if false // link loads unbalanced
18. change ratio $r$;
19. else // more bandwidth is needed for $l^*$
20. increase ratio $r$;
21. else // still can not accommodate
22. Reject request i;

Figure 6. Local Dynamic Adjustment Algorithm.

When the most-used link $l^*$ is found, LDA firstly finds its link coverage $W(l^*)$. To avoid unnecessary computation, LDA checks if the request could be admitted if 100% available bandwidth of links in $W(l^*)$ are used in line 4, where $r$ represents how much of
available bandwidth are taken. We use \( r=1 \) in this case. If the request is not admitted in this case, it will be rejected. Otherwise, \texttt{Local\_Adjustment()} procedure uses a dichotomy approach (from line 11 to 20) to adjust related links, such that we do not exhaust link bandwidth or create bottlenecks, and try to let all related links have same loads after admitting this request.

4.7 Algorithm Complexity

We approximate the computational complexity for each round when a new request flow arrives. We first take a look at the computational complexity of LSS. Since the actual running time of LSS depends on the choice of threshold, considering the dichotomy algorithm converges quickly, we set \( m \) as the maximum running loops in LSS, without loss of generality. Simulation shows \( m=50 \) works well. Therefore, the complexity of LSS is \( O(m*n) \), where \( n \) is the number of hops. Because \( n \) is relative much smaller, the computational complexity of LSS is \( O(m) \).

When a request flow meets both the conditions of GBA and LDA, our DAA algorithm performs in three steps: GBA is applied first and LDA is followed; at last, LSS is called to perform the actual partition based on the adjusted situation. Thus, compared to LSS, our DAA algorithm brings in the extra cost due to GBA and LDA. We will look at their costs separately. Looking back to the algorithm of GBA, its first three lines all
have the complexity as $O(1)$. Line 4 to 11 is the part of distributing bandwidth to links along the route. There are at most $(n-1)$ links in the set $\text{link\_min}$, if condition in line 2 satisfies. Line 5 to 7 has the total complexity of $O(N)$, where $N$ is the number of admitted flows so far. Since we choose to distribute the bandwidth to links along the route, there are at most $(n-2)$ qualified links and line 8 to 12 has a total complexity $O(n-2)$. Thus, line 4 to 12 has a total complexity as $O((n-1)*N)$. Since $N$ is always larger than $n$, the distribution part of GBA has the complexity of $O(N)$. Similarly, for line 13 to 20 part, there are at most $(n-1)$ links in the set $\text{link\_max}$ and line 14 to 15 has the complexity of $O(N)$ and line 16 to 20 has the complexity as $O(E)$, where $E$ is the total number of links of the network, because we choose link_coverage by the way (c) as in section 4.4.2. In fact, if we denote $V$ as the set of nodes representing routers in the topology and $d(v)$ as the degree of a node $v$ representing the number of links that are connected to $v$, we can have a upper bound of the size of link_coverage of any single link $l$ as $O(E^*)$.

$$O(E^*) = \sum_{v \in \lambda(i,n)}[d(v)-1]$$

where $\lambda(i,n)$ denotes the set of nodes whose distance/hops to any one of nodes of link $l$ is no larger than $n$. Therefore, the total complexity through line 13 to 20 is $O((n-1)*(N+E^*))$, which could be approximated as $O(N+E^*)$. Thus, the total complexity of GBA is $O(N+E^*)$.

Concerning LDA algorithm, since each run of function DAA_CHECK is similar to the second procedure of reducing link loads in GBA, which has the complexity of
$O(N+E^*)$, the first part of checking feasibility of LDA through line 1 to 4 has the complexity of $O(N+E^*)$. If the result of check is positive, the loop to find a suitable value of r to evenly adjust the reservations is applied which results in the complexity of $O(k*[N+E^*])$, where $k$ is the total allowable number of iterations. Because this loop is another dichotomy that converges very quickly, we set a small number $k$ such as 10 in simulation later, and the condition in line 16 could be loose. Thus, the complexity of LDA is $O(k*[N+E^*])$.

In conclusion, our DAA brings in extra worst-case complexity of $O(k*[N+E^*])$. Note that LDA is only applied when the request has been rejected, so it is not called frequently. Thus, in most cases, as LDA is not applied frequently, our algorithm DAA only performs GBA which has an extra cost as $O(N+E)$. Note that, even GBA is not applied every time. Further conditions of applying GBA and LDA will be discussed in future.
Chapter 5

Simulations

In this chapter, we will firstly introduce simulation environment, followed by the evaluation criterion. Then, we perform our algorithm DAA on several topologies, and compare the results with the conventional scheme LSS. To better evaluate the effectiveness of DAA, we not only test it on both symmetric and asymmetric topologies, but also extend the simulations with imbalanced input traffic. Several typical situations are discussed case by case.

5.1 Simulation Settings

In this section, we will specify the parameters of simulation environment. We evaluate our DAA algorithm for unicast flows with deterministic QoS requirements. For incoming traffic, each VoIP stream has an average rate of 13 kbps and a peak rate of 34 kbps. We interleave different VoIP streams to generate aggregate traffic traces with an average rate of 100 kbps, denoted by $\rho_{avg}$. The aggregated flows are assumed to have fairly long lifetime, during the simulation, the connections are thought to be static and would always stay there. We tried different end-user link capacities $C$ and denote the ratio $C/\rho_{avg}$ as $r$. A typical end-to-end delay requirement is set to 60ms. A flow burst size is
5kbits and its largest packet size is 1kbits. We defined a topology with 35 nodes showed in Figure 7, to emulate the sprint IP backbone topology [13][35]. Topology 1 is tested throughout all simulations. In addition, we will define some other simple but extreme topologies in the following cases. For each topology, there are two types of links: backbone links and end-user links. We use $w$ to denote the ratio of the capacity of a backbone link to that of an end-user link. Assume that the shortest path routing protocol is implemented.

Figure 7. Topology 1.

For each trial on a topology, each request is across a path with a fixed number of hops, e.g., three hops or four hops on topology 1. The assumption of regulating the number of hops enables us to eliminate the impact of different hops to resource consumption of one admitted flow and thus simply use the total number of admitted flows to evaluate the overall resource utilization. Note that we can still change the number of hops in different trials and will study its impact later.
5.2 Evaluation Criteria

Based on the simulation setup, we know that the flows are with the same parameters including the number of hops, the average rate, the burst size, the largest packet size and the deterministic QoS delay requirement. Therefore, more flows admitted, higher the system utilization is. For each trial on a topology, we generate \( N \) request flows, which is a number that is much larger than the maximum number of flows that the system can support. We choose \( N = \frac{\sum_i (\text{capacity}(l))/\rho_{\text{avg}} \times \text{hops})}{\text{capacity}(l)/(\rho_{\text{avg}} \times \text{hops})} \). Then, we compare the number of admitted flows by using LSS and DAA, respectively, and represent the difference percentage.

5.3 Simulation Results

We present the simulation results in different cases. For each case, we perform the simulations on topology 1 and other selected topologies.

- **Effect of heterogeneity of link capacity**

  In this section, we assume the input is uniformly distributed, i.e., a source is randomly chosen among all the nodes in a given topology. We then perform two types of tests to find out how the heterogeneity of link capacity affects the improvement of DAA over LSS with random inputs.

  (a) Tests on topologies with a mix of links
Comparison tests in this part are similar to those in [14]: there is a 5*5 mesh topology with a mix of link capacities between 45Mbps to 200Mbps and for each flow, the number of hops is fixed as six; similarly, the capacity links on topology 1 is randomly chosen between 45Mbps to 200Mbps and the number of hops is set to four. Rest of settings about flow specification and end-to-end delay QoS remain the same. The results are showed in Figure 8.

![Figure 8. Comparison of DAA and LSS on topology 1 and 5*5 mesh topology.](image)

In Figure 8, the height of bars represents the average number of maximized admitted flows. It is clear to see that DAA outperforms LSS in this mix of link capacities case that it admits 36.45% and 41.78% more flows on topology 1 and 5*5 grid mesh, respectively. Note that since we are more concerning about whether and how much DAA outperforms LSS, we will take the improvement percentage as the result in the following comparisons.

(b) Tests on two-hierarchy topologies

Results in (a) show that: DAA outperforms LSS on topologies with random mix of link capacities. Back to the classic two hierarchy topology 1, in order to learn that how
the difference between capacities of backbone links and of end-user links, we run DAA with $w=1,2,4$, respectively and compare the results with LSS in Figure 9.

Figure 9. Improvement of DAA over LSS on topology 1 with hops=4.

Additionally, we perform tests on more two-hierarchy topologies showed in Figure 10 with the number of hops equal to three. Figure 11 shows the performance improvement of DAA over LSS for some selected classic topologies. For each figure in Figure 9 and Figure 11, the x-axis is $r$ ranging from 10 to 100. We repeat the simulation with different $w$ shown in the figure, i.e., $w=2$, 4, 8 and 16 in Figure 11 that $w$ and $r$ determine the link capacities. The y-axis shows the percentage increase of the total number of admitted flows. Since the average incoming flow rate is 100kbps, when $w = 4$, $r = 50$, for topology 2, the capacity of end-user links is 5Mbps and the capacity of backbone links is 20Mbps.
Figure 10. Even and uneven topologies.

Figure 11. Improvement of DAA over LSS with hops=3 on (a) topology 2, (b) topology 3, (c) topology 4, and (d) topology 5, respectively.
The simulations on those simply topologies are more likely to be extreme cases such that the choice of \( w = 16 \) lets the difference of capacities of backbone links and of end-user links be significant. Results clearly show that DAA outperforms LSS over 30\% on both symmetric topologies such as Topology 2 and 4 and asymmetric ones such as Topology 1, 3 and 5. We are more interested in asymmetric topologies because in reality, most network topologies are of that type. We use \( w \) to represent the imbalanced capacity situation among network links. The larger \( w \), the better DAA performs.

- **Effectiveness of LDA**

  As we introduced in Chapter 4, LDA helps to solve the bottleneck link problem. When a request does not meet the end-to-end delay requirement, we will apply LDA to make a local adjustment particularly for the request. This helps to reduce the rejection rate but may disturb the load balance, because it may use more bandwidth from related links and make them overloaded or exhausted. Thus, we like to examine if this local adjustment is helpful to overall performance. Since we have shown DAA’s performance with LDA earlier, we further run DAA again without LDA, and present the performance difference of DAA with LDA over DAA without LDA in Figure 12. The x-axis is \( r \), same as before, and the y-axis is the difference percentage of DAA over LSS with LDA and without LDA. We only show the comparison results on topology 1, 3 and 4, and results on other topologies have similar trends.
Figure 12. Performance Comparison of DAA and LSS with and without LDA on (a) topology 1, (b) topology 3, and (c) topology 4.

We can see that the difference is positive in Figure 12, which means the overall performance increased about 10% with LDA. For topology 1, LDA enhances the performance a little more when the parameter $r$ is small. That is because LDA is always applied close to the end of the resource allocation procedure. Thus, flows getting benefits from LDA are limited and their admissions contribute more when $r$ and the number of total admitted flows are small. In addition, the higher $w$, the more benefits LDA provides. Therefore, we adopt LDA in all other simulations.

- **Effect of path length**

Although we fix the number of hops in the previous tests, we are interested in seeing how the path length affects the performance of DAA. First, we choose two highly symmetric topologies as the extreme case in order to reduce the effect from topology. The results are shown in Figure 13. We test hops=4, 5, 6 on a 5*5 mesh topology and test
hops=5, 6, 7 on a larger 8*8 mesh topology.

Figure 13. Comparison of Improvement with different path length on (a) 5*5 mesh topology and (b) 8*8 mesh topology.

From the results in Figure 13, DAA gets improvement more than 30% with different path lengths in most cases. Considering the only exception that the improvement with six hops on 5*5 grid topology is only about 20%, one major reason is that we employ the shortest path routing protocol and the size of 5*5 mesh topology is limited, thus there are much fewer applicable routes to be picked. Note that in a larger 8*8 mesh topology where there are more long routes, the improvement of DAA over LSS with six hops becomes over 30% as shown in Figure 13(b).
(a) \hspace{1cm} (b) \hspace{1cm} (c)

Figure 14. Improvement of DAA over LSS on topology 1 with (a) hops=5, (b) hops=4, and (c) hops=3, respectively.

We test the algorithms on topology 1 with hops=3, 4, 5 as showed in Figure 14. Apart from the earlier conclusion that DAA gets greater improvement with larger $w$ and other fixed parameters, it is clear to see that fewer hops, larger improvement that DAA gets. The reason is most likely to be the shortest routing protocol we used that in topology 1, there are a limited number of long routes with more than four hops. Therefore, the length of paths does not affect much, and DAA always outperforms LSS with at least 30% improvement. The less influence from the routing protocol, the larger improvement is.

- Effect of active nodes

All the previous tests are done with uniformly distributed incoming traffic and DAA outperforms LSS on both symmetric and asymmetric topologies. In reality, network topologies are often asymmetric, and input traffic is not uniformly distributed, such that some areas of a network may be congested sooner than other areas. We can simply define two types of imbalanced input traffic: one is to make some nodes more active in
communications; the other one is to have some active sub-domains. In the following, we choose the first type of active nodes. The modified topologies are shown in Figure 15, where a triangle node represents a heavy node, which issues a request with a higher probability than a regular circle node.

(a). Modified Topology 4 (b). Modified Topology 5 (c). Modified Topology 1

Figure 15. Topologies with heavy nodes.

(a) (b)

Figure 16. Comparison of DAA and LSS with twice active heavy nodes on (a) modified topology 4 and (b) modified topology 5.
Comparisons of OM and LSS

Figure 17. Comparison of DAA and LSS with heavy nodes on topology 1, Hops = 4.

With twice active heavy nodes on topology 4 and 5, DAA still outperformed LSS on both symmetric and asymmetric topologies in Figure 16. The improvement is around 40% on these two tested topologies. On topology 1, we get the same results: DAA always has higher system utilization that it admits more than 30% flows with twice, three times, or four times active nodes.

- Effect of active sub-domains

In the following, we test DAA with the second type of imbalanced input traffic that there are some active sub-domains on each topology in Figure 18. In this case, some nodes from sub-domains are more active and have more communications. As a result, the traffic between two active sub-domains may increase and get congested during some time periods. Although the time-variant traffic congestion may not last very long, it still could exhaust the bandwidth of neighbor links very quickly.
We further specify two types of time-variant imbalanced traffic: one is called bursty congestion that the congestion is severe, but only occurs when the link loads are low and lasts for a short period; the other one is called long period congestion that traffic among sub-domains last fairly long.

(1) Bursty congestion

Bursty congestion is assumed to be relatively short and occurs at the beginning when the overall loads are low. During that period, some selected sub-domains are considered to be more active that they have more communication requests among them. Among the first bunch of incoming requests, a flow has higher probability to be among the active sub-domains. We denote the probability as $r_{\text{bursty}}$ such that one flow has probability of $r_{\text{bursty}}$ to be picked from active sub-domains, whereas having probability of $(1-r_{\text{bursty}})$
to be equally picked throughout the whole network. The larger $r_{\text{bursty}}$, the more severe
of bursty congestion is. Additionally, we have to specify the length of a bursty period.
Since flows have fairly long lifetime in our simulations, a tentative number of earlier
requests, $C/\rho_{\text{avg}}$, is set to roughly specify the length of bursty period occurs at the
beginning of simulations.

![Comparison of DAA and LSS](image1.png)

(a). $r_{\text{bursty}} = 0.5$

(b). $r_{\text{bursty}} = 0.7$

Figure 19. Comparison of DAA and LSS with bursty congestion on topology 6, hops=4.

![Comparison of DAA and LSS](image2.png)

(a). $r_{\text{bursty}} = 0.5$

(b). $r_{\text{bursty}} = 0.7$

Figure 20. Comparison of DAA and LSS with bursty congestion on modified topology 1,
hops=4.
We test DAA and LSS on topology 6 and modified topology 1, as shown in Figure 18(a) and (c). Among the first $C/\rho_{avg}$ of request flows, there are two active sub-domains on each testing topologies and we set the ratio $r_{bursty}$ to be 0.5 and 0.7, respectively. The results in Figure 19 and Figure 20 show that DAA still outperforms LSS with an improvement about over 30%.

(2) Long period congestion

Compared with bursty congestion, long period congestion means that the communications among active sub-domains last for a relatively long period. Currently, we assume that those selected sub-domains keep active for all the time. Similarly, we denote the ratio $r_{long}$ such that one flow has the probability of $r_{long}$ to be picked among active sub-domains and the probability of $(1-r_{long})$ to be equally picked throughout the whole network. To avoid exhausting some widely used links’ resource, $r_{long}$ is always smaller than $r_{bursty}$ because congestion in this case lasts longer.

Figure 21. Comparison of DAA and LSS with long congestion on topology 6, hops=4.
The simulation results are showed in Figure 21 and Figure 22. When the ratio $r_{\text{long}}$ is set to be 0.1 or 0.2, DAA gets a 25% improvement on topology 6 and 30% on modified topology 1. Note that the difference of improvement under $w=1$ and $w=2$ becomes smaller on modified topology 1. That is because long period congestion reduces the impact of heterogeneity of link capacity.

On the other hand, we further develop the imbalanced traffic on topology 6. We modified topology 6 as showed in Figure 18(b) such that there are four active sub-domains 1 to 4. We like to see how DAA works if the congestion firstly occurs between domain 1 and 3 and then occurs between domain 2 and 4. This is more close to the time-variant imbalanced traffic among different sub-domains in real traffic. We emulated a new long period congestion on modified topology 6 as follows. Based on all previous simulation results, we know that the acceptance of LSS is about 7% to 12%, and the acceptance of DAA is always 30% larger. Then we are able to assume that at least the first 10% of total
requests are very likely to be admitted by DAA. Thus, among these first 10% of total request flows, we generate them in a manner that the first half of them have a higher probability to be the traffic between domain 1 and 2, and the latter half of them are more likely to come between domain 3 and 4. In another word, there is a half-long period congestion with parameter \( r_{long} \) between domain 1 and 2 and then they become normal, whereas there is another half-long period congestion with \( r_{long} \) between domain 3 and 4.

![Comparison of DAA and LSS](image)

Figure 23. Comparison of DAA and LSS with two half-long period congestion on modified topology 6, hops=4.

From the results in Figure 23, it is clear that the improvement of DAA over LSS is over 30% and is much larger when \( w \) increases. Additionally, the larger \( r_{long} \), the greater improvement is. Thus, DAA performs better in more imbalanced cases.

- **Effect of Comprehensive Imbalance Input Traffic**
Based on the above discussion, there are three main parameters that we have to consider: first, ratio $w$ represents the extent of heterogeneity of link capacity; second, some nodes are considered to be active that they have larger probability to be picked as sources (destinations); third, during some particular periods, two or more sub-domains could be active that they have more communications among them. Thus, we combine these three factors together in the following simulations on topology 1 with heavy nodes and active sub-domains in Figure 24.
Figure 25. Comparison of DAA and LSS with twice active nodes and active sub-domains on topology 1, hops=4.
Figure 26. Comparison of DAA and LSS with three times active nodes and active sub-domains on topology 1, hops=4.
For each set of tests, we make the heavy nodes twice, three times and four times active than normal nodes, respectively; and we perform with both of bursty congestion and long period congestion. From the simulation results in the same subfigure in Figure 25, Figure 26 and Figure 27, we can compare the results with different active subdomains but with different kinds of active nodes. It is clear that DAA improves LSS by at least 30% with those active nodes. When we read the results with the same type of active nodes but in different congestion cases, we can see that the long period congestion
becomes the major reason of imbalanced input traffic. If we look at three sub-figures(c), we can find out that the long period congestion with a large $r_{long}$ parameter reduces the impact of heterogeneity of link capacity to improvement.

5.4 Simulation Conclusion

In this chapter, we have compared DAA and LSS on several topologies, including a 35 nodes topology, which is similar to Sprint IP Backbone topology [13][35]. At first, we have conducted some general tests on both symmetric and asymmetric topologies, as well as an equivalent test as in [14] on 5*5 mesh topology with same parameters and the improvement could be over 30%. Then, we have shown the local adjustment LDA helps the overall utilization, especially when the ratio of backbone link capacity over end-user link capacity is small. Additionally, the effect of path length has also been discussed. Next, in terms of three main parameters affecting link capacity and input traffic trend, we have analyzed and tested them case by case.

(a) Ratio $w$ represents the extent of heterogeneity of link capacity, and the larger $w$, greater the improvement is. Sometimes the improvement is over 60% or even 100% with larger $w$.

(b) We have considered heavy nodes, which generate more requests. DAA still outperforms LSS by 30%.

(c) Then we have emulated two types of time-variant congestion traffic among sub-
domains. The more imbalanced of the input traffic, the greater improvement that DAA gets. In most cases, the improvement is over 30%.

At the end of this chapter, we have combined these factors together and conducted simulations with comprehensive imbalanced input traffic. The results show that DAA outperforms LSS by at least 30%.
Chapter 6

Conclusion and Future work

Partitioning the end-to-end delay requirement is critical for performance sensitive applications. Conventional schemes deal with this topic based on instant load situations. They can ensure each hop has enough bandwidth resource to meet the allocated local delay bound. However, their instant estimation of traffic loads may not reveal the long time traffic trend and thus they sometimes fail to avoid the imbalanced reservation which may cause congestion or bottleneck links very quickly. Therefore, to increase the network resource utilization, we propose a novel Dynamic Allocation Adjustment (DAA) scheme which provides a way to intelligently adjust the existing reservations. It can not only balance the loads along the path of a request flow, but also spread the traffic loads to other selected neighbor links. DAA reduces the bandwidth consumption and balance the loads for links in a broader range other than the path. Moreover, the adjustment function helps to address the bottleneck link problem. To test the effectiveness of DAA, several types of imbalanced traffic have been emulated and DAA is conducted on both of symmetric and asymmetric topologies. Results show that DAA outperforms existing solutions by at least 30%.

Further work can be developed from the following aspects.
(a). DAA can be extended to deal with the delay bound partition problem with end-to-end statistical guarantees.

(b). In this project, we assume that the flows are long-lived and thus the maximum number of admitted flows is used as the criterion to evaluate system utilization. We can let each flow have a lifetime and system releases the reservation resources when the flow terminates. In that case, the system can employ a new measuring criterion associated with a time index, and we can get a better understanding of how DAA works when the time-variant congestion occurs.

(c). DAA increases the system efficiency by mainly reducing the load of critical link to alleviate the smoothing delay. More explicit relationship of link loads and overall bandwidth resource needs to be investigated. For example, a good heuristic is that all flows sharing the same route should have a similar reservation pattern at each hop.

(d). In order to increase the efficiency of DAA, several problems need to be clearly specified such as: more specified condition about when to perform General Balance Adjustment (GBA) in order to reduce the costs; selecting more links as potential candidates to adjust other than neighbor links which are directly connected to links of the request flow; optimally picking flows when there are multiple existing flows going on the links to adjust, etc.

(e). On the other hand, more efforts are needed to reduce the computational complexity of DAA. As a network admits more flows, many flows are overlapped, and
DAA needs to search a large table with the reservation information to get a relatively small number of flows' information. One solution may be to set up an individual table at each hop with the reservation information of a limited number of flows.
REFERENCES


[19] IETF: Differentiated Services Working Group,

[20] IETF: Integrated Services Working Group,


[35] Sprint North American IP Backbone,


