# QoS Support in Large-Scale Computer Networks based on Aggregate Scheduling and BGP Routing Enhancement

by

**Wei Sun**

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
2007

Doctoral Committee:
Professor Kang G. Shin, Chair
Professor Farnam Jahanian
Professor A. Galip Ulsoy
Assistant Professor Z. Morley Mao

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.



ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

To my grandparents

# ACKNOWLEDGEMENTS

This dissertation would have never been completed without the guidance, care and support of many wonderful people.

First of all, I would like to thank my advisor, Prof. Kang G. Shin, for his guidance during the course of my Ph.D. study. I have benefited tremendously from his vision, technical insights, and strong pursuit of excellence. I would also like to thank him for his patience with me and for giving me the freedom to select research topics that I was interested in.

During the last three years, I was fortunate to have the opportunity to work closely with Prof. Z. Morley Mao. I thank her for her guidance and all the inspiring discussions that greatly refined my research ideas. I learned a lot from her deep insights regarding BGP routing, creativity and work ethic. I would also like to thank Prof. A. Galip Ulsoy and Prof. Farnam Jahanian for serving on my thesis committee and for their critical feedback and advice that helped me to improve the overall quality of this dissertation.

My gratitude also goes to the staff members of CSE division, especially BJ Monaghan, Stephen Reger, Karen Liska, and Dawn Freysinger for their excellent support.

I would like to thank my friends and current/former fellow students of RTCL. I thank Zhigang Chen, Jian Wu, Daji Qiao, Mohamed El Gendy, Haining Wang, Zonghua (Sam) Gu, Kyu-Han Kim, and Shige Wang for their friendship and help at various stages of my study. I also thank the members of networking group for their invaluable comments on my research work.

Last but not least, I would like to thank my family, especially my wife Haini, for their constant love and support.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF APPENDICES

# CHAPTER 1

# Introduction

Today's Internet has become an indispensable part of people's daily life. In addition to the traditional web surfing and file downloading, it also provides new applications such as voice-over-IP (VoIP), IPTV, video conferencing, and Internet gaming, which attract millions of users. These real-time, multimedia applications require the network to provide better Quality-of-Service (QoS) than the currently dominant best-effort service, in terms of bandwidth, delay, jitter, and loss rate. Packet scheduling and routing are two basic functions of a data network. Packet scheduling determines the order of packets that are transmitted from a network device to its neighbor; routing is to choose a path between two network devices. To provide QoS in the current Internet to support those emerging real-time, multimedia applications, it is important to have high-performance, low-cost scheduling algorithms, as well as efficient routing algorithms that converge to new routes quickly when network changes occur.

In this dissertation, we focus on QoS support in packet scheduling and routing. More specifically, we study the performance of aggregate scheduling by both analysis and simulation. Also, we propose enhancements to the current BGP (Border Gateway Protocol) to reduce both its convergence time and the amount of routing updates. We also analyze the impact of BGP routing changes on application traffic.

## 1.1 Aggregate Scheduling

To provide QoS support in a data network, the IntServ architecture [1] has been proposed, which supports QoS via *per-flow* resource reservation (*e.g.*, RSVP [2]) and packet scheduling.

Numerous scheduling algorithms (*e.g.*, see [3] for an excellent survey) have been proposed to support IntServ-like QoS, such as fairness, bounded per-flow (per-node or e2e) delay and backlog under a certain traffic model like the token bucket model. One class of such scheduling algorithms are called *guaranteed rate* (GR) scheduling algorithms [4], which include WFQ (Weighted Fair Queueing), GPS (Generalized Process Sharing) [5], VC (Virtual Clock) [6], etc. Since both its resource reservation and packet scheduling are per-flow-based, and hence, the routers in the network must keep a large number of flow states, IntServ does not scale well for use in the core of the Internet that carries millions of flows.

To solve the IntServ's scalability problem, the DiffServ architecture [7] has been proposed by classifying traffic into a number of predefined classes, such as Expedited Forwarding (EF), Assured Forwarding (AF), and Best-Effort (BE) at the edge of each DiffServ domain. The traffic class is identified by the marking in the DiffServ field of each packet. Flow information is visible only at edge routers of a DiffServ domain, and sophisticated packet classification, marking, policing, and shaping operations need only be implemented at edge routers. Within each DiffServ domain, packets receive a particular per-hop forwarding behavior at routers on their path based on the DiffServ field in their IP headers. In other words, flows are invisible and packet scheduling is done based on the traffic class, *not* based on individual flows. The EF class [8] receives priority over AF and BE classes. The DiffServ architecture is more scalable than IntServ, but FIFO queueing—commonly used to schedule packets in each traffic class—is not suitable for hard QoS guarantees [9, 10].

In this dissertation, we consider an extension of the IntServ architecture to support traffic aggregation. This extension is made on the premise that there are *aggregation regions* in the network, which "see" only aggregated (not individual) flows. Resource reservation and packet scheduling in an aggregation region are done on a per-aggregate basis. An example of this is the Virtual Paths (VPs) in ATM networks. Traffic aggregation has been studied extensively: see [11, 12, 13, 14] for resource reservation, [15, 16] for the admission control of aggregates, and [17] for flow state aggregation. This dissertation focuses on the scheduling issues associated with traffic aggregation, and evaluates the deterministic e2e delay bounds of aggregate scheduling when guaranteed-rate (GR) algorithms [18] are used.

Traffic aggregates discussed here is similar to the *traffic trunk*—which is defined as an aggregate of traffic flows that belong to the same class—in Multiprotocol Label Switching (MPLS) [19,

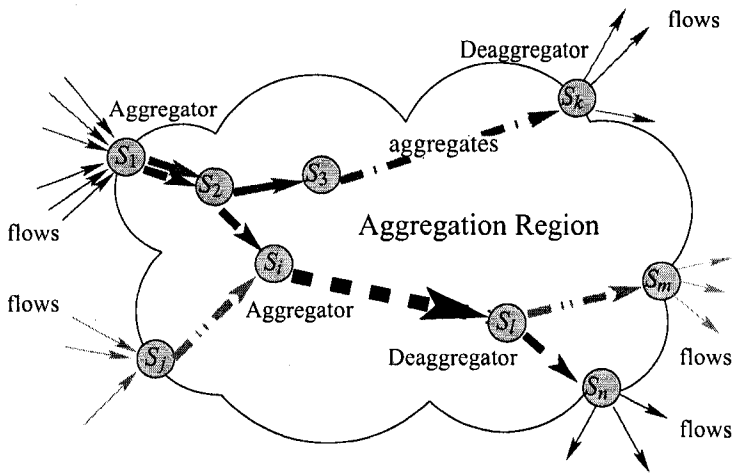



Figure 1.1: Aggregate scheduling

20]. As shown in Fig. 1.1, some flows enter an aggregation region from ingress router $S_1$, and share the same path for a number of hops inside the region. Within the region they are treated as a *single* aggregate at the routers on the path. $S_1$ aggregates or "bundles" the flows by using GR scheduling algorithms. When the aggregate leaves $S_n$, it is split back into individual flows. Similarly, another aggregate can be set up between $S_1$ and $S_k$. In general, a traffic aggregate can be created and terminated at any point in the network, and it can be created recursively. For example, at $S_i$ two aggregates are bundled again into another (higher level) aggregate, which is terminated later at $S_l$. The router (*e.g.*, $S_1$) that aggregates flows (using GR algorithms) is called an *aggregator*; the router (*e.g.*, $S_k$) that splits the aggregate is called a *deaggregator*. The term "flows" ("traffic aggregates" or "aggregates") means the entities before (after) aggregation. We specify the aggregator and deaggregator as part of the aggregation region, although they can differentiate among the constituent flows of each aggregate.

Similarly to the DiffServ architecture, this aggregate scheduling architecture pushes the overhead to the edge of the network and keeps the core network simple. The admission control and resource reservation of aggregate scheduling can be implemented by the extension of RSVP that supports traffic aggregation [11]. The idea is to use aggregate *Path* and *Resv* messages between the aggregator and the deaggregator, and hide the e2e per-flow RSVP messages from the core routers in the aggregation region. Aggregate Path messages are sent from the aggregator to the deaggregator, and aggregate Resv messages are sent from the deaggregator to the aggregator, thus

establishing the aggregate reservation on behalf of the flows within the same aggregate. Per-flow RSVP messages trigger the transmission of aggregate Path or Resv messages, but these messages themselves are ignored inside the aggregation region. This results in a smaller number of (aggregate) reservations inside the aggregation region. To reduce the number of changes to aggregate reservations, advance reservation or "bulk" reservation is needed. For example, the authors of [21] examined the issue of "bulk" reservation for traffic aggregates.

### Advantages and Disadvantages of Traffic Aggregation

With traffic aggregation, the core routers need to maintain fewer states, making packet classification and scheduling simpler. Flow aggregation is also beneficial to the flows with low bandwidth requirements, because, under GR scheduling algorithms—which couples bandwidth and delay—the flows with low bandwidth requirements will suffer long delay, but aggregation usually alleviates this problem. More importantly, as shown earlier, aggregate-based GR scheduling can provide guaranteed QoS, such as e2e delay bounds.

However, aggregation also comes with its own disadvantages: the flows in the same aggregate cannot isolate from, and protect against, other flows. Therefore, the QoS guarantee of a flow will be affected by others in the same aggregate, and all the flows within an aggregate will receive roughly the same service. Due to the lack of isolation within an aggregate, bursty flows can "steal" the bandwidth from well-behaving flows and unduly degrade their QoS. This problem can become worse in multi-service networks [22]. However, as pointed out in [23], the problem can be controlled if flows are selectively aggregated, *i.e.*, only those flows that have some common characteristics are aggregated (which is similar to our conclusion; see the results in Chapter 2). Moreover, "fair" aggregation can also alleviate this problem. In Chapter 2, we will show this feature by using GR scheduling algorithms to aggregate flows.

## 1.2 BGP Routing Enhancement

Emerging delay- and jitter-sensitive applications not only demand more sophisticated scheduling algorithms, but also impose more stringent requirements on the underlying Internet routing system. As a global network, the Internet has a hierarchical structure. It is first divided into thou-

sands of Autonomous Systems (AS). An AS is a group of networks that generally are under the management of the same entity (such as an ISP) and share the same routing policy. Each AS is assigned a unique AS number. Thus routing function in the Internet is also divided into intra-domain and inter-domain routing. Intra-domain routing is the routing inside an AS; inter-domain routing is the routing between ASes. BGP (Border Gateway Protocol) is the *de facto* inter-domain routing protocol in the current Internet.

BGP routing issues have attracted significant attention from both the research and operator communities. A key problem associated with BGP is the excessive number of BGP updates possibly triggered by routing changes, such as session resets, link failures, and policy changes. For example, a recent study of a large tier-1 ISP shows that within just one minute, a "rich peering" router can experience hundreds of routing updates all at once due partly to the interactions between intra- and inter-domain routing [24]. A related issue is the long convergence time caused by BGP path exploration. The authors of [25, 26, 27] have shown that the BGP convergence time is surprisingly long and depends on the length of the longest backup path. The convergence time is also shown to be proportional to the number of alternative routes to a given destination [28]. The prevalence of multi-homing in AS relationships (e.g., a customer AS peering with multiple providers ASes) [29, 30, 31] increases the number of backup routes in the Internet significantly, which, in turn, prolongs BGP convergence.

To enhance the performance of BGP, we propose a novel method of differentiated update processing. BGP updates are classified into different classes based on whether their routes are used in the forwarding tables of the receiving routers for related destination prefixes. High-priority updates will be processed sooner while low-priority ones will be delayed.

## 1.3 Main Contributions and Organization of the Dissertation

The main contributions of this research are summarized as follows.

1. By using the GR algorithms to schedule traffic aggregates, we show not only the existence of e2e delay bounds, but also the fact that in many cases the bounds are smaller than those of per-flow scheduling [32].

2. Using in-depth simulation we not only confirm the analytical results, but also show the ad-

vantages of aggregate scheduling [32].

3. We propose a differentiated BGP update processing algorithm to reduce both the BGP convergence time and the amount of routing updates [33].

4. We also propose a difference-based route selection algorithm, which significantly reduces the amount of unnecessary exchange of routing information and speeds up BGP convergence during network failures [34].

5. We study the impact of BGP routing changes on application traffic at both AS level and traffic level. We show that the top prefixes generating/receiving more traffic tend to have fewer routing changes [35].

The rest of the dissertation is organized as follows: Chapters 2 and 3 focus on aggregate scheduling. Chapter 2 derives three deterministic bounds of end-to-end delay for aggregate scheduling. It also compares the performance of aggregate scheduling with that of per-flow scheduling. Chapter 3 further enhances the performance of aggregate scheduling by using coordination among multiple nodes along an end-to-end path. Chapter 4 deals with BGP enhancement, and proposes the idea of differentiated BGP update processing. It also compares the performance of the new algorithm with that of the default BGP. Chapter 5 studies the impact of BGP routing changes on application traffic. Finally, Chapter 6 summarizes the main contributions of the dissertation and discusses future directions.

# CHAPTER 2

# End-to-End Delay Bounds for Traffic Aggregates under Guaranteed-Rate Scheduling Algorithms

This chapter evaluates, via both analysis and simulation, the end-to-end (e2e) delay performance of *aggregate scheduling* with guaranteed-rate (GR) algorithms. Deterministic e2e delay bounds for a *single* aggregation are derived under the assumption that all incoming flows at an aggregator conform to the token bucket model. An aggregator can use any of three types of GR scheduling algorithms: stand-alone GR, two-level hierarchical GR, and rate-controlled two-level hierarchical GR. E2e delay bounds are also derived for the case of *multiple* aggregations within an aggregation region when aggregators use the rate-controlled two-level hierarchical GR. By using the GR scheduling algorithms for traffic aggregates, we show not only the existence of delay bounds for each flow, but also the fact that under certain conditions (*e.g.*, when the aggregate traverses a long path after the aggregation point) the bounds are smaller than that of per-flow scheduling. We then compare the analytic delay bounds numerically, and conduct in-depth simulation to (i) confirm the analytic results, and (ii) compare the e2e delays of aggregate and per-flow scheduling. The simulation results have shown that aggregate scheduling is very robust and can exploit statistical multiplexing gains. It performs better than per-flow scheduling in most of the simulation scenarios we considered.

Overall, aggregate scheduling is shown theoretically to provide bounded e2e delays, and practically to provide excellent e2e delay performance. Moreover, it incurs lower scheduling and state-maintenance overheads at routers than per-flow scheduling. All of these salient features make aggregate scheduling very attractive for use in Internet core networks.

The rest of the chapter is organized as follows. Section 2.1 explains the definitions of GR

scheduling [4] and $\mathcal{LR}$ server [36], discusses their relationship, and reviews the delay bound results for per-flow scheduling in [4]. Section 2.2 introduces the concept of fair aggregator, and derives the delay bound for aggregate scheduling under the token bucket traffic model. Two delay bounds are derived: the first for stand-alone aggregator and the second for hierarchical aggregator, both of which use work-conserving GR scheduling algorithms. The hierarchical aggregator is shown to improve the delay bound. Section 2.3 improves the delay bound further by having the aggregators use non-work-conserving scheduling algorithms. Section 2.4 presents numerical results, comparing the above deterministic delay bounds with that of per-flow scheduling. Simulation is also used to compare the delay performance of both aggregate and per-flow scheduling, confirming the benefits of aggregate scheduling derived from the analysis. Section 2.5 discusses some related work on aggregate scheduling, putting our results in a comparative perspective. Finally, Section 2.6 summarizes our contributions.

## 2.1 Guaranteed-Rate Scheduling Algorithms

Before deriving the delay bound under aggregate scheduling, in this section we introduce the definitions of *Guaranteed-Rate* (GR) scheduling algorithm and *Latency-Rate* ($\mathcal{LR}$) server, discuss their relationship, and review the e2e delay bound results for per-flow scheduling. For the convenience of discussion, we first give a list of symbols in Table 2.1.

### 2.1.1 Guaranteed-Rate (GR) Scheduling Algorithms

The authors of [4] defined a class of GR scheduling algorithms. The delay guarantees provided by these algorithms are based on the *Guaranteed Rate Clock* (GRC) value associated with each packet, which is defined as follows [4].

**Definition 2.1.1 (GR Clock Value).** *Consider a flow $f$ associated with a guaranteed rate $r_f$. Let $p_f^j$ and $\ell_f^j$ denote the $j^{th}$ packet of flow $f$ and its length, respectively. Also, let $GRC^i(p_f^j)$ and $A^i(p_f^j)$ denote the GRC value and arrival time of packet $p_f^j$ at router $S_i$, respectively. Then, the*

*GRC values for packets of flow $f$ are given by:*

$$GRC^i(p_f^j) = \begin{cases} 0, & j = 0 \\ \max\{A^i(p_f^j), GRC^i(p_f^{j-1})\} + \frac{l_f^j}{r_f}, & j \geq 1. \end{cases} \tag{2.1}$$

**Definition 2.1.2 (GR Scheduling Algorithm).** *A scheduling algorithm at router $S_i$ is said to belong to the GR class for flow $f$ if it guarantees that packet $p_f^j$ is transmitted by time $GRC^i(p_f^j) + \beta^i$, where $\beta^i$ is a* scheduling constant *[37] that depends on the scheduling algorithm and the router.*

We will henceforth call a router equipped with the GR scheduling algorithm a *GR server*. Many scheduling algorithms are shown in [4] to belong to the GR class. For example, both Packet-level Generalized Processor Sharing (PGPS) [5] and Virtual Clock (VC) [6] are GR scheduling algorithms with $\beta^i = \frac{L_{max}^i}{C^i}$, where $L_{max}^i$ is the maximum packet length seen by router $S_i$ and $C^i$ the output link capacity of $S_i$.

### 2.1.2 Latency-Rate ($\mathcal{LR}$) Server

The authors of [36] also defined a class of scheduling algorithms as *Latency-Rate ($\mathcal{LR}$)* servers, the definition of which is repeated below for self containment, although some notations in the definition are modified for consistency.

**Definition 2.1.3 ($\mathcal{LR}$ Server).** *Let $\tau$ be the starting time of a busy period of flow $f$ in server $S_i$ and $\tau^*$ the time at which the last bit of traffic arrived during the busy period leaves the server. Then, server $S_i$ is an $\mathcal{LR}$ server if and only if a nonnegative constant $C_f^i$ can be found such that, at every instant $t$ in the interval $(\tau, \tau^*]$,*

$$W_f^i(\tau, t) \geq \max\{0, r_f(t - \tau - C_f^i)\}, \tag{2.2}$$

*where $W_f^i(\tau, t)$ denotes the total service provided by the server $S_i$ to flow $f$ during the busy period until time $t$. The minimum nonnegative constant $C_f^i$ satisfying the above inequality is defined as the* latency *of the server, denoted by $\theta_f^i$.*

The definition of the $\mathcal{LR}$ server helps establish the relationship between the burst size of an output flow and that of the corresponding input flow at a router. Before introducing Lemma 2.1.1,

we define the *token bucket traffic model* as follows: flow $f$ conforms to the token bucket $(\sigma_f, \rho_f)$ if for any time instant $\tau$ and $t$ satisfying $0 \leq \tau < t$, its traffic volume arrived during $(\tau, t]$, $A_f(\tau, t)$, satisfies:

$$A_f(\tau, t) \leq \sigma_f + \rho_f \cdot (t - \tau), \tag{2.3}$$

where $\sigma_f$ and $\rho_f$ are the burst size and average rate of flow $f$, respectively.

**Lemma 2.1.1.** *Suppose an incoming flow $f$ to router $S_i$ conforms to the token bucket model $(\sigma_f, \rho_f)$. If $S_i$ is an $\mathcal{LR}$ server with parameters $(\theta^i_f, r_f)$ for flow $f$, where $\theta^i_f$ is the latency at $S_i$ and $r_f$ ($r_f \geq \rho_f$) the guaranteed rate for flow $f$, respectively, then the output traffic of flow $f$ conforms to the token bucket model with parameters $(\tilde{\sigma}_f, \rho_f)$, where $\tilde{\sigma}_f \leq \sigma_f + \theta^i_f \cdot \rho_f$.*

For example, both PGPS and VC are $\mathcal{LR}$ servers with $\theta^i_f = \frac{\ell^{max}_f}{r_f} + \frac{L^i_{max}}{C^i}$ [36], in which case we have

$$\tilde{\sigma}_f \leq \sigma_f + \ell^{max}_f + \frac{\rho_f \cdot L^i_{max}}{C^i}. \tag{2.4}$$

The proof of Lemma 2.1.1 is similar to that of Theorem 3 in [36], which assumes $\rho_f = r_f$; but the result can be easily extended to the case of $\rho_f \leq r_f$.

### 2.1.3 Relationship between GR Server and $\mathcal{LR}$ Server

From the definition of the $\mathcal{LR}$ server, it is easy to show that an $\mathcal{LR}$ server is also a GR server.

**Theorem 2.1.1.** *For any flow $f$, an $\mathcal{LR}$ server $S_i$ with latency $\theta^i_f$ is also a GR server with scheduling constant $\theta^i_f$.*

*Proof.* Without loss of generality, we consider only one busy period of flow $f$. Suppose the reserved rate for flow $f$ is $r_f$, and the busy period starts at time $\tau$ and ends at $\tau^*$. From the definition of GRC value in Eq. (2.1), it is easy to see that the GRC values of the packets in this busy period are:

$$GRC(p^k_f) = \frac{\sum_{n=1}^{k} \ell^n_f}{r_f} + \tau, \quad k \geq 1.$$

Suppose $t_k$ is the time when the $k^{th}$ packet leaves the server, then

$$W_f^i(\tau, t_k) = \sum_{n=1}^{k} \ell_f^n.$$

From the definition of $\mathcal{LR}$ server, we obtain

$$\sum_{n=1}^{k} \ell_f^n \geq \max\{0, r_f(t_k - \tau - \theta_f^i)\} \geq r_f(t_k - \tau - \theta_f^i).$$

Rearranging the terms, we obtain

$$t_k \leq \frac{\sum_{n=1}^{k} \ell_f^n}{r_f} + \tau + \theta_f^i = GRC(p_f^k) + \theta_f^i, \quad \forall k \geq 1. \tag{2.5}$$

The theorem is proven. □

Similarly, a GR server is also an $\mathcal{LR}$ server.

**Theorem 2.1.2.** *For any flow $f$, an GR server with scheduling constant $\beta_f$ is also a $\mathcal{LR}$ server with latency smaller than or equal to $\beta_f + \frac{\ell_f^{max}}{r_f}$.*

*Proof.* Similarly, without loss of generality, we consider only one busy period of flow $f$. Suppose the reserved rate for flow $f$ is $r_f$, and the busy period starts at time $\tau$ and ends at $\tau^*$. From the definition of GRC value, it is easy to see that the GRC values of the packets in this busy period are:

$$GRC(p_f^k) = \frac{\sum_{n=1}^{k} \ell_f^n}{r_f} + \tau, \quad k \geq 1.$$

First, consider the time instants when packets leave the server. Suppose the $k^{th}$ packet leaves the server at time $t_k$, $k \geq 1$, then from the definition of GR server, we obtain

$$t_k \leq GRC(p_f^k) + \beta_f, \quad k \geq 1,$$

$$\Longrightarrow t_k \leq \frac{\sum_{n=1}^{k} \ell_f^n}{r_f} + \tau + \beta_f, \quad k \geq 1,$$

$$\Longrightarrow W_f^i(\tau, t_k) = \sum_{n=1}^{k} \ell_f^n \geq r_f(t_k - \tau - \beta_f), \quad k \geq 1. \tag{2.6}$$

Next, consider time instant $t$ other than $t_k$ (*i.e.*, $t_k < t < t_{k+1}$ for all $k \geq 0$, assuming $t_0 = \tau$). Since $W_f^i(\tau,t)$ only increases when the last bit of a packet leaves the server, for any $k \geq 0$,

$$\begin{aligned} W_f^i(\tau,t) &= W_f^i(\tau,t_k) \\ &= W_f^i(\tau,t_{k+1}) - \ell_f^{k+1} \\ &\geq r_f(t_{k+1} - \tau - \beta_f - \frac{\ell_f^{k+1}}{r_f}) \\ &\geq r_f(t - \tau - (\beta_f + \frac{\ell_f^{max}}{r_f})), \quad t_k < t < t_{k+1}. \end{aligned} \tag{2.7}$$

Combining Eqs. (2.6) and (2.7), we obtain

$$W_f^i(\tau,t) \geq r_f(t - \tau - (\beta_f + \frac{\ell_f^{max}}{r_f})), \qquad \forall \tau \leq t \leq \tau^*.$$

Since $W_f^i(\tau,t) \geq 0$ for all $\tau \leq t \leq \tau^*$, we have

$$W_f^i(\tau,t) \geq \max\{0, r_f(t - \tau - (\beta_f + \frac{\ell_f^{max}}{r_f}))\}, \quad \forall \tau \leq t \leq \tau^*. \tag{2.8}$$

Then the theorem is proven. □

### 2.1.4 End-to-End Delay Bound under Per-Flow Scheduling

We now review the e2e delay bound results for per-flow scheduling. Both Lemma 2.1.2 and Theorem 2.1.3 stated below were proven in [4].

**Lemma 2.1.2.** *Suppose routers $S_i$ and $S_{i+1}$ are two neighboring GR servers on the path of flow $f$. If both routers guarantee service rate $r_f$ for flow $f$, then*

$$GRC^{i+1}(p_f^j) \leq GRC^i(p_f^j) + \frac{\ell_f^{max}}{r_f} + \alpha^i, \tag{2.9}$$

*where $\ell_f^{max}$ is the maximum packet size in flow $f$, and $\alpha^i = \beta^i + \tau^{i,i+1}$. Here $\tau^{i,i+1}$ is the propagation delay between $S_i$ and $S_{i+1}$.*

Lemma 2.1.2 states the relationship between the GRC values of a packet at two neighboring GR servers. Based on this relationship, the authors of [4] derived an e2e delay bound.

**Theorem 2.1.3.** *If flow $f$ conforms to the token bucket model $(\sigma_f, \rho_f)$ and all the routers on its path are GR servers with guaranteed rate $r_f \geq \rho_f$, then the e2e delay for $p_f^j$, $d_f^j$, is bounded as follows:*

$$d_f^j \leq \frac{\sigma_f}{r_f} + \frac{(K-1)\ell_f^{max}}{r_f} + \sum_{n=1}^{K} \alpha^n, \tag{2.10}$$

*where $\alpha^i = \beta^i + \tau^{i,i+1}$ and $K$ is the number of hops on the path of flow $f$.*

From Theorem 2.1.3, one can see that the delay bound is inversely proportional to the flow's guaranteed rate, but is proportional to the number of hops ($K$), the size of packets, and the burst size of the flow. With a large number of hops and large-size packets, the delay can be substantially large. To understand the physical meaning of Eq. (2.10), let us consider the fluid traffic model, where packet size is infinitely small. Then, if we omit the propagation delay, the delay bound in Eq. (2.10) can be simplified as:

$$d_f \leq \frac{\sigma_f}{r_f}. \tag{2.11}$$

In other words, the delay is upper bounded by the burstiness of the flow. The larger the burst size, the larger the delay bound. If we assume $\rho_f = r_f$, then the delay bound is decided by the *burst ratio* ($\frac{\sigma_f}{\rho_f}$ for flow $f$) of the flow.

In the next two sections, we derive e2e delay bounds under aggregate scheduling with GR scheduling algorithms. We show that if the incoming flows at aggregators conform to the token bucket model, the e2e delay is bounded.

Note that by using GR scheduling algorithms, we always assume that the guaranteed rate for a flow is greater than, or equal to, its average rate, *i.e.*, $r_f \geq \rho_f$. Also, in the remainder of this chapter, we omit the propagation delay $\tau^{i,i+1}$. Therefore, $\alpha^i = \beta^i$.

## 2.2 Work-Conserving Aggregator

To derive the e2e delay bound under aggregate scheduling, an important step is to derive the delay at the aggregator. We need to find the relationship between the GRC values at the aggregator and its next hop. In this section, we use work-conserving GR scheduling algorithms at aggregators. Two types of GR scheduling algorithms are examined: *stand-alone* (or non-hierarchical) and *hierarchical* algorithms.

### 2.2.1 Fair Aggregator

Before deriving the delay at the aggregator, we first introduce an important concept, the *fair aggregator*.

**Definition 2.2.1 (Fair Aggregator).** *Let router $S_i$ be an aggregator bundling flow $f$ and others into aggregate flow A, which, in turn, is an input to router $S_{i+1}$. Then if $p_f^j = p_A^{j'}$,* i.e., *the $j^{th}$ packet of flow $f$ corresponds to the $j'^{th}$ packet of aggregate flow A, $S_i$ is said to be a* fair aggregator *if and only if $S_i$ is a GR server and*

$$GRC^{i+1}(p_A^{j'}) \leq GRC^i(p_f^j) + \gamma^i + \alpha^i, \qquad \forall j \geq 1, \tag{2.12}$$

*where $\gamma^i$ is a constant that depends on the scheduling algorithm, the router, and other flows in the same aggregate. We call it an* aggregation constant.

The value $\gamma^i + \alpha^i$ can be considered as the *fairness index* of the aggregator, *e.g.*, an aggregator is considered fairer than others if it has a smaller value of $\gamma^i + \alpha^i$. For a fair aggregator, both of its aggregation constant ($\gamma^i$) and scheduling constant ($\alpha^i$) should be small. Also, our definition of *fair aggregator* is slightly different from that in [37, 38]. It is based on the relationship between a packet's GRC values at the aggregator and its next hop.

**Lemma 2.2.1.** *Suppose $S_i$ and $S_{i+1}$ are two neighboring GR servers and $S_i$ is an aggregator. $S_i$ aggregates flow $f$ and other (N-1) flows into aggregate flow A, which, in turn, is an input to $S_{i+1}$. Suppose incoming flow $k$ at $S_i$ conforms to the token bucket model $(\sigma_k, \rho_k)$ $(1 \leq k \leq N)$, and the guaranteed rate for flow $k$ at $S_i$ is $r_k$ $(1 \leq k \leq N)$. Let $p_f^j = p_A^{j'}$,* i.e., *the $j^{th}$ packet of flow $f$ corresponds to the $j'^{th}$ packet of aggregate flow A. If the outgoing flows at $S_i$ conform to the token bucket model $(\tilde{\sigma}_k, \rho_k)$ $(1 \leq k \leq N)$, then for packet $p_f^j$,*

$$GRC^{i+1}(p_A^{j'}) \leq GRC^i(p_f^j) + \frac{\sum_{k \neq f} \tilde{\sigma}_k + \ell_f^{max}}{R} + \alpha^i, \quad j \geq 1, \tag{2.13}$$

*where $R = \sum_{k=1}^{N} r_k$, which is the guaranteed rate for aggregate flow A at $S_{i+1}$.*

*Proof.* Similar to the definition of busy period of a single flow [36], we define a *busy period of aggregate flow A* at router $S_{i+1}$ as the maximum length of time period $(\tau_1, \tau_2]$, such that at any time

$t \in (\tau_1, \tau_2]$, the total traffic of $A$ arrived since the beginning of the interval is $A(\tau_1, t) \geq R \cdot (t - \tau_1)$. Without loss of generality, we consider only one busy period in the proof. Suppose the busy period starts at time $t_0$ when the first packet $p_A^1$ arrives, and in that busy period, time $t_j$ is the instant when packet $p_f^j$ arrives at $S_{i+1}$. Since $p_f^j = p_A^{j'}$, the total traffic arrival of aggregate $A$ up to time $t_j$ is

$$\begin{aligned}
\sum_{k=1}^{j'} \ell_A^k &= A(t_0, t_j) = \sum_{k=1}^{N} A_k(t_0, t_j) \\
&= A_f(t_0, t_j) + \sum_{k \neq f} A_k(t_0, t_j) \\
&\leq \sum_{m=1}^{j} \ell_f^m + \sum_{k \neq f} (\tilde{\sigma}_k + r_k \cdot (t_j - t_0)) \\
&= \sum_{m=1}^{j} \ell_f^m + \sum_{k \neq f} \tilde{\sigma}_k + (R - r_f)(t_j - t_0). \qquad (2.14)
\end{aligned}$$

Since all the packets arrive in the same busy period of aggregate $A$, from the definition of GRC value, we have

$$\begin{aligned}
GRC^{i+1}(p_A^{j'}) &= \frac{\sum_{k=1}^{j'} \ell_A^k}{R} + t_0 \\
&\leq \frac{\sum_{m=1}^{j} \ell_f^m + \sum_{k \neq f} \tilde{\sigma}_k + (R - r_f)(t_j - t_0)}{R} + t_0. \qquad (2.15)
\end{aligned}$$

Next, we prove that the theorem holds for both $j = 1$ and $j > 1$.

*Case 1*: $j = 1$.

$$\begin{aligned}
GRC^{i+1}(p_A^{j'}) &\leq \frac{\ell_f^1 + \sum_{k \neq f} \tilde{\sigma}_k + (R - r_f)(t_1 - t_0)}{R} + t_0 \\
&= \frac{\sum_{k \neq f} \tilde{\sigma}_k + \ell_f^1}{R} + t_1 - \frac{r_f \cdot (t_1 - t_0)}{R} \\
&\leq \frac{\sum_{k \neq f} \tilde{\sigma}_k + \ell_f^{max}}{R} + t_1.
\end{aligned}$$

Since $S_i$ is a GR server for flow $f$, we have $t_1 \leq GRC^i(p_f^1) + \alpha^i$. Therefore,

$$GRC^{i+1}(p_A^{j'}) \leq GRC^i(p_f^1) + \frac{\sum_{k \neq f} \tilde{\sigma}_k + \ell_f^{max}}{R} + \alpha^i.$$

*Case 2*: $j > 1$. From the definition of GRC value, for flow $f$, we have

$$\begin{aligned} GRC^i(p_f^2) &\geq GRC^i(p_f^1) + \frac{\ell_f^2}{r_f}, \\ GRC^i(p_f^3) &\geq GRC^i(p_f^2) + \frac{\ell_f^3}{r_f}, \\ &\vdots \\ GRC^i(p_f^j) &\geq GRC^i(p_f^{j-1}) + \frac{\ell_f^j}{r_f}. \end{aligned}$$

Adding them up, we obtain

$$\begin{aligned} & GRC^i(p_f^j) - GRC^i(p_f^1) \geq \frac{\sum_{m=2}^{j} \ell_f^m}{r_f} \\ \Longrightarrow & \sum_{m=1}^{j} \ell_f^m \leq \ell_f^1 + r_f \cdot (GRC^i(p_f^j) - GRC^i(p_f^1)). \end{aligned} \tag{2.16}$$

Then, from Eqs. (2.15) and (2.16), and the fact $t_j \leq GRC^i(p_f^j) + \alpha^i$, we obtain

$$\begin{aligned} GRC^{i+1}(p_A^{j'}) &\leq \frac{\ell_f^1 + r_f \cdot (GRC^i(p_f^j) - GRC^i(p_f^1))}{R} + \frac{\sum_{k \neq f} \tilde{\sigma}_k + (R - r_f)(t_j - t_0)}{R} + t_0 \\ &\leq \frac{\sum_{k \neq f} \tilde{\sigma}_k + \ell_f^1}{R} + \frac{r_f(GRC^i(p_f^j) - GRC^i(p_f^1))}{R} + \frac{(R - r_f)(GRC^i(p_f^j) + \alpha^i - t_0) + R \cdot t_0}{R} \\ &= GRC^i(p_f^j) + \frac{\sum_{k \neq f} \tilde{\sigma}_k + \ell_f^1}{R} + \alpha^i - \frac{r_f(GRC^i(p_f^1) + \alpha^i - t_0)}{R} \\ &\leq GRC^i(p_f^j) + \frac{\sum_{k \neq f} \tilde{\sigma}_k + \ell_f^1}{R} + \alpha^i - \frac{r_f(t_1 - t_0)}{R} \\ &\leq GRC^i(p_f^j) + \frac{\sum_{k \neq f} \tilde{\sigma}_k + \ell_f^{max}}{R} + \alpha^i. \end{aligned}$$

The lemma is proven. □

Next, we study the stand-alone aggregator, which uses stand-alone GR scheduling algorithm to bundle flows into aggregates.

### 2.2.2 Delay Bound I: Stand-Alone Aggregator

Combining Lemmas 2.1.1 and 2.2.1, we obtain the following theorem on the relationship between the GRC values at a stand-alone aggregator and its next hop.

**Theorem 2.2.1.** *Suppose $S_i$ is an LR server, and both $S_i$ and $S_{i+1}$ are GR servers. As a stand-alone aggregator, $S_i$ aggregates flow $f$ and other (N-1) flows into aggregate A, which, in turn, is an input to router $S_{i+1}$. Suppose incoming flow k at $S_i$ conforms to the token bucket model $(\sigma_k, \rho_k)$ $(1 \leq k \leq N)$, and the guaranteed rate for flow k at router $S_i$ is $r_k$ $(1 \leq k \leq N)$. Let $p_f^j = p_A^{j'}$,* i.e., *the $j^{th}$ packet of flow $f$ corresponds to the $j'^{th}$ packet of aggregate A. Then, $S_i$ is a fair aggregator with $\gamma^i = \frac{\sum_{k \neq f} \sigma_k}{R} + \frac{\sum_{k \neq f} \theta_k^i \cdot \rho_k + \ell_f^{max}}{R}$. In other words,*

$$GRC^{i+1}(p_A^{j'}) \leq GRC^i(p_f^j) + \alpha^i + [\frac{\sum_{k \neq f} \sigma_k}{R} + \frac{\sum_{k \neq f} \theta_k^i \cdot \rho_k + \ell_f^{max}}{R}], \quad j \geq 1. \tag{2.17}$$

The proof is trivial.

Now, we are ready to derive the e2e delay bound for aggregate scheduling under the token bucket model and the stand-alone aggregator.

**Theorem 2.2.2.** *Suppose N flows share the same K hops of GR servers inside an aggregation region, and they are bundled into aggregate A at stand-alone aggregator $S_1$ and split back at $S_K$. Routers $S_2, \ldots, S_{K-1}$ schedule packets of aggregate A. If flow k conforms to the token bucket model $(\sigma_k, \rho_k)$ and has the guaranteed rate $r_k$ with $r_k \geq \rho_k$ $(1 \leq k \leq N)$ at $S_1$ and $S_K$, and A has the guaranteed rate $R = \sum_{k=1}^{N} r_k$ at $S_2, \ldots, S_{K-1}$, then for any flow $f$ $(1 \leq f \leq N)$, the e2e delay of packet $p_f^j$, $d_f^j$, is bounded as follows:*

$$d_f^j \leq \frac{\sigma_f}{r_f} + [\frac{\sum_{k \neq f} \sigma_k}{R} + \frac{\sum_{k \neq f} \theta_k^1 \cdot r_k + \ell_f^{max}}{R}] + (K-3)\frac{\ell_A^{max}}{R} + \frac{\ell_f^{max}}{r_f} + \sum_{i=1}^{K} \alpha^i, \quad j \geq 1. \tag{2.18}$$

*Proof.* Let $p_A^{j'}$ be the packet in the aggregate flow $A$ corresponding to $p_f^j$ in flow $f$. From Theorem 2.2.1, we have

$$GRC^2(p_A^{j'}) \leq GRC^1(p_f^j) + \gamma^1 + \alpha^1,$$

where $\gamma^1 = \frac{\sum_{k\neq f}\sigma_k}{R} + \frac{\sum_{k\neq f}\theta_k^1\cdot\rho_k+\ell_f^{max}}{R}$. Since $S_2$, ..., $S_{K-1}$ are all GR servers for aggregate flow $A$ with guaranteed rate $R = \sum_{k=1}^{N} r_k$, from Lemma 2.1.2, we have

$$\begin{aligned}
GRC^3(p_A^{j'}) &\leq GRC^2(p_A^{j'}) + \frac{\ell_A^{max}}{R} + \alpha^2, \\
GRC^4(p_A^{j'}) &\leq GRC^3(p_A^{j'}) + \frac{\ell_A^{max}}{R} + \alpha^3, \\
&\vdots \\
GRC^{K-1}(p_A^{j'}) &\leq GRC^{K-2}(p_A^{j'}) + \frac{\ell_A^{max}}{R} + \alpha^{K-2}
\end{aligned}$$

Adding them all up, we obtain

$$GRC^{K-1}(p_A^{j'}) \leq GRC^1(p_f^j) + (K-3)\frac{\ell_A^{max}}{R} + \gamma^1 + \sum_{i=1}^{K-2}\alpha^i.$$

Since $p_f^j = p_A^{j'}$, for packet $p_f^j$, the departure time at $S_{K-1}$ is

$$\begin{aligned}
D^{K-1}(p_f^j) &= D^{K-1}(p_A^{j'}) \\
&\leq GRC^{K-1}(p_A^{j'}) + \alpha^{K-1} \\
&\leq GRC^1(p_f^j) + \gamma^1 + (K-3)\frac{\ell_A^{max}}{R} + \sum_{i=1}^{K-1}\alpha^i.
\end{aligned}$$

From the definition of the GR server, the first ($K$-1) servers for flow $f$ can be viewed as a virtual GR server with scheduling constant $\alpha^* = \gamma^1 + (K-3)\frac{\ell_A^{max}}{R} + \sum_{i=1}^{K-1}\alpha^i$. Since $S_K$ is also a GR server for flow $f$ with guaranteed rate $r_f$, from Lemma 2.1.2, we have

$$GRC^K(p_f^j) \leq GRC^1(p_f^j) + \frac{\ell_f^{max}}{r_f} + \alpha^*.$$

Therefore, the departure time of packet $p_f^j$ from $S_K$ is

$$\begin{aligned}
D^K(p_f^j) &\leq GRC^K(p_f^j) + \alpha^K \\
&\leq GRC^1(p_f^j) + \frac{\ell_f^{max}}{r_f} + \alpha^* + \alpha^K \\
&= GRC^1(p_f^j) + [\frac{\sum_{k\neq f}\sigma_k}{R} + \frac{\sum_{k\neq f}\theta_k^1\cdot r_k + \ell_f^{max}}{R}] + (K-3)\frac{\ell_A^{max}}{R} + \frac{\ell_f^{max}}{r_f} + \sum_{i=1}^{K}\alpha^i.
\end{aligned}$$

Then, the e2e delay $d_f^j$ satisfies

$$\begin{aligned} d_f^j &= D^K(p_f^j) - A^1(p_f^j) \\ &\leq [GRC^1(p_f^j) - A^1(p_f^j)] + (K-3)\frac{\ell_A^{max}}{R} + \frac{\ell_f^{max}}{r_f} + [\frac{\sum_{k\neq f}\sigma_k}{R} + \frac{\sum_{k\neq f}\theta_k^1 \cdot r_k + \ell_f^{max}}{R}] + \sum_{i=1}^{K}\alpha^i. \end{aligned} \tag{2.19}$$

From [4], for flow $f$ conforming to the token bucket model $(\sigma_f, \rho_f)$ and with reserved rate $r_f (r_f \geq \rho_f)$, $GRC^1(p_f^j) - A^1(p_f^j) \leq \frac{\sigma_f}{r_f}$. Thus the theorem is proven. □

As in Section 2.1, to further understand the physical meaning of Eq. (2.18), we consider the fluid traffic model, where packet size is infinitely small. Then, Eq. (2.18) can be simplified as (since in general the term $\theta_k^1$ also relies on packet size and will become infinitely small):

$$d_f \leq \frac{\sigma_f}{r_f} + \frac{\sum_{k\neq f}\sigma_k}{R}. \tag{2.20}$$

Note that Eq. (2.20) provides the lower limit of the e2e delay bound. Comparing Eqs. (2.11) and (2.20), we can see that with aggregate scheduling, the delay bound of a flow is not only decided by the burstiness of the flow itself (term $\frac{\sigma_f}{r_f}$), but also strongly related to the burstiness of other flows that share the same aggregate with it (term $\frac{\sum_{k\neq f}\sigma_k}{R}$). Intuitively, this is easy to understand, since a packet from flow $f$ has to wait behind not only the earlier packets from the same flow, but also those from other flows sharing the same aggregate. The result implies how aggregation should be done—a flow should not be aggregated with other flows with substantially larger burst ratios. This is similar to the conclusion in [23]. This issue will be explored further in Section 2.4.

From Eq. (2.18), we can see that the delay bound is affected by the latency of the scheduling algorithm at the aggregator. To get a smaller bound, we should use a low-latency scheduling algorithm. For example, with PGPS and VC ($\theta_f^1 = \frac{\ell_f^{max}}{r_f} + \frac{L_{max}^1}{C^1}$) at the aggregator $S_1$, the e2e delay is bounded as follows:

$$d_f^j \leq \frac{\sigma_f}{r_f} + [\frac{\sum_{k\neq f}\sigma_k}{R} + \frac{\sum_{k=1}^{N}\ell_k^{max}}{R}] + (K-3)\frac{\ell_A^{max}}{R} + \frac{\ell_f^{max}}{r_f} + \frac{L_{max}^1}{C^1} + \sum_{i=1}^{K}\alpha^i. \tag{2.21}$$

Finally, comparing Eqs. (2.10) and (2.18), we note that, depending on the burst ratios of the constituent flows and the maximum packet size in the aggregate, the delay bound under aggregate

scheduling can be smaller than that under per-flow scheduling. We will compare the delay bounds numerically later in Section 2.4.

In the next subsection, we further decrease the delay bound by using hierarchical GR scheduling algorithms at aggregators.

### 2.2.3 Delay Bound II: Hierarchical Aggregator

So far, the stand-alone GR scheduling algorithms have been used at aggregators. Thus, when computing the burstiness of an outgoing aggregate at aggregators, the burst size of each constituent flow in the aggregate was computed separately and then summed up. In this subsection, we use hierarchical GR algorithms at aggregators to improve the delay bound, as shown in Fig. 2.1. First, the flows that end up in the same aggregate are grouped together at the lower-level schedulers. Then, the aggregates are scheduled at the upper-level scheduler. This way, we can reduce the burstiness of the outgoing aggregate and the aggregation constant at aggregators, thus reducing the e2e delay.

How does hierarchical scheduler reduce the burstiness of the outgoing aggregates? Fig. 2.2 shows the intuition behind it. Suppose there are 20 flows coming into one aggregator, and all of them have the same reserved rate. The first 10 flows belong to one aggregate flow $A_1$, and the last 10 belong to another aggregate flow $A_2$. Suppose 20 packets of identical size arrive at the idle aggregator at exactly the same time, one from each flow. Then, with a stand-alone scheduler, the 20 packets will be scheduled in a random order. Thus, it is possible that all the 10 packets of $A_1$ are scheduled before all the packets of aggregate flow $A_2$. Therefore, the output of the aggregator will look like: 10 packets of aggregate $A_1$ come out first, then 10 packets of aggregate $A_2$. For the next router which is handling aggregates, the burstiness of both $A_1$ and $A_2$ is very high. In contrast, with hierarchical scheduler, the upper-level scheduler will make sure the packets from $A_1$ and $A_2$ are scheduled alternately, making the traffic in both aggregate flows smoother.

The hierarchical scheduling algorithms under consideration are two-level *hierarchical packet fair queueing* (H-PFQ) algorithms [39].

As defined in [39], flow $f$ arriving at a H-PFQ of height $H$ has $H$ ancestors in the tree, with $p(f)$ as its immediate parent node, and $p^h(f)$ is the parent node of $p^{h-1}(f)$ $(h = 1, \ldots, H)$. Here $p^0(f) = f, p^1(f) = p(f), \ldots$, and $p^H(f)$ = link capacity $C$. Since we only need two-level hierarchy, $H = 2$.

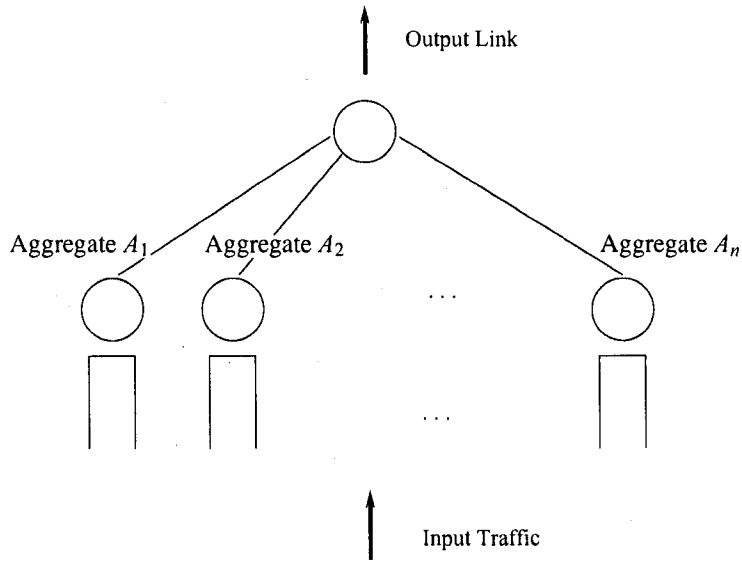



Figure 2.1: Hierarchical scheduling

Before deriving the delay bounds, we first repeat the definition of *Bit Worst-case Fair Index* (B-WFI) below. Note that some notations in the definition are modified for consistency.

**Definition 2.2.2 (Bit Worst-case Fair Index (B-WFI)).** *A server node $S_i$ is said to guarantee a Bit Worst-case Fair Index (B-WFI) of $\eta_{i,f}$ for flow $f$, if for any packet $p_f^j$ the following holds*

$$W_f^i(t, d_f^j) \geq \frac{\phi_f}{\Phi} W^i(t, d_f^j) - \eta_{i,f} \tag{2.22}$$

*where $d_f^j$ is the time $p_f^j$ departs the server, $t$ is any time instant such that $t < d_f^j$ and session $f$ is continuously backlogged during $[t, d_f^j]$, and $\frac{\phi_f}{\Phi}$ is the service share guaranteed to flow $f$ by server $S_i$.*

Next, we show that H-PFQ also belongs to the GR class.

**Fact 2.2.1.** *If a stand-alone packet fair queueing (PFQ) algorithm belongs to the GR class with the scheduling constant $\beta_{PFQ}^i$, then its corresponding H-PFQ of height $H$ also belongs to the GR class with scheduling constant:*

$$\beta_{HPFQ}^i \leq \beta_{PFQ}^i + \sum_{h=1}^{H-1} \frac{\eta_{p^h(f),f}}{r_{p^h(f)}}, \tag{2.23}$$

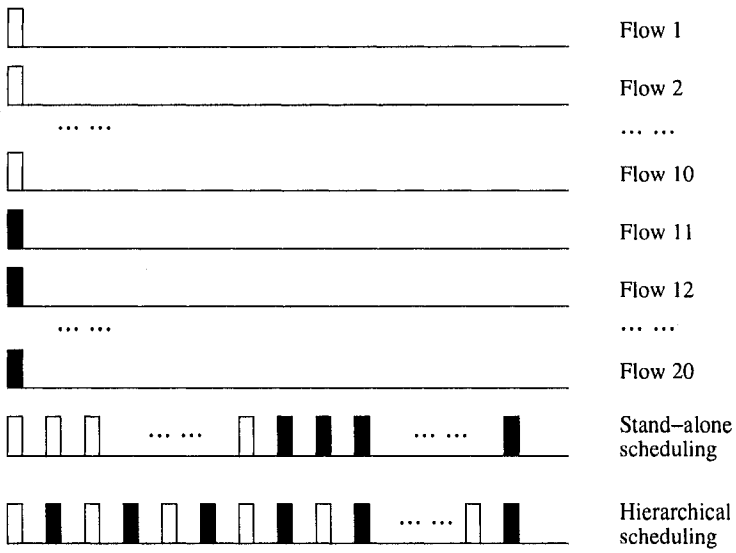



Figure 2.2: Benefits of hierarchical scheduling

*where* $\eta_{p^h(f),f}$ *and* $r_{p^h(f)}$ *are the* Bit Worst-case Fair Index *(B-WFI) [39] of the logical queue at node* $p^h(f)$ *and the guaranteed rate at node* $p^h(f)$*, respectively.*

*Proof.* From Theorem 2 of [39], the delay bound for a flow $f$ at H-PFQ is at most $\sum_{h=1}^{H-1} \frac{\eta_{p^h(f),f}}{r_{p^h(f)}}$ larger than that at the corresponding stand-alone PFQ server. Therefore, $\beta^i_{HPFQ} \leq \beta^i_{PFQ} + \sum_{h=1}^{H-1} \frac{\eta_{p^h(f),f}}{r_{p^h(f)}}$. □

Now, we derive the burstiness of the outgoing traffic at aggregators. For any flow $f$, we derive $\sum_{k \neq f} \tilde{\sigma}_k$—the total outgoing burst size of all the other flows sharing the same aggregate with $f$. In the derivation, we make use of the corresponding fluid GPS (Generalized Processor Sharing) server of a PFQ (Packet Fair Queueing) server Let $W^i_k(\tau,t)$ ($W^i_{k,GPS}(\tau,t)$) be the service received by flow $k$ at the PFQ server $S_i$ (the corresponding GPS fluid server) during $(\tau,t]$, and define $\lambda^i_k$ and $\delta^i_k$ as:

$$\lambda^i_k = \max_{t \geq 0} \{W^i_{k,GPS}(0,t) - W^i_k(0,t)\}, \tag{2.24}$$

$$\delta^i_k = \max_{t \geq 0} \{W^i_k(0,t) - W^i_{k,GPS}(0,t)\}. \tag{2.25}$$

Intuitively, $\lambda^i_k$ and $\delta^i_k$ define the maximum difference between the amount of services flow $k$ gets

from the corresponding GPS server and the PFQ server $S_i$ in any time interval: $\lambda_k^i$ ($\delta_k^i$) defines how much more (less) service flow $k$ can get from the corresponding GPS server than the PFQ server.

**Theorem 2.2.3.** *Suppose $S_i$ is an aggregator with a two-level H-PFQ scheduler and $S_i$ aggregates flow $f$ and other (N-1) flows into aggregate A. Also, suppose flow $k$ ($1 \leq k \leq N$) at $S_i$ conforms to the token bucket model $(\sigma_k, \rho_k)$ and has guaranteed rate $r_k$. Then, we have*

$$\sum_{k \neq f} \tilde{\sigma}_k \leq \sum_{k \neq f} \sigma_k + [\lambda_f^i + \delta_f^i] + (1 - \frac{r_f}{R})[\lambda_A^i + \delta_A^i]. \tag{2.26}$$

The proof is similar to the proof of Theorem 3 in [36]. We first prove the total backlog size of the ($N$-1) flows is upper bounded, then derive the burst size from the backlog size. The details of the proof can be found in Appendix A.2.

To get a small burst size, we should use PFQ with small $\lambda_k^i$ and $\delta_k^i$, especially small $\delta_k^i$. From Theorem 1 of [40], WF$^2$Q (Worst-case Fair Weighted Fair Queueing) yields $\lambda_k^i = L_{max}^i$ and a very small $\delta_k^i = (1 - \frac{r_k}{C^i})\ell_k^{max}$ for flow $k$ at server $S_i$ with capacity $C^i$. Therefore, if H-WF$^2$Q is used at the aggregator, the burst size becomes:

$$\begin{aligned} \sum_{k \neq f} \tilde{\sigma}_k &\leq \sum_{k \neq f} \sigma_k + [\ell_A^{max} + (1 - \frac{r_f}{R})\ell_f^{max}] + (1 - \frac{r_f}{R})[L_{max}^i + (1 - \frac{R}{C^i})\ell_A^{max}] \\ &\leq \sum_{k \neq f} \sigma_k + L_{max}^i + \ell_f^{max} + 2\ell_A^{max}. \end{aligned} \tag{2.27}$$

**Corollary 2.2.1.** *Using the H-WF$^2$Q algorithm at the aggregator, and under the same condition of Theorem 2.2.2, the e2e delay $d_f^j$ is bounded as follows:*

$$d_f^j \leq \frac{\sigma_f}{r_f} + \frac{\Sigma_{k \neq f}\sigma_k + 2(\ell_f^{max} + L_{max}^1) + (K-1)\ell_A^{max}}{R} + \frac{\ell_f^{max}}{r_f} + \sum_{i=1}^{K} \alpha^i. \tag{2.28}$$

The proof is similar to that of Theorem 2.2.2. Refer to the Appendix A.1 for details.

In general, the bound in Eq. (2.28) is smaller than that in Eq. (2.21), especially when the number of flows ($N$) is large. However, as can be seen, the bound in Eq. (2.28) is still affected by the average burst ratio of other flows in the same aggregate.

### 2.2.4 Multiple Aggregations in a Region

The derivation of the two delay bounds (using work-conserving aggregators) in this section relies on the fact that the incoming traffic conforms to the token-bucket model with known parameters. Inside an aggregation region, the traffic pattern (the burst size in particular) will be distorted and become unpredictable. If further aggregations are done inside an aggregation region, this change of traffic pattern makes the derivation of delay bounds very difficult, if not impossible. In fact, we derived the two delay bounds assuming that aggregation is done only once in each aggregation region. Thus, in the case of multiple aggregations in the region, they are not true e2e delay bounds, but rather delay bounds between two nodes (between which the traffic of interest experiences aggregation and split only once) along the e2e path. To use our method to derive e2e delay bounds in the case of multiple aggregations in a region, the traffic has to be reshaped before every aggregator to make it follow the token-bucket model. The shaping incurs an extra delay, which is not considered here, though.

## 2.3 Non-Work-Conserving Aggregator

In the previous section, work-conserving scheduling algorithms were used at aggregators, and the derivation of the delay bounds required to know the incoming traffic pattern at the aggregator. This requirement limits the ability of multiple aggregations within an aggregation region, since the traffic pattern inside the region is difficult to predict. The authors of [41] proved that reshaping a flow inside the network will not change its e2e delay bound. However, their proof was done for a single flow, and it is not clear if reshaping an aggregate flow will change the e2e delay bound of a constituent flow of the aggregate. We conjecture that the delay bound will be affected.

Cobb [37, 38] overcame this difficulty by using non-work-conserving scheduling algorithms at aggregators. As stated in [42, 43], non-work-conserving scheduling algorithms have the following features: (i) the burstiness of traffic can be controlled; (ii) the sum of the per-hop bounds can tightly bound the e2e delay and jitter; and (iii) since the non-work-conserving scheduler shapes the traffic at each hop, it is easy to bound the performance of heterogeneous networks. Both the *basic fair aggregator* and *greedy fair aggregator* in [38] use non-work-conserving scheduling algorithms at aggregators to shape the outgoing traffic aggregate. This way, very few packets in the same aggregate queue up at later hops, which, in turn, makes the queueing delays at those hops very small. However, the implicit assumption in [38] is that there is only one output aggregate from

each aggregator, which is not generally the case. In this section, we extend the result in [38] by allowing multiple outgoing aggregates from an aggregator and derive the e2e delay bound under the token bucket traffic model.

### 2.3.1 Delay Bound III: Non-Work-Conserving Aggregator

First, we define a new type of fair aggregator—*rate-controlled fair aggregator.*

**Definition 2.3.1 (Rate-Controlled Fair Aggregator).** *Server $S_i$ is said to be a* rate-controlled fair aggregator *if (i) it is a two-level hierarchical GR scheduler; (ii) each lower-level scheduler handles the flows belonging to one aggregate with capacity R, which is the sum of the guaranteed rates for all the constituent flows of the aggregate; and (iii) the upper level is the scheduler for all the aggregates.*

Note that the hierarchical scheduling algorithm defined here is different from the one used in Section 2.2.3: it is non-work-conserving, and the lower-level consists of constant-rate servers. A packet at a lower-level scheduler will not be sent to the upper-level scheduler if the capacity of that scheduler does not permit it, even when the upper-level scheduler is idle. In contrast, in the ordinary H-PFQ, the lower-level schedulers are variable-rate servers [39]. Since the lower-level schedulers are constant-rate servers, we will regard the lower-level schedulers and the upper-level scheduler as two virtual hops.

**Lemma 2.3.1.** *Suppose server $S_i$ is a rate-controlled fair aggregator, with one of the lower-level schedulers, $S_{i,l}$, serving N incoming flows (flow f is one of them). The output of $S_{i,l}$ is the aggregate A. $S_{i,h}$ is the upper-level scheduler dealing with all of the aggregates. Suppose at $S_{i,l}$, these N flows have guaranteed rates $r_k$, $1 \leq k \leq N$, respectively, and at $S_{i,h}$ the guaranteed rate for A is $R = \sum_{k=1}^{N} r_k$. Let the $j^{th}$ packet of A correspond to the $j'^{th}$ packet of flow f (*i.e.*, $p_A^j = p_f^{j'}$). Then, we have*

$$GRC^{i,h}(p_A^j) \leq GRC^{i,l}(p_f^{j'}) + \alpha^{i,l}, \tag{2.29}$$

*where $\alpha^{i,l}$ is the scheduling constant at $S_{i,l}$ (with capacity $R = \sum_{k=1}^{N} r_k$).*

*Proof.* Without loss of generality, we consider only one busy period of aggregate $A$ at $S_{i,h}$. Since the lower-level server $S_{i,l}$ is rate-controlled with capacity $R$, the start times of the transmission of

two consecutive packets ($p_A^j$ and $p_A^{j+1}$) in the busy period are separate by a interval of $\frac{\ell_A^j}{R}$. Also, since $S_{i,l}$ and $S_{i,h}$ are two virtual nodes, the virtual link capacity between them is infinite, *i.e.*, $C \rightarrow \infty$. Thus the transmission time of a packet is infinitely small. Therefore, we have

$$A^{i,h}(p_A^j) = \begin{cases} A^{i,l}(p_A^j), & j = 1 \\ A^{i,h}(p_A^{j-1}) + \frac{\ell_A^{j-1}}{R}, & j > 1. \end{cases} \tag{2.30}$$

Next, we prove that for all $j \geq 1$,

$$GRC^{i,h}(p_A^j) = A^{i,h}(p_A^j) + \frac{\ell_A^j}{R}. \tag{2.31}$$

We prove this by induction on $j$.

*Base Case:* $j = 1$. Since it is the first packet of the busy period, we have

$$GRC^{i,h}(p_A^1) = A^{i,h}(p_A^1) + \frac{\ell_A^1}{R}.$$

*Induction Hypothesis:* Suppose Eq. (2.31) holds for $1 \leq j \leq m$.

*Induction Step:* $j = m+1$. From the definition of GRC value, we have

$$GRC^{i,h}(p_A^{m+1}) = \max\{A^{i,h}(p_A^{m+1}), GRC^{i,h}(p_A^m)\} + \frac{\ell_A^{m+1}}{R}.$$

However, from Eq. (2.30) and the *Induction Hypothesis* step, we know

$$A^{i,h}(p_A^{m+1}) = A^{i,h}(p_A^m) + \frac{\ell_A^m}{R} = GRC^{i,h}(p_A^m),$$

$$\Longrightarrow \quad GRC^{i,h}(p_A^{m+1}) = A^{i,h}(p_A^{m+1}) + \frac{\ell_A^{m+1}}{R}.$$

Also, since $S_{i,l}$ is a GR server, by definition we have

$$A^{i,h}(p_A^j) + \frac{\ell_A^j}{R} \leq GRC^{i,l}(p_A^j) + \alpha^{i,l}, \quad \forall j \geq 1.$$

Since $p_A^j = p_f^{j'}$ and thus $\ell_A^j = \ell_f^{j'}$, from Eq. (2.31) we obtain

$$GRC^{i,h}(p_A^j) \leq GRC^{i,l}(p_f^{j'}) + \alpha^{i,l}, \quad \forall j \geq 1.$$

□

From Lemma 2.3.1, one can easily see that the rate-controlled fair aggregator is a fair aggregator. Next, we derive the e2e delay bound and show that by using non-work-conserving scheduling algorithms at the aggregator, the e2e delay bound of aggregate scheduling can be decreased.

**Theorem 2.3.1.** *Suppose flow $f$ traverses $K$ hops of GR servers $S_1, S_2, \ldots, S_K$, and $S_1$ is a rate-controlled fair aggregator, which bundles $f$ and other (N-1) flows into aggregate flow $A$. $S_K$ is the deaggregator for $A$. Suppose flow $k$ has a guaranteed rate $r_k$ ($1 \leq k \leq N$) at $S_1$ and $S_K$, and the guaranteed rate for aggregate $A$ at $S_2, S_3, \ldots, S_{K-1}$ is $R = \sum_{k=1}^{N} r_k$. Then, for packet $p_f^j$, the e2e delay $d_f^j$ satisfies:*

$$d_f^j \leq [GRC^{1,I}(p_f^j) - A^1(p_f^j)] + (K-2)\frac{\ell_A^{max}}{R} + \alpha^{1,I} + \frac{\ell_f^{max}}{r_f} + \sum_{i=1}^{K} \alpha^i. \tag{2.32}$$

The proof is similar to that of Theorem 2.2.2. Refer to Appendix A.3 for a detailed proof.

**Corollary 2.3.1.** *If flow $f$ conforms to the token bucket model $(\sigma_f, \rho_f)$ and $\rho_f \leq r_f$, the e2e delay result of Theorem 2.3.1 becomes:*

$$d_f^j \leq \frac{\sigma_f}{r_f} + (K-2)\frac{\ell_A^{max}}{R} + \frac{\ell_f^{max}}{r_f} + \alpha^{1,I} + \sum_{i=1}^{K} \alpha^i. \tag{2.33}$$

Note that thanks to the rate-control mechanism at aggregators, the e2e bound does not depend on the burstiness of other flows in the same aggregate. Thus, the bound is smaller than those of the work-conserving cases. If scheduling algorithms, such as PGPS, VC, and WF$^2$Q, are used at the rate-controlled fair aggregator $S_1$, we have $\alpha^{1,I} = \frac{\ell_A^{max}}{R}$. Then Eqs. (2.32) and (2.33) can be simplified as

$$d_f^j \leq [GRC^{1,I}(p_f^j) - A^1(p_f^j)] + (K-1)\frac{\ell_A^{max}}{R} + \frac{\ell_f^{max}}{r_f} + \sum_{i=1}^{K} \alpha^i \tag{2.34}$$

and

$$d_f^j \leq \frac{\sigma_f}{r_f} + (K-1)\frac{\ell_A^{max}}{R} + \frac{\ell_f^{max}}{r_f} + \sum_{i=1}^{K} \alpha^i, \tag{2.35}$$

respectively.

In addition, the derivation of the bound in Eq. (2.32) does not require the knowledge of the traffic pattern of incoming flows. This enables us to derive the delay bound for the case of multiple

aggregations. For simplicity, in the following discussion, we use the term $GRC^i(p_f^j)$ to represent $GRC^{i,I}(p_f^j)$. Also, we assume that scheduling algorithms such as PGPS, VC, and WF$^2$Q are used at the rate-controlled fair aggregators. Therefore, the delay bound for single aggregation is in the form of Eq. (2.34).

### 2.3.2 Multiple Aggregations

**Theorem 2.3.2.** *Suppose flow $f$ traverses $K$ hops of GR servers. Any hop can be an aggregator and all the aggregators are rate-controlled fair aggregators. For each aggregator, there is a corresponding deaggregator at a later hop. Then, the e2e delay for packet $p_f^j$, $d_f^j$, satisfies:*

$$d_f^j \le [GRC^1(p_f^j) - A^1(p_f^j)] + \sum_{i=2}^{K} \frac{\ell_{\hat{A}_i}^{max}}{\hat{R}_i} + \sum_{i=1}^{M} \frac{\ell_{A_i}^{max}}{R_i} + \sum_{n=1}^{K} \alpha^n, \tag{2.36}$$

*where $M$ is the number of aggregators along the path; $\hat{A}_i$ is the aggregate flow that contains flow $f$ at $S_i$, $\ell_{\hat{A}_i}^{max}$ is the maximum packet size in aggregate flow $\hat{A}_i$, and $\hat{R}_i$ is the guaranteed rate for $\hat{A}_i$ at $S_i$; $A_i$ is the $i^{th}$ aggregate along the path that contains flow $f$, and $\ell_{A_i}^{max}$ and $R_i$ are the maximum packet size in the aggregate and its guaranteed rate, respectively.*

To prove the theorem, we first prove two lemmas that consider two basic cases of multiple aggregations: pure recursive aggregation and pure sequential aggregation. Refer to Appendix A.4 for a detailed proof.

Note that there are a total of ($K$ - 1) + $M$ terms related to packet size in the delay bound above. They can be understood as follows: the ($K$-1) terms in $\sum_{i=2}^{K} \frac{\ell_{\hat{A}_i}^{max}}{\hat{R}_i}$ correspond to the delays at all the hops (except the first hop). In addition, for each aggregation with guaranteed rate $R_i$, there is a term $\frac{\ell_{A_i}^{max}}{R_i}$ as overhead. Compared to the delay bound for per-flow scheduling in Eq. (2.10), Eq. (2.36) has $M$ more terms due to aggregation. However, since the guaranteed rates for aggregate flows are much higher at the routers in the aggregation region, the total delay bound in Eq. (2.36) can be smaller than that in Eq. (2.10). Moreover, if there is only one aggregate ($M$=1), and $S_1$ and $S_K$ are the aggregator and deaggregator, respectively, Eq. (2.36) becomes just Eq. (2.34).

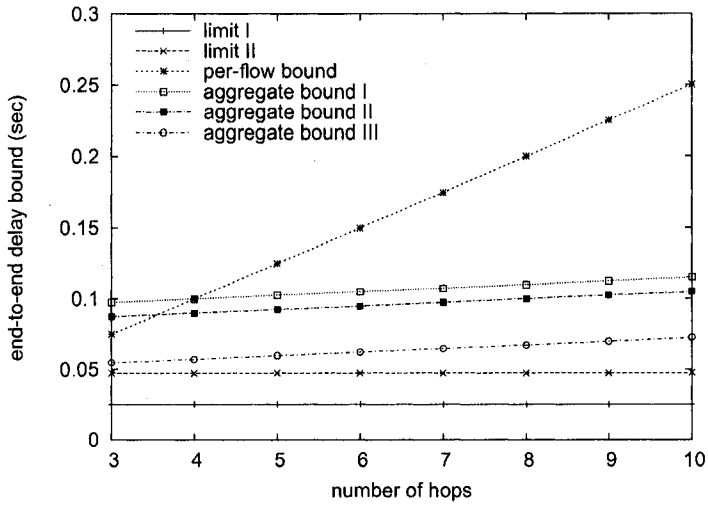



Figure 2.3: Comparison of deterministic delay bounds

## 2.4 Evaluation

In this section, we first use some sample data to numerically calculate the deterministic delay bounds derived so far, and compare them with that of per-flow scheduling. Then, we perform extensive simulations to compare the e2e delays of aggregate and per-flow scheduling.

First, the deterministic bounds were compared using the data in Table 1 of [9]: $\sigma_f = 100\text{B}$, $r_f = 32Kb/s$, $C^i = 150Mb/s$. All packets were of the same size, 100B, and 10 identical flows made up an e2e aggregate. To see the effect of the number of hops, we varied $K$ from 3 to 10. As shown in Fig. 2.3, all the bounds increase linearly with the number of hops, but those for aggregate scheduling increase much more slowly. When the number of hops is large, the delay bounds of aggregate scheduling are smaller than that of per-flow scheduling. The *per-flow bound* is computed from Eq. (2.10); the *aggregate bound* I, II and III are computed from Eqs. (2.21), (2.28) and (2.35), respectively; and the *limit I* and *II* are computed from $\frac{\sigma_f}{r_f}$ and $\frac{\sigma_f}{r_f} + \frac{\sum_{k \neq f} \sigma_k}{\sum_k r_k}$, respectively, which are independent of the number of hops.

From this example, one can see that if the degree of burstiness of the flows is not very high, the delay bounds of aggregate scheduling can be smaller than that of per-flow scheduling. In addition, it shows that hierarchical aggregator and rate-controlled hierarchical aggregator can provide even smaller delay bounds. Moreover, packet size and hop count play a big role in the delay bound under per-flow scheduling, which is much larger than $\frac{\sigma_f}{r_f}$. In contrast, they make much less impact

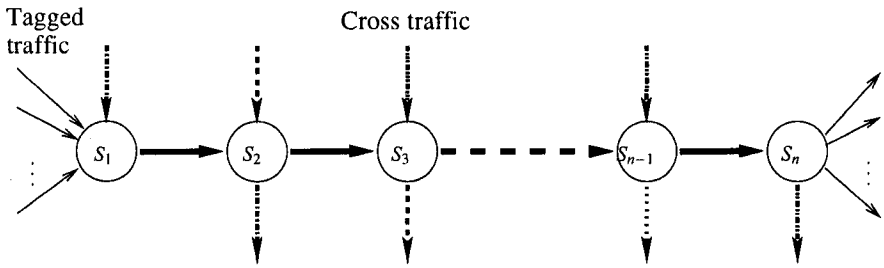



Figure 2.4: Simulation topology

on the delay bounds of aggregate scheduling. The result also shows that aggregation is more advantageous if the number of hops is large, similar to the conclusion in [22].

Next, extensive simulations were conducted to compare the actual delay performance of aggregate and per-flow scheduling. The objective of our simulation is twofold: (i) compare the deterministic delay bounds with the worst-case delay from the simulation and see how tight/loose the deterministic bounds are; and (ii) compare the worst-case delays of aggregate and per-flow scheduling. We intended to find conditions under which aggregate scheduling outperforms per-flow scheduling in terms of e2e delay.

### 2.4.1 Simulation Setup

In the simulation, we used the *ns2* [44] simulator and the topology in Fig. 2.4 which is widely-used elsewhere, *e.g.*, [45, 46]. In this topology, a number of "tagged" flows enter the network at the ingress node $S_1$, and traverse all the other nodes until they reach the egress node $S_n$. The "tagged" flows are those of interest to our study; their e2e delays were checked at the egress node. In order to simulate interference by cross-traffic, external traffic was injected at every node on the path. The cross-traffic at each node shared the path with the tagged traffic for only one hop before exiting the network at the next hop. For backbone links, we set the bandwidth to 160Mb/s and the propagation delay to 2ms, respectively, while for incoming and outgoing links, we set the bandwidth to 10Mb/s and the propagation delay to 10ms. The total number of tagged flows was fixed at 128, which is divided into multiple aggregates in the aggregate scheduling cases.

The tagged flows were generated by using a modified CBR model with varying packet and burst sizes. Each incoming tagged flow was shaped by a token bucket. The cross traffic was generated by using the Pareto On/Off distribution [47, 48], which can simulate long-range dependencies and is known to be suitable for a large volume of traffic.

The *ns2* version of WFQ (Weighted Fair Queueing) was used as the GR scheduler at each backbone node for both per-flow and aggregate scheduling. For aggregate scheduling, two versions of aggregator were used: work-conserving stand-alone aggregator and non-work-conserving rate-controlled (RC) aggregator. To support the non-work-conserving aggregator, a version of rate-controlled fair queueing was implemented.

- *Parameters*

  To run simulations under different scenarios, we varied several parameters, including:

  - packet size of the tagged flows
  - packet size of cross traffic
  - flow rate of the tagged flows
  - the number of hops the tagged flows travel
  - the number of flows in each aggregate
  - the link utilization
  - the burstiness of the tagged flows
  - the burstiness of the cross traffic

- *Metrics*

  The main performance metric is the worst-case e2e delay. For each scenario, 36 independent runs were conducted. All the results were plotted with the 95% confidence interval [49].

### 2.4.2 Simulation Results

#### A Typical Result

Fig. 2.5 shows a typical result of e2e delays under three different scheduling algorithms—per-flow FQ, aggregate FQ (which uses the stand-alone work-conserving aggregator), and FIFO queueing. As can be seen, the e2e delay under aggregate FQ is found to be most stable. Aggregate FQ yields not only the smallest worst-case delay but also very small delay variation. In contrast, per-flow FQ yields larger worst-case delays and delay variations; the delay under FIFO has the largest fluctuation and the worst performance. The above results were obtained by using: burst size

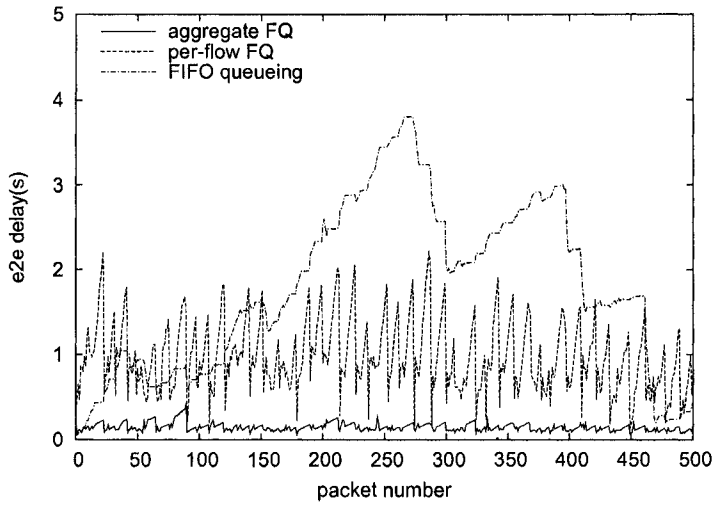



Figure 2.5: End-to-end delay comparison

10000B; packet size 1000B; tagged flow rate 32Kb/s; hop count $K = 15$; the number of flows in the aggregate $N = 16$; and link utilization 55%. With these values, we can compute the deterministic bounds from Eqs. (2.10) and (2.21). The bounds turn out to be 6s for per-flow FQ and 5.53s for aggregate FQ. Both the bounds are much larger than the worst-case delay found from the simulation, implying that they are rather pessimistic.

**Homogeneous Case**

To find the main factors that affect the delay performance of both aggregate and per-flow scheduling, we first used homogeneous flows in each aggregate. We used the $2^k \cdot r$ *factorial design* method [49] to evaluate the contribution of different parameters. Eight parameters ($k = 8$) were used, each with two different values. Each scenario was run 36 times ($r = 36$). The parameters and their values are summarized in Table 2.2.

After collecting data, we used statistical tool ANOVA (ANalysis Of VAriance) [49] to analyze the significance of each parameter. Our focus was on the value $d_r = \frac{d_a}{d_f}$, where $d_a$ ($d_f$) is the worst-case delay under aggregate (per-flow) scheduling. The intention was to find which parameters affect most the relative delay performance between aggregate and per-flow scheduling. The second parameter—packet size of cross traffic—turned out to have little effect on value $d_r$. This is easy to explain, since from the delay bounds in Eqs. (2.10), (2.21) and (2.35), the packet size of cross-

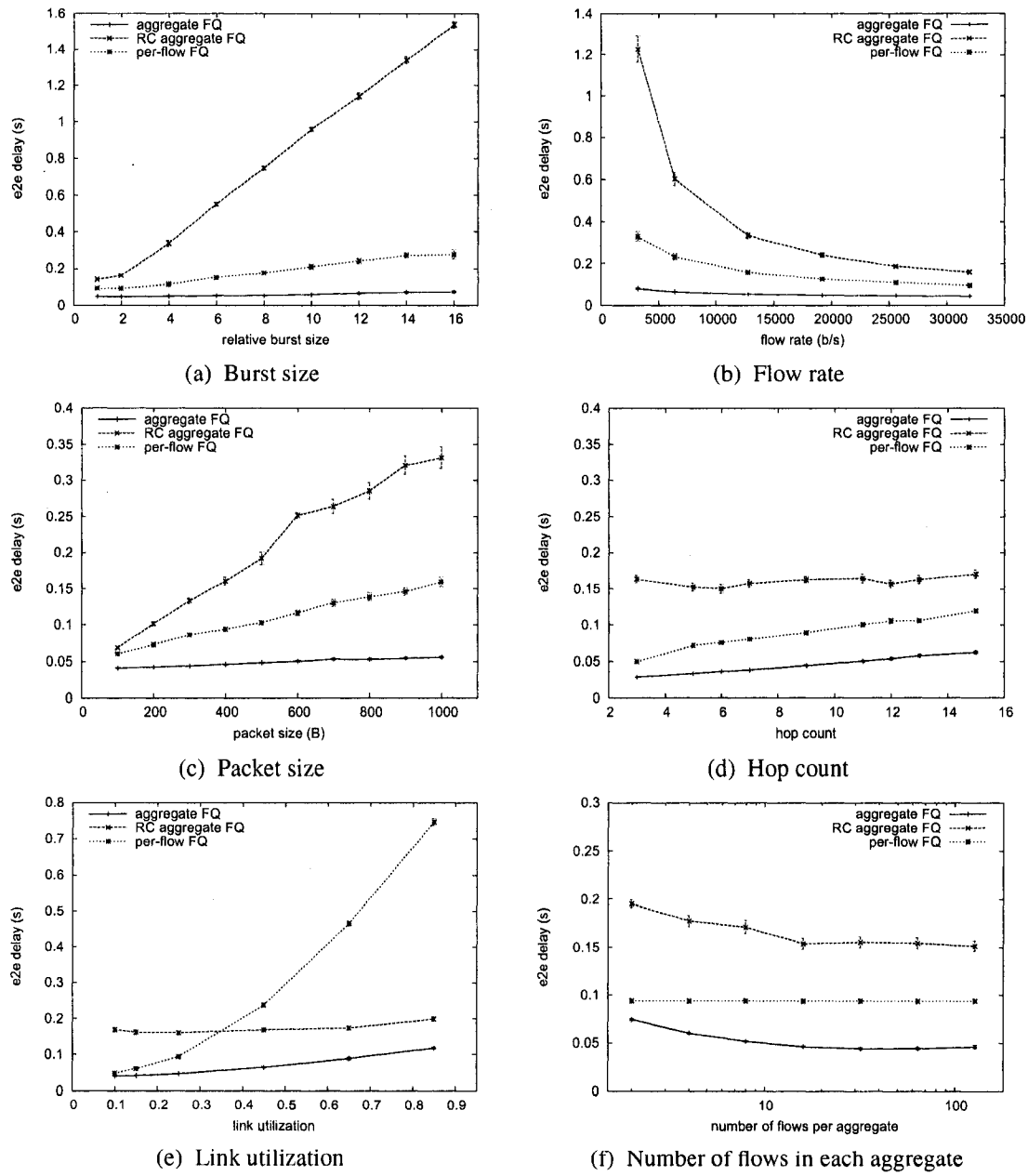
traffic shows up only in the term $\frac{L^i_{max}}{C^i}$, which is usually negligible since $C^i$ is very large. Therefore, we fixed its value at 1500B in all of the following simulations.

To illustrate the effect of the factors on the e2e delay, we ran multiple sets of simulation. For each set of simulation, we varied only one parameter with all the others fixed. All the default values of the parameters are summarized in Table 2.3; the simulation results are summarized in Fig. 2.6. In Figs. 2.6 and 2.7, as well as in the following discussion, "aggregate FQ" stands for aggregate scheduling using stand-alone fair queueing; "RC aggregate FQ" stands for aggregate scheduling using rate-controlled (RC) fair queueing.

Let us compare the performance of per-flow and aggregate FQ first. In Fig. 2.6 (a), as the burst size of the flows increases, the worst-case delay under per-flow FQ increases significantly faster than that under aggregate FQ. Surprisingly, the delays of aggregate FQ do not increase as fast as expected. This behavior can be attributed to the multiplexing gain of aggregate scheduling. In other words, although all flows become burstier, the probability that all flows reach their peaks (of load) at the same time is very low. Instead, the peaks and valleys are more likely to be evened out.

In Fig. 2.6 (b), as the flow rate increases, the worst-case delay under per-flow FQ decreases faster than that under aggregate FQ. Thus aggregate FQ is shown to be more advantageous for lower-rate flows. In Figs. 2.6 (c) and (d), as the packet size (hop count) increases, the delay under per-flow FQ increases faster than that under aggregate FQ. Therefore, aggregate FQ is more advantageous when the packet size (hop count) is large. In Fig. 2.6 (e), as network link utilization increases, the delay under per-flow FQ increases dramatically faster than that under aggregate FQ, showing that when the network utilization is high, aggregate FQ becomes more advantageous. In Fig. 2.6 (f), as the number of flows in an aggregate increases, the e2e delay under aggregate FQ decreases, while that under per-flow FQ remains unchanged. Aggregate FQ is shown to be more advantageous when the number of flows aggregated gets larger. However, as the number increases, the pace of increase becomes smaller since the margin of multiplexing gain decreases.

The effects of flow rate, packet size and hop count can be easily explained by the deterministic bounds. Since GR scheduling algorithms are rate-based, the delay of a flow is coupled with its reserved rate. Although aggregate scheduling has the same problem as per-flow scheduling, the reserved rate for an aggregate ($R_A$) is much larger than that of a single flow ($r_f$), thus the problem is not as severe as in per-flow scheduling. As for packet size, according to Eqs. (2.10) and (2.21), the delay bounds are proportional to the maximum packet size in a flow (aggregate). With aggregation, however, $R_A$ is much larger than $r_f$. Thus the delay bound increases much faster under per-flow




(a) Burst size (b) Flow rate

(c) Packet size (d) Hop count

(e) Link utilization (f) Number of flows in each aggregate

Figure 2.6: Comparison of aggregate and per-flow scheduling

scheduling. The same holds true for hop count. The effect of link utilization is also related to the coupling problem of GR scheduling. When the link utilization becomes higher, there is less spare bandwidth left. Therefore, the rate for a flow (aggregate) is decreased to its reserved rate. Since $R_A$ is much larger than $r_f$, aggregate scheduling has less increase in delay. Finally, since per-flow scheduling is independent of the number of flows in an aggregate ($N$), its delay should not change with $N$. For aggregate scheduling, from Eq. (2.21), two terms are related to $N$, the number of flows: $\frac{\sum_{k=1}^{N} \ell_k^{max}}{R}$ and $(K-3)\frac{\ell_A^{max}}{R}$. Since only identical flows are considered, the first term does not vary with the number of flows. However, $R$ increases proportionally to $N$. Thus the second term decreases slightly, and the total e2e delay decreases.

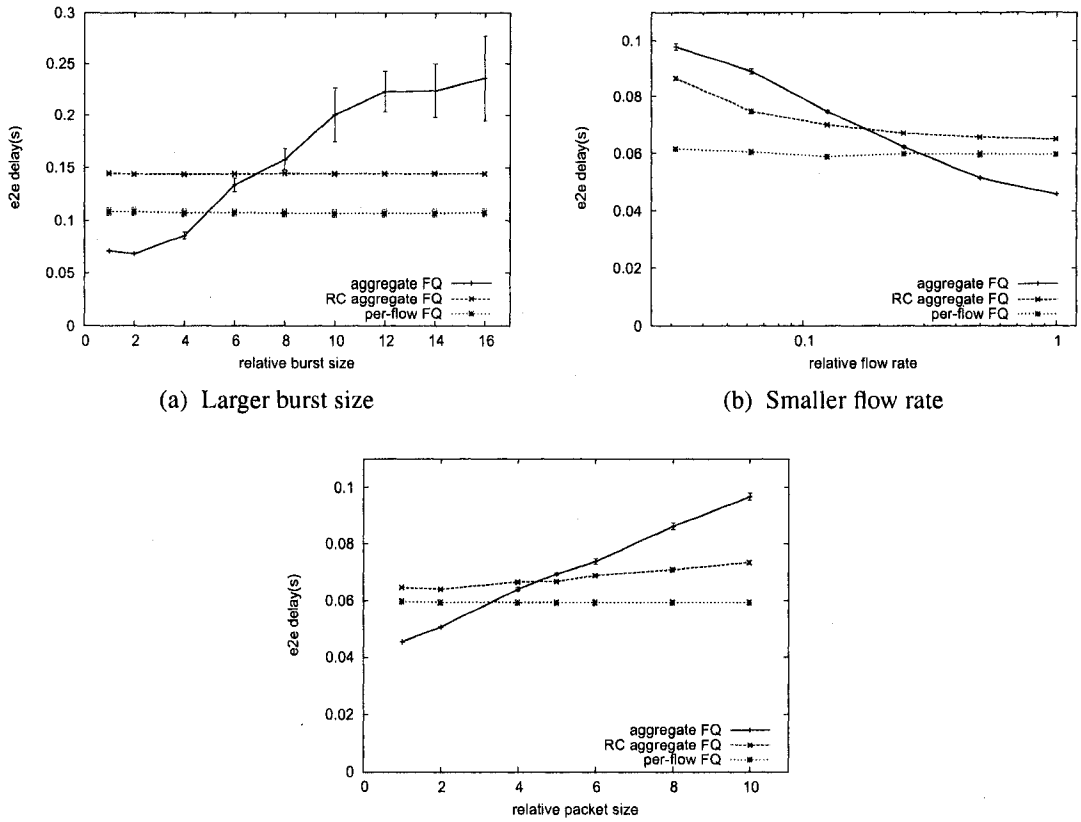
Now, consider the results for RC aggregate FQ. In all the scenarios, the worst-case delays under RC aggregate FQ follow the same trend as those under aggregate FQ. However, since the default link utilization is only 25%, RC aggregate FQ performs worse than aggregate FQ; in most cases, its performance is even worse than that of per-flow FQ. This is mainly because RC aggregate FQ is non-work-conserving and does not take advantage of spare bandwidth. On the other hand, as Fig. 2.6 (e) shows, when the link utilization becomes larger, RC aggregate FQ still performs better than per-flow FQ, and the difference between RC aggregate FQ and aggregate FQ becomes smaller.

We also ran simulations by varying the burstiness of cross-traffic. As the burstiness of cross traffic increases, the worst-case delay for per-flow scheduling increases faster than those under the two aggregate FQ cases, showing that aggregate scheduling is more robust to the burstiness of cross traffic than per-flow scheduling.

**Heterogeneous Case**

After studying the effects of different parameters by using homogeneous flows in a traffic aggregate, we mixed heterogeneous flows (in terms of packet size, flow rate, and burst ratio) into an aggregate to determine the type of flows suitable to be aggregated together. In our simulation, 16 flows were aggregated: 15 of them were identical flows; only one flow was different from the rest. We measured the worst-case delay of this flow while varying the parameters of other flows.

Fig. 2.7 (a) shows the result when the small-burst flow (with relative burst size 1) is aggregated with larger-burst flows. As the burst size of the other flows increases, the worst-case delay under aggregate FQ increases quickly. As expected, mixing a flow with other larger-burst flows will hurt its delay performance. Note that the performance of RC aggregate FQ is very stable since the




(a) Larger burst size

(b) Smaller flow rate

(c) Larger packet size

Figure 2.7: Aggregate heterogeneous flows

aggregator controls the burstiness of the output aggregate.

Fig. 2.7 (b) plots the result when a high-rate flow (with rate 128Kbps) is aggregated with low-rate flows. As the rate of other flows gets smaller, the delay under both aggregate FQ and RC aggregate FQ increases, and the delay under aggregate FQ increases faster and eventually becomes larger than that under per-flow FQ. From Eq. (2.21) one can see that as the rates of other flows decrease, $R$ decreases, and thus, all three terms—$\frac{\Sigma_{k \neq f} \sigma_k}{R}$, $\frac{\Sigma_{k=1}^{N} \ell_k^{max}}{R}$ and $(K-3)\frac{\ell_A^{max}}{R}$—increase, so the total delay increases. Due to the rate-control at the aggregator, the delay increase under RC aggregate FQ is slower.

Fig. 2.7 (c) shows the result when a small-packet flow (with default size 100B) $f_s$ is aggregated with large-packet flows. When the packet size of other flows becomes larger, the delay performance of $f_s$ is shown to suffer—resulting in an even larger delay than that under per-flow FQ and RC aggregate FQ. This is because when aggregated with large-packet flows, a small packet has to wait

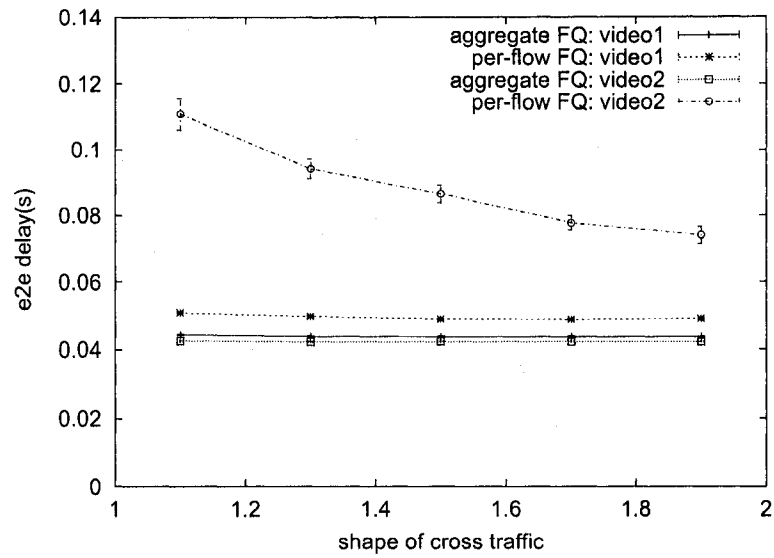



Figure 2.8: Delay results for video traces

behind other large packets in the same aggregate. This implies that flows of similar packet size should be aggregated together.

For the purpose of comparison, we also mixed a flow with other flows with smaller burst size, larger flow rate, and smaller packet size. The results show little or marginal change on the worst-case delay of aggregate scheduling. Similarly, the results can be explained with the delay bound in Eqs. (2.21) and (2.35).

### A Case Study with MPEG Traces

To further compare the delay performance of aggregate FQ and per-flow FQ, we also used real MPEG-4 traces [50] in the simulation. Two video traces were used—high rate "Soccer" and low rate "Silence of the Lambs". The parameters of these two traces are shown in Table 2.4. They have very different burstiness, packet rate, packet rate variation. With the same simulation setup as before, we mixed 8 tagged flows driven by the "Soccer" trace and 8 other tagged flows driven by the "Silence of the Lambs" trace into an aggregate. The results are summarized in Fig. 2.8, which shows that for flows driven by both traces, aggregate FQ provides smaller maximum e2e delays; also, the improvement is much larger for burstier flows driven by the "Silence of the Lambs" trace.

## 2.5 Related Work and Discussions

Aggregate scheduling has been studied extensively in the literature. In [51, 52], the authors proposed some grouping techniques to optimize the implementation of fair queueing. Certain flows (with similar throughput parameters) are grouped together. For example, the scheme in [52] is confined to ATM networks, in which the routers support only a fixed number of rates, and all the flows of the same rate are placed into a single group. It takes advantage of the fact that all the cells have the same size and all the flows in the same group have the same rate, thus simplifying the sorting of flows. However, it still uses per-flow-based scheduling. Although these two algorithms are sometimes called *aggregate scheduling*, they are different from the aggregate scheduling considered here, because the core routers still recognize *each individual* flow. They are just efficient implementations of per-flow-based fair queueing. Other work on implementations of fair queueing include [53, 54, 55].

The authors of [56] studied QoS guarantees under aggregation (e2e aggregation was called *grouping*). Similar to our study, fair queueing is used to handle aggregates at core routers (or in aggregation regions). Based on some given e2e delay requirement, they derived the bandwidth and buffer requirements by using the IETF Guaranteed Service (GS) traffic specification (TSpec), and demonstrated the advantages of aggregating flows of equal or similar delay requirements. By contrast, we derived the e2e delay bound under a given bandwidth guarantee without considering any buffer requirement.

Although the main focus of [23] is QoS routing, it discussed flow aggregation by defining the notion of "*burst ratio*." For flows conforming to the token bucket model, the burst ratio $r$ is the ratio of the burst size to the average rate, or $\sigma/\rho$. The authors suggested aggregation of flows of same or similar burst ratio, since flows with the same burst ratio can be merged and divided without changing the burst ratio of the resulting flows. This conclusion is the same as ours in the work-conserving case in Section IV. The authors of [23] analyzed e2e delay for fluid traffic model, without considering any non-fluid traffic model. In contrast, our delay bound analysis is based on packet traffic model.

Cobb's work [37, 38] is closest to ours. He studied the delay bound problem of aggregate scheduling by using rate-based scheduling algorithms. The core routers treat each aggregate as a single flow, and handle all the aggregates by using rate-based fair queueing. He defined a class of fair aggregator and showed that, by using such fair aggregator, the e2e delay is bounded. Such

aggregator can be used to aggregate the traffic recursively, and the e2e delay bound still exists. He also proposed two types of fair aggregators, called the *basic fair aggregator* and *greedy fair aggregator*. By using such aggregators, he showed that the e2e delay bound can be even smaller than the per-flow e2e delay bound.

We have several observations on the results in [37, 38]. First, the main idea of its *fair aggregator* is to control the burst of an aggregate. Such an aggregator works in a non-work-conserving way. Second, the implicit assumption of the *basic fair aggregator* and *greedy fair aggregator* is that only one aggregate goes out of an aggregator. In other words, all the flows going through the same outgoing link belong to the same aggregate. In contrast, our work is more general. In Section 2.2 we showed that the delay bounds exist even for work-conserving aggregators. In Section 2.3, we extended the result in [38] by allowing multiple aggregates going out of the same link of an aggregator, and derived the e2e delay bound under the token bucket traffic model.

## 2.6 Concluding Remarks

In this chapter, we first derived deterministic delay bounds for aggregates under the assumption that the incoming traffic at each aggregator conforms to the token bucket model and guaranteed-rate (GR) scheduling algorithms are used in each aggregation region. We considered three types of GR scheduling algorithms at an aggregator: stand-alone, two-level hierarchical, and rate-controlled two-level hierarchical GR algorithms. The delay bounds are shown to depend on several factors, such as the scheduling constant at each hop and the latency at the aggregator. We should, therefore, use the scheduling algorithms with a small scheduling constant at each hop, and those with small latency at aggregators. Among all the rate-based scheduling algorithms, PGPS, VC, and WF$^2$Q have the smallest scheduling constant ($\frac{L^i_{max}}{C^i}$) and latency ($\frac{\ell^{max}_f}{r_f} + \frac{L^i_{max}}{C^i}$ for flow $f$). The delay bounds also indicate that it is beneficial to aggregate flows with similar burst ratios. Aggregate scheduling provides better e2e delay bounds when a large number of hops use aggregate scheduling, because the overhead at the aggregators will be offset by the larger guaranteed rate for an aggregate. If the number of hops is small, the aggregation overhead becomes "relatively" significant.

We also showed by simulation that aggregate scheduling is robust. By exploiting multiplexing gains, it can provide better worst-case delay performances than per-flow scheduling, as long as those flows aggregated together are not very diverse in terms of packet size, flow rate, and burst

ratio. In addition, the simulation results also showed that in most scenarios non-work-conserving aggregator performs worse than work-conserving aggregator, since it does not take advantage of the spare bandwidth in the network. Also, the deterministic bounds are shown to be rather pessimistic. Although the simulation may not capture the worst-case e2e delay, it implies that the probability for the worst-case e2e delay to happen be very small.

Note that resource reservation and admission control were not covered in this chapter, but techniques in the literature (*e.g.*, [16]) can be used for this purpose. We also assumed that the e2e path in an aggregation region can be set up by traffic engineering mechanisms similar to the way the Label Switched Path (LSP) is set up in an MPLS network.

Table 2.1: Symbols

| | |
|---|---|
| $S_i$ | the $i^{th}$ router/server along the path of a flow or aggregate |
| $p_f^j$ | the $j^{th}$ packet of flow $f$ |
| $p_A^j$ | the $j^{th}$ packet of aggregate flow $A$ |
| $\ell_f^j$ | packet length of $p_f^j$ |
| $\ell_A^j$ | packet length of $p_A^j$ |
| $\ell_f^{max}$ | max. packet length in flow $f$ |
| $\ell_A^{max}$ | max. packet length in aggregate flow $A$ |
| $L_{max}^i$ | max. packet length at router $S_i$ |
| $C^i$ | link capacity between $S_i$ and $S_{i+1}$ |
| $\sigma_f$ | burst size of flow $f$ under token bucket model |
| $\rho_f$ | average rate of flow $f$ under token bucket model |
| $r_f$ | guaranteed rate for flow $f$ ($r_f \geq \rho_f$) |
| $R$ | sum of the guaranteed rates of all the flows in the aggregate flow, *i.e.*, $R = \sum_{k=1}^{N} r_k$ |
| $A_f(\tau,t)$ | traffic arrived from flow $f$ during $(\tau,t]$ |
| $A^i(p_f^j)$ | arrival time of packet $p_f^j$ at router $S_i$ |
| $GRC^i(p_f^j)$ | guaranteed rate clock for $p_f^j$ at $S_i$ |
| $D^i(p_f^j)$ | departure time of $p_f^j$ at router $S_i$ |
| $\alpha^i$ | $\alpha^i = \beta^i + \tau^{i,i+1}$ |
| $\beta^i$ | scheduling constant at router $S_i$ |
| $\tau^{i,i+1}$ | propagation delay between $S_i$ and $S_{i+1}$ |
| $\gamma^i$ | aggregation constant at router $S_i$ |
| $d_f^j$ | e2e delay bound for $p_f^j$ |

Table 2.2: Parameters and their values in ANOVA test

| **Parameters** | **Values** |
|---|---|
| Tagged flow packet size | 100B, 1000B |
| Cross traffic packet size | 100B, 1000B |
| Tagged flow rate | 3.2Kbps, 32Kbps |
| Hop count | 5, 20 |
| Burst size[a] of tagged flows | 2, 10 |
| Number of tagged flows in one aggregate | 4, 128 |
| Link utilization | 5%, 50% |
| Shape[b] of cross traffic | 1.1, 1.9 |

[a] *The burst size is a relative value: value k means the burst size is k times of the default packet size.*

[b] *The "shape" parameter is used by the Pareto traffic model, which affects the burstiness of the traffic. We use it to vary the burstiness of the cross traffic.*

Table 2.3: Default parameter values

| **Parameters** | **Values** |
|---|---|
| Tagged flow packet size | 400B |
| Tagged flow rate | 32Kbps |
| Hop count | 10 |
| Burst size of tagged flows | 2 |
| Number of tagged flows in each aggregate | 16 |
| Link utilization | 25% |
| Shape of cross traffic | 1.5 |
| Cross traffic packet size | 1500B |
| Total number of tagged flows | 128 |

Table 2.4: Parameters of the video traces

| **Video Name** | **Frame Size (KB)** | | **Bit Rate (Kbps)** | | **Burst Size (Kb)** |
|---|---|---|---|---|---|
| | **Mean** | **Peak** | **Mean** | **Peak** | |
| **Soccer** | 5.53 | 17.93 | 1,106.6 | 3,585.4 | 140 |
| **Silence of the Lambs** | 0.53 | 11.29 | 105.6 | 2,258.4 | 70 |

# CHAPTER 3

# Coordinated Aggregate Scheduling for Improving End-to-End Delay Performance

In Chapter 2, using aggregate GR scheduling algorithms, we derived deterministic e2e delay bounds under the assumption that all incoming flows at an aggregator conform to the token bucket model. Each aggregator uses a GR scheduling algorithm which is either work-conserving or non-work-conserving. We showed not only the existence of e2e delay bounds for each flow, but also the fact that under certain conditions (*e.g.*, when the aggregate traverses a long path after the aggregation point) the bounds are tighter than that of per-flow scheduling. The simulation results have shown that aggregate scheduling is very robust and can exploit statistical multiplexing gains, and that it performs better than per-flow scheduling in most cases.

However, when work-conserving scheduling algorithms are used at aggregators, the delay bound of a flow is dictated by the burstiness of other flows in the same aggregate. Thus, if a flow is aggregated with other bursty flows, it will suffer a long e2e delay. The main culprit of this long delay lies in the fact that packets within each aggregate are handled by the routers using FIFO scheduling.

To remedy this problem, we propose a new aggregate scheduling algorithm, which improves the e2e delay by reordering the packets in an aggregate using EDF scheduling based on their deadlines. The deadline of a packet at an intermediate node is related to its guaranteed rate clock (GRC) value at the aggregator. This new algorithm makes the delay of a flow independent of the burstiness of other flows in the same aggregate, yielding smaller e2e delays. Since the algorithm uses EDF inside an aggregate and computation of the deadline of a packet at each intermediate node is coordinated between the node itself and its upstream nodes, we call it *coordinated aggregate scheduling* (CAS). For the purpose of differentiation, we call the aggregate scheduling algorithms discussed

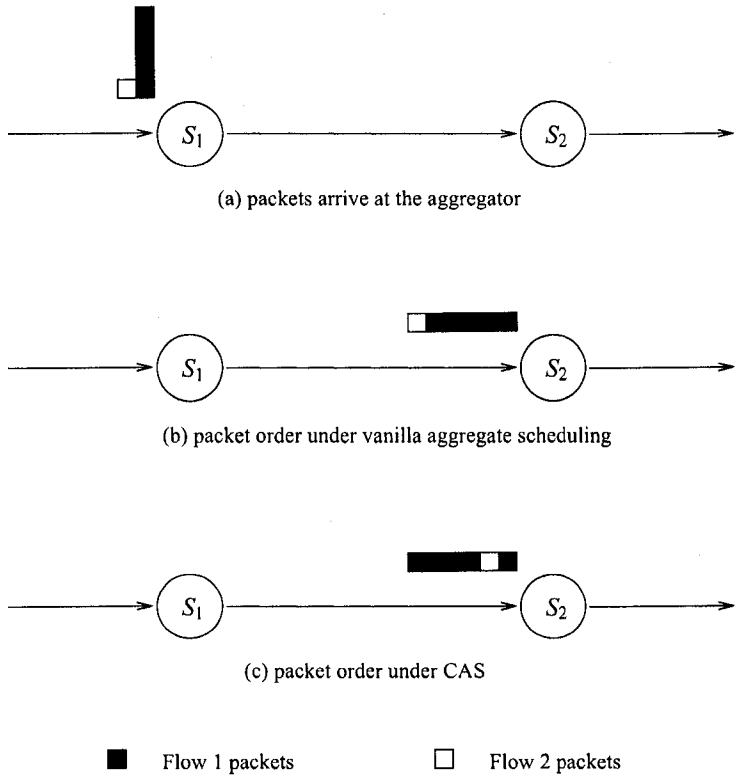



Figure 3.1: Coordinated aggregate scheduling

in Chapter 2 that use FIFO queueing inside each aggregate *"vanilla" aggregate scheduling* (VAS).

The rest of the chapter is organized as follows. Section 3.1 introduces the CAS algorithm, and proves that under the token bucket traffic model CAS provides tighter e2e delay bound than VAS. Section 3.2 discusses the implementation issues of CAS, and proposes a simple multi-queue structure and an adaptive queue management algorithm. Section 3.3 presents evaluation results. Simulation is used to compare the delay performance of both coordinated and vanilla aggregate scheduling, confirming the benefits of CAS derived from the analysis. Section 3.4 discusses related work on CAS, putting our results in a comparative perspective. Finally, Section 3.6 summarizes our contributions.

## 3.1 Coordinated Aggregate Scheduling with EDF inside an Aggregate

As discussed above, the delay bound under aggregate scheduling depends on the burst sizes of all the other flows in the same aggregate, mainly because at the downstream nodes of the aggregator, the packets in the same aggregate are handled with FIFO queueing, irrespective of which flows they belong to. Thus, if a large burst of packets of a flow arrive at an idle aggregator, the burst can traverse the aggregator very quickly. If packets from other flows in the same aggregate arrive immediately after the burst, they must wait behind the burst in the FIFO queue at the remaining nodes of the aggregate's path. Figs. 3.1 (a) and (b) illustrate this scenario. In Fig. 3.1 (a), a burst of flow 1 arrives at an aggregator $S_1$ immediately before the arrival of a packet from flow 2. Suppose $S_1$ is idle at that time, then the burst goes through the aggregator very quickly, as does the packet from flow 2. Since both flows share the same aggregate, the burst of flow 1 will be ahead of the flow-2 packet in the aggregate until the aggregate is split later on the path. Suppose the next node $S_2$ is the bottleneck, then the burst of flow 1 will be scheduled at a slower speed, and the flow-2 packet has to wait behind the burst of flow 1 in the FIFO queue, suffering a long delay. To solve this problem, one may use a rate-controlled scheduler at the aggregator $S_1$, thus controlling the output rate for the aggregate, and there will be no large traffic burst waiting at the downstream nodes. However, rate-controlled schedulers work in a non-work-conserving fashion, causing a longer average delay.

We propose another method to solve the delay problem by changing the order of transmitting packets in the queue of an aggregate, such that higher-priority packets can be scheduled earlier. In essence, we change the FIFO queue for an aggregate used by vanilla aggregate scheduling algorithms into an EDF queue. The rationale behind this is that, if the aggregator is relatively lightly-loaded, a large burst of packets can get through it earlier than their expected finish times according to their GRC values. If a downstream node becomes bottleneck, this burst will stay ahead of some packets with smaller GRC values in the queue of that node. By reordering the packets in the queue based on their GRC values at the aggregator, if multiple packets of an aggregate are waiting in the queue of a downstream node, a packet with the minimum GRC value will be scheduled first. In other words, the GRC value at the aggregator plays the role of a packet's deadline. Note that this algorithm only reorders the packets in the queue of the same aggregate; the scheduling of different aggregates remains the same—the GR scheduling algorithms. This new algorithm is called the *coordinated aggregate scheduling* (CAS). As shown in Fig. 3.1 (c), after

reordering the packets in the queue at $S_2$, the packet from flow 2 will be scheduled earlier.

Next, we give the details of the CAS algorithm and prove that this algorithm improves the e2e delay bound of a flow. Note that when using GR scheduling algorithms, we always assume that the guaranteed rate for a flow is greater than, or equal to, its average rate, *i.e.*, $r_f \geq \rho_f$.

### 3.1.1 CAS Algorithm

For the downstream nodes to be able to reorder packets, the packets have to carry some information about their GRC values at the aggregator. The key idea of the CAS algorithm is to insert a *lag* field in each packet, which contains information on how much the packet was behind its "deadline" at the previous hop. Then, at the next hop the server can adjust the packet's arrival time by subtracting its lag value. The order of transmitting packets in the waiting queue of an aggregate will be adjusted according to the new "virtual" arrival times.

Now, let us consider the scheduling algorithm at both the aggregator and the downstream nodes. For simplicity, the propagation delay between neighboring nodes is omitted in the following discussion. At the aggregator (server $S_i$), per-flow scheduling is used. Thus, for flow $f$, $GRC^i(p_f^j)$ is defined based on its reserved rate $r_f$ according to Eq. (2.1). Let $\delta^i(p_f^j)$ and $D^i(p_f^j)$ be the lag value and the departure time of packet $p_f^j$ at $S_i$, respectively. Then,

$$\delta^i(p_f^j) = D^i(p_f^j) - GRC^i(p_f^j). \tag{3.1}$$

At the downstream nodes $S_{i+k}$ ( $k \geq 1$), the lag value $\delta^{i+k}(p_f^j)$ is defined differently:

$$\delta^{i+k}(p_f^j) = D^{i+k}(p_f^j) - VA^{i+k}(p_f^j), \quad k \geq 1. \tag{3.2}$$

The virtual arrival (VA) time $VA^{i+k}(p_f^j)$ is calculated recursively as:

$$\begin{aligned} VA^{i+k}(p_f^j) &= A^{i+k}(p_f^j) - \delta^{i+k-1}(p_f^j) \\ &= A^{i+k}(p_f^j) - (D^{i+k-1}(p_f^j) - VA^{i+k-1}(p_f^j)) \\ &= VA^{i+k-1}(p_f^j) \\ &\vdots \\ &= GRC^i(p_f^j), \quad k \geq 1. \end{aligned} \tag{3.3}$$

We assume $A^{i+k}(p_f^j) = D^{i+k-1}(p_f^j)$ since the propagation delay is omitted. From Eq. (3.3), the virtual arrival time of a packet at downstream nodes is exactly the same as the packet's GRC value at the aggregator. Thus, at every node after the aggregator, the packets are ordered in their GRC values at the aggregator. (A simple implementation is to just store the GRC value in the packet at the aggregator. However, to remove the need for clock synchronization, we store the lag value and adjust the virtual arrival time at each downstream hop, as is done in the above algorithm.) Moreover, the algorithm also preserves the order of packets in the same flow, since

$$\begin{aligned} VA^{i+k}(p_f^j) = GRC^i(p_f^j) &\geq GRC^i(p_f^{j-1}) + \frac{\ell_f^j}{r_f} \\ &= VA^{i+k}(p_f^{j-1}) + \frac{\ell_f^j}{r_f} \\ &> VA^{i+k}(p_f^{j-1}). \end{aligned} \tag{3.4}$$

Also, the "deadline" of a packet is defined differently at different nodes: at the aggregator, the packet's deadline is its GRC value, while at the downstream nodes, the deadline is its virtual arrival time, not its GRC value (or expected finish time) at those nodes. The reason for this is that we want to re-adjust the order of packets in the same aggregate to keep them in the order of GRC values at the aggregator. Using the virtual arrival time achieves this goal.

Intuitively, because of the packet reordering, the delay of the packets in a flow has the potential to be independent of the burst sizes of other flows within the same aggregate, which leads to smaller e2e delay.

### 3.1.2 End-to-End Delay Bound

An important step in deriving the e2e delay bound under CAS is to derive the delay at the aggregator. The following lemma is a stepping stone, which considers a special case: the busy periods of all flows in the same aggregate coincide with one another at the aggregator.

**Lemma 3.1.1.** *Let $S_i$ and $S_{i+1}$ be two neighboring GR servers using the CAS algorithm and $S_i$ be an aggregator. $S_i$ aggregates flow $f$ and other (N-1) flows into an aggregate flow A, which is an input to $S_{i+1}$. Suppose the guaranteed rate for flow k at $S_i$ is $r_k$ ($1 \leq k \leq N$), and let $p_f^j = p_A^{j'}$,* i.e., *the $j^{th}$ packet of flow $f$ corresponds to the $j'^{th}$ packet of aggregate flow A. If the busy periods of*

*all flows in the same aggregate coincide with one another at the aggregator, then for packet* $p_f^j$,

$$GRC^{i+1}(p_A^{j'}) \leq GRC^i(p_f^j) + \frac{\sum_{k=1}^{N} \ell_k^{max}}{R} + \alpha^i, j \geq 1, \tag{3.5}$$

*where* $R = \sum_{k=1}^{N} r_k$ *is the guaranteed rate for aggregate flow A at* $S_{i+1}$.

See Appendix B.1 for a detailed proof of this lemma.

Using Lemma 3.1.1, we can now derive the e2e delay bound for CAS.

**Theorem 3.1.1.** *Suppose N flows (i) share the same K hops of GR servers (using CAS) inside an aggregation region, and (ii) are bundled into aggregate flow A at aggregator* $S_1$ *and split back into the original individual flows at* $S_K$. *Routers* $S_2, \ldots, S_{K-1}$ *schedule packets of aggregate A. If flow k has the guaranteed rate* $r_k$ *with* $r_k \geq \rho_k$ $(1 \leq k \leq N)$ *at* $S_1$ *and* $S_K$, *and A has the guaranteed rate* $R = \sum_{k=1}^{N} r_k$ *at* $S_2, \ldots, S_{K-1}$, *then for any flow f* $(1 \leq f \leq N)$, *the e2e delay of packet* $p_f^j$, $d_f^j$, *is bounded as:*

$$\begin{aligned} d_f^j &= D^K(p_f^j) - A^1(p_f^j) \\ &\leq [GRC^1(p_f^j) - A^1(p_f^j)] + \frac{\sum_{k=1}^{N} \ell_k^{max}}{R} + (K-3)\frac{\ell_A^{max}}{R} + \frac{\ell_f^{max}}{r_f} + \sum_{i=1}^{K} \alpha^i. \end{aligned} \tag{3.6}$$

*Then, if flow f conforms to the token bucket model* $(\sigma_f, \rho_f)$,

$$d_f^j \leq \frac{\sigma_f}{r_f} + \frac{\sum_{k=1}^{N} \ell_k^{max}}{R} + (K-3)\frac{\ell_A^{max}}{R} + \frac{\ell_f^{max}}{r_f} + \sum_{i=1}^{K} \alpha^i, \quad j \geq 1. \tag{3.7}$$

See Appendix B.2 for the details of the proof, which consists of the following two parts.

**Part 1:** The theorem holds for an extreme case when the busy periods of all flows in the same aggregate coincide with one another at the aggregator. In this case, no packet reordering is needed since packets leave the aggregator in the order of their GRC values.

**Part 2:** The above extreme case is the worst case, and thus, the theorem holds for all other cases.

Comparing Eqs. (2.21) and (3.7), we can see that the delay bound under CAS is independent of the burst sizes of other flows in the same aggregate, yielding a tighter bound than that in Eq. (2.21). On the other hand, the bound in Eq. (3.7) is related to the summation of the maximum packet sizes

of all the flows in the aggregate. Therefore, if a flow of smaller-size packets are bundled with other flows of larger-size packets into the same aggregate, it will suffer a long delay. However, in practice, the maximum packet size is upper-bounded by the underlying network protocol. For example, the maximum Ethernet packet size is 1518 bytes, and in the case of ATM network, it is only 53 bytes.

Note that when $N = 1$ (*i.e.*, there is only one flow in the aggregate), Eq. (3.7) is simplified to Eq. (2.10), the delay bound under per-flow scheduling. Also, comparing Eqs. (2.10) and (3.7), we note that, depending on the maximum packet sizes and reserved rates of the constituent flows, the delay bound under CAS can be tighter than that under per-flow scheduling.

### 3.1.3 Multiple Aggregations

Note that the derivation of the delay bound in Eq. (3.6) does not require the knowledge of the incoming traffic pattern (such as token bucket). Therefore, we can derive the delay bound even when flows are aggregated multiple times during their journey to their respective destinations. To support multiple aggregations, the lag information has to be carried into aggregators. Therefore, at aggregators in the middle of the network, the lag value of a packet is defined as in Eq. (3.2).

We prove that (i) the scheme still preserves the order of packets in the same flow, and (ii) the e2e delay bound result still holds for the case of multiple aggregations. To derive the e2e delay bound for the case of multiple aggregations, we first prove that the extreme case considered in the proof of Lemma 3.1.1 is still the worst case for e2e delay irrespective of further aggregations at downstream nodes.

**Lemma 3.1.2.** *Suppose $A$ is an aggregate that is bundled with other aggregates at node $S_i$, and $f$ is a constituent flow of $A$. Then, the extreme case is still the worst case for the e2e delay of flow $f$'s packets, irrespective of further aggregations or not.*

*Proof.* Since further aggregations do not change the relative order of packets in each aggregate and at downstream nodes, CAS still chooses first the packet with the smallest VA value, the relative order of scheduling packets in each aggregate does not change. Thus, the worst case does not change. □

With this lemma, we only need to derive the e2e delay bound using the extreme case. The proof of the following theorem is similar to that of Theorem 8 in Chapter 2, thus it is omitted.

**Theorem 3.1.2.** *Suppose (i) flow $f$ traverses $K$ hops of GR servers that use the CAS algorithm; (ii) any of these $K$ hops can be an aggregator; and (iii) for each aggregator there is a corresponding deaggregator at a downstream node. Then, the e2e delay, $d_f^j$, of packet $p_f^j$ satisfies:*

$$d_f^j \leq [GRC^1(p_f^j) - A^1(p_f^j)] + \sum_{i=2}^{K} \frac{\ell_{\hat{A}_i}^{max}}{\hat{R}_i} + \sum_{i=1}^{M} [\frac{\sum_{k \in \mathbb{A}_i} \ell_k^{max} - \ell_{A_i}^{max}}{R_i}] + \sum_{i=1}^{K} \alpha^i, \tag{3.8}$$

*where $M$ is the number of aggregators along the path; $\hat{A}_i$ is the aggregate flow that contains flow $f$ at $S_i$, $\ell_{\hat{A}_i}^{max}$ is the maximum packet size in aggregate flow $\hat{A}_i$, and $\hat{R}_i$ is the guaranteed rate for $\hat{A}_i$ at $S_i$; $A_i$ is the $i^{th}$ aggregate along the path that contains flow $f$, and $\ell_{A_i}^{max}$ and $R_i$ are the maximum packet size in the aggregate and its guaranteed rate, respectively; $\mathbb{A}_i$ represents the set of constituent flows contained in aggregate $A_i$.*

Note that there are a total of ($K$-1)+$M$ terms related to packet size in the above delay bound. They can be understood as follows: the ($K$-1) terms in $\sum_{i=2}^{K} \frac{\ell_{\hat{A}_i}^{max}}{\hat{R}_i}$ correspond to the delays at all of the hops (except the first hop). In addition, for each aggregation with guaranteed rate $R_i$, there is an overhead term $\frac{\sum_{k \in \mathbb{A}_i} \ell_k^{max} - \ell_{A_i}^{max}}{R_i}$. Compared to the delay bound for per-flow scheduling in Eq. (2.10), Eq. (2.36) has $M$ more terms due to aggregation. However, since the guaranteed rates for aggregate flows are much higher at the routers in the aggregation region, the total delay bound in Eq. (2.36) can be tighter than that in Eq. (2.10). Further, if there is only one aggregate ($M$=1), and $S_1$ and $S_K$ are the aggregator and the deaggregator, respectively, Eq. (2.36) can be simplified to Eq. (3.6).

## 3.2 Implementation

So far, we have shown that CAS has very good e2e delay performance. However, since CAS converts fair queueing problem to an EDF problem at core nodes, its implementation overhead can be a concern. Therefore, we need to find an efficient algorithm for packet sorting.

The calendar queue [57] has been widely used for packet sorting. It is shown to have $O(1)$ complexity. However, depending on the number of packets in the queue, the calendar queue has to do a lot of resizing and copying to maintain the "optimal" calendar structure, *i.e.*, not many packets in each bucket nor many empty buckets. Resizing and copying incur significant overhead. Moreover, the calendar queue does not work well over "skewed" priority distribution.

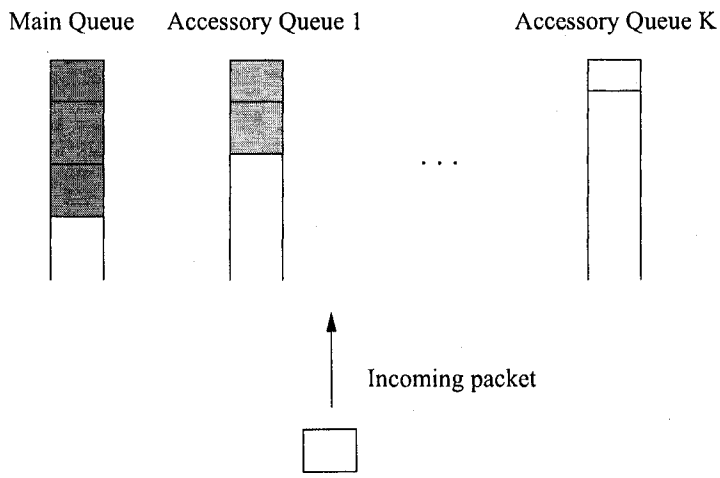



Figure 3.2: The multi-queue structure

Since in the CAS algorithm, fair queueing is used at each aggregator, the virtual arrival (VA) time values of packets in each aggregate are likely to be monotonically increasing. Thus, one FIFO queue is almost enough to handle them. Only a few more queues are necessary to handle those packets that have smaller VA values (higher priority) than their predecessors.

Based on the observations above, we propose a simple multi-queue structure that consists of a main queue and multiple accessory queues. The number of accessory queues varies with the VA values and actual arrival times of packets. A newly-arrived packet is put at the end of the main queue as long as its VA value is larger than that of the last packet in the queue; otherwise, the VA value of the packet is compared to that of the last packet in the first accessory queue. Similarly, if its VA value is larger, the packet will be put at the end of the queue; otherwise, its VA value is compared with that of the last packet at the next accessory queue. The process continues until either the packet is put at the end of an existing queue, or (when its VA value is smaller than that of the last packet of the last accessory queue) a new accessory queue will be created where the packet will be placed. It is easy to see that the VA values of the last packets in the queues are monotonically decreasing—the one in the main queue is larger than that in the first accessory queue; the one in the first accessory queue is larger than that in the second accessory queue, etc.

When a packet from the aggregate needs to be transmitted, the packet with the minimum VA value is chosen. It can be the first packet of any of the existing queues. When an accessory queue becomes empty, it will be deleted; on the other hand, the main queue will always remain there, even if it gets empty.

Compared to the calendar queue, our simple multi-queue scheme has the following advantages: (i) it is simpler, avoiding the "copy" and "resize" operations of the calendar queue when the calendar structure is adjusted; (ii) the clustering problem (due to "skewed" distribution of packet priority) of the calendar queue is avoided; and (iii) the number of queues is independent of the total number of backlogged packets of an aggregate.

In addition, there are several interesting properties associated with the queue operations. First, since packets are scheduled according to their VA values, the accessory queues will be deleted in the reverse order of their creation. In other words, the most recently created queue will be deleted first, since its last packet has higher priority (a smaller VA value) than all the last packets in other queues. In this sense the accessory queues works as a LIFO stack. Second, the packets of a given flow will keep their order. In other words, packets in the same flow will never be reordered. Therefore, the number of queues cannot be greater than the total number of flows in the aggregate. Clearly, a new packet is put into a new queue only when it belongs to a different flow from the last packets in all the existing queues; otherwise, it will have a larger VA value than at least one of them. Third, the packets in the same queue are in increasing order of VA values.

To have small overhead, the number of accessory queues should be small. In fact, the number of queues required is related to several factors, such as the burstiness of flows, network utilization, etc. We expect the average number of queues to be small, since VA values of incoming packets tend to be monotonically increasing.

To further reduce the number of accessory queues, the following optimizations can be incorporated into the basic multi-queue algorithm.

*Optimization 1*: if a queue is short (*e.g.*, of length one/two or shorter than the number of accessory queues), the new packet will be inserted into it.

*Optimization 2*: a small discrepancy is allowed, so that if a packet's VA value is only a little bit ($\delta$) smaller than that of the last packet in the queue, it will be put behind that packet in the queue without moving to the next accessory queue.

However, this optimization can be recursively done so that the VA values of the packets in a queue may be in a reverse order. To solve this problem, the algorithm is revised by recording the maximum value of VA in each queue. A newly-arrived packet compares its VA with this maximum value, not the VA value of the last packet.

Then, the question is what value to use for $\delta$. If it is large, the number of queues will be small, but the delay performance will suffer; if it is too small, then the number of queues will increase.

We design a simple adaptive algorithm, which uses the number of accessory queues as a parameter in determining the $\delta$ value. Suppose the current number of accessory queues is $N_q$, then we set $\delta = N_q \cdot \delta_c$, where $\delta_c$ is a small constant. Thus, if the number of accessory queues is large, we use a larger $\delta$ to make the number of queues smaller; when it is small, we use a smaller $\delta$ to make the ordering in each queue more accurate, improving the delay performance. The pseudocode of the adaptive algorithm is presented in Table 3.1.

The $\delta_c$ value is directly related to the delay. When it is 0.01, for instance, the actual delay discrepancy would be in the range of 0.01 – 0.1sec per hop, if the number of queues is less than 10. In our simulation, $\delta_c$ is set to 0.01, 0.02, or 0.05.

With Optimization 2, the packets in a queue are no longer in increasing order of their VA values. This fact leads to the following optimization to reduce the overhead in choosing the next packet to transmit.

*Optimization 3*: Suppose a packet from queue $i$ is chosen when a packet is to be transmitted from an aggregate, then the next time a packet is to be transmitted from the aggregate, another packet from queue $i$ will be chosen (without searching) as long as the VA value of the first packet in queue $i$ is smaller than that of the previous packet. This is because the first packets in other queues must have larger VA values. This can reduce the search effort during packet transmission from an aggregate. This optimization has to be used together with Optimization 2.

## 3.3 Evaluation

To demonstrate the advantages of the CAS algorithm, we conducted extensive simulations using the *ns2* [44] simulator, especially comparing the e2e delays of CAS and VAS.

### 3.3.1 The Simulation Setup

In the simulation, we used the topology shown in Fig. 2.4 where a number of "tagged" flows enter the network through the ingress node $S_1$, and traverse all the other nodes until they reach the egress node $S_n$. The "tagged" flows are the ones of interest to our study, and their e2e delays are checked by the egress node. In order to simulate interferences by cross-traffic, external traffic is injected at every node on the path. The cross-traffic at each node shares the path with the tagged traffic for only one hop before exiting the network at the next hop. For the backbone links, we set

the bandwidth to 160Mbps and the propagation delay to 2ms, respectively, while for the incoming and outgoing links, we set the bandwidth to 10Mbps and the propagation delay to 10ms.

The tagged flows are generated by using a modified CBR model with varying packet and burst sizes. Each tagged incoming flow is shaped by a token bucket. The cross traffic is generated by using the Pareto On/Off distribution [47, 48], which can simulate long-range dependencies and is known to be suitable for a large volume of traffic.

To verify the performance of CAS, we used two fair queueing algorithms—WFQ (Weighted Fair Queueing) and WF$^2$Q (Worst-case Fair Weighted Fair Queueing) [40]. The *ns2* versions of WFQ and WF$^2$Q were used as the GR scheduler at each backbone node. Both algorithms were modified to support VAS and CAS. For each simulation scenario, we ran simulation to obtain two independent results using WFQ and WF$^2$Q. Each simulation run lasted 50 seconds.

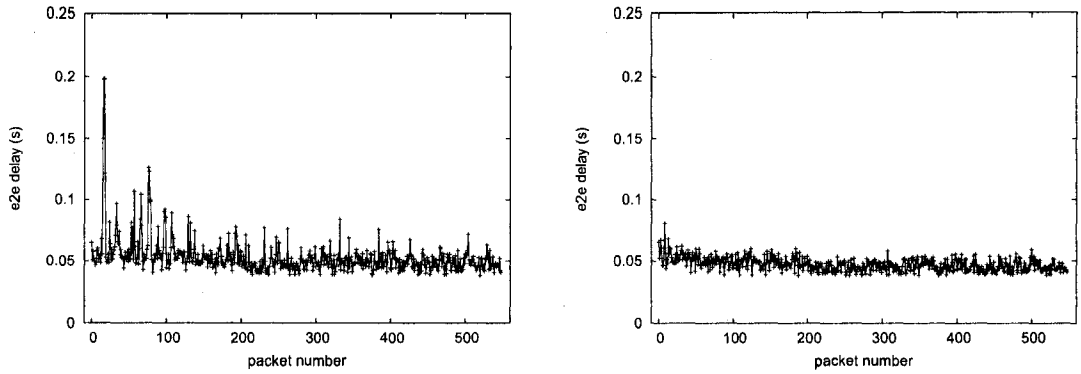
In the simulation, the tagged flows were divided into multiple groups. In each group, one flow (called *red* flow) has a fixed small burst size [1] 2, while all the other flows (called *blue* flows) have variable burst sizes (with the default value 8). Each group of flows were bundled into an aggregate flow at the aggregator. We focused on the e2e delay of the red flows to see the delay performance of flows under different scheduling schemes and network conditions. All the parameters used in the simulation and their default values are summarized in Table 3.2.

### 3.3.2 Simulation Results

First, we compared the e2e delay of the red flows under VAS and CAS. Fig. 3.3 shows the result of one red flow when WFQ is used as the GR scheduling algorithm. As can be seen from the figure, CAS yields not only a smaller worst-case delay but also a very small delay variation. This confirms the analysis results in Section 3.1. The above results were obtained by using the default values of parameters in Table 3.2. Using the same parameters, we repeated the simulation using the WF$^2$Q algorithm, the results of which are plotted in Fig. 3.4. As can be seen from this figure, the performance of CAS is consistently superior.

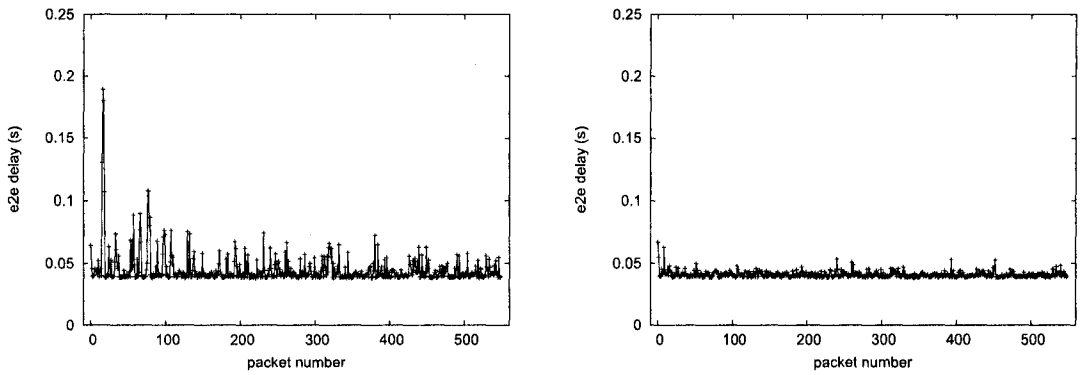
Next, we compared the performance of the red flows under different link utilizations and burst sizes of the blue flows. The main performance metric is the worst-case e2e delay. For each scenario, 36 independent runs were conducted. All the results are plotted with the 95% confidence

[1] Note that the burst size is a relative value: value $k$ means that the burst size is $k$ times of the default packet size (*e.g.*, if the packet size is 400B, then burst size 2 equals 800B).




(a) VAS (b) CAS

Figure 3.3: End-to-end delay comparison: WFQ



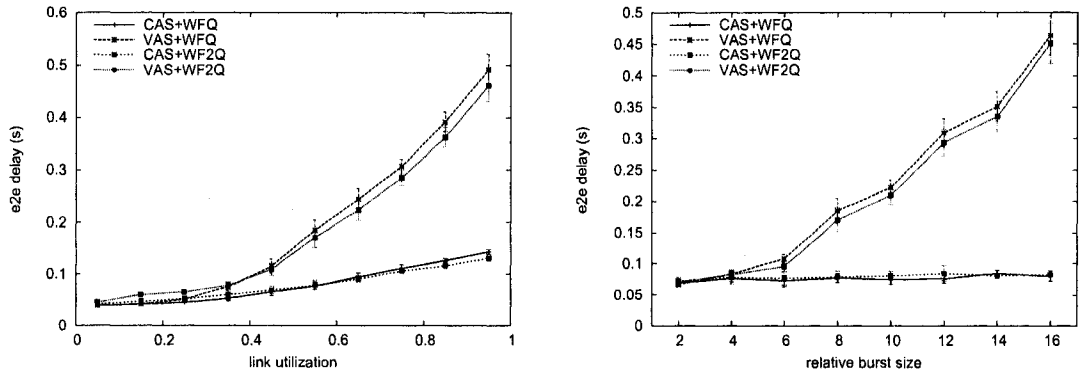

(a) VAS (b) CAS

Figure 3.4: End-to-end delay comparison: $WF^2Q$

interval [49].

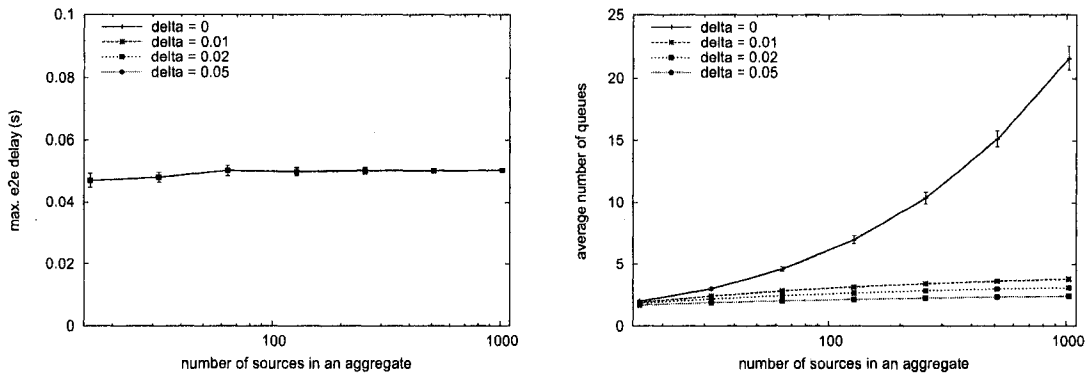
To see the robustness of the performance of CAS, we compared the performance of VAS and CAS under different link utilizations. As shown in Fig. 3.5(a), as the link utilization of the network links increases, the worst-case delay of the red flow under VAS increases significantly faster than that under CAS. This shows that CAS is more robust to high link utilization and congestion than VAS.

To examine the performance of CAS for large burst sizes of other flows sharing the same aggregate, we fixed the burst size of red flow at 2, and increased the burst size of the blue flows from 2 to 16. All the other parameters are set to the default values in Table 3.2. The e2e delay of the red flow is shown in Fig. 3.5(b). As can be seen, with the burst size of the blue flows increasing,




(a) Varying link utilization (b) Varying burst size

Figure 3.5: End-to-end delays under different conditions

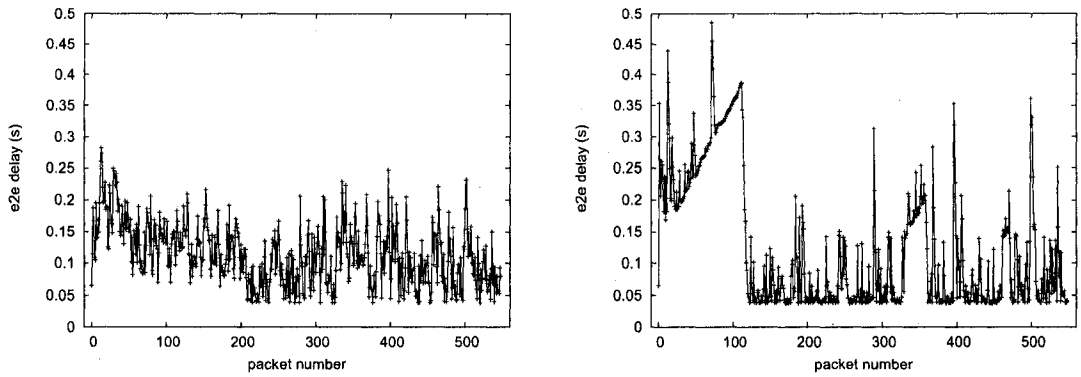



(a) maximum e2e delay (b) average number of queues

Figure 3.6: The effectiveness of the adaptive algorithm

the e2e delay of the red flow under VAS increases very fast. By contrast, the e2e delay under CAS changes very little, confirming the delay analysis in Section 3.1. The results under both WFQ and WF$^2$Q algorithms are very similar and thus consistent.

To examine the effectiveness of the adaptive algorithm, we also ran simulations with hop count set to 3 and total number of tagged flows set to 1024. Other parameters use the default values in Table 3.2. We varied the number of flows in each aggregate and monitored the maximum e2e delay and the average number of queues used by an aggregate at the second backbone router. The results are shown in Fig. 3.6. As can be seen from this figure, the adaptive algorithm can reduce the average number of queues significantly while almost maintaining the same maximum e2e delay (when $\delta_c = 0.01, 0.02$, or $0.05$).




(a) Per-flow scheduling

(b) Core-stateless (GRCore)

Figure 3.7: End-to-end delay comparison II: WFQ

## 3.4 Comparison with Related Work

### 3.4.1 Delay bound of aggregate scheduling

The delay bound problem of aggregate scheduling was also studied by Cobb [38]. By using rate-based scheduling algorithms and *fair aggregators*, he showed that the e2e delay of an aggregate is bounded and the bound can be smaller than the per-flow e2e delay bound. Our CAS scheme differs from the schemes in [38] mainly in how and where the burstiness of other flows in the same aggregate is controlled. In [38], Cobb used non-work-conserving scheduling algorithms at aggregators—both the *basic fair aggregator* and *greedy fair aggregator* use rate-controlled schedulers. By contrast, our scheme is work-conserving, allowing any work-conserving GR scheduling algorithm to be used at aggregators. Burstiness is controlled in a on-demand fashion, by reordering packets in the same aggregate *only* at the congested downstream nodes of the aggregator. The aggregator simply stores the lag information in the packets. Our scheme is general since any GR scheduling algorithms can be modified to become CAS algorithms.

### 3.4.2 Core-stateless scheduling

Recently, in an effort to solve the scalability problem of per-flow scheduling, there have been some work on core-stateless scheduling [58]. The key idea of core-stateless scheduling is to use per-flow scheduling (rate-based or delay-based) only at edge routers. At the same time, edge routers store some key per-flow information in the packets. Therefore, the core routers need not

keep per-flow information; all the needed information is carried in the packets themselves. Inside the network, core routers can restore the per-flow information from the packets and schedule them accordingly. By using this method, core-stateless scheduling can achieve the same delay bound as per-flow scheduling. The idea was first proposed by Stoica [58], and was generalized later by Kaur [59] and Zhang [60].

Our CAS scheme is similar to the core-stateless scheduling scheme in the sense that it also uses packets to carry scheduling related information. However, it differs from core-stateless schemes in the following aspects. First, core-stateless is totally stateless in the core network. All the needed information is carried in the packets themselves, from which the states of flows can be restored. By contrast, by keeping the states of traffic aggregates in core routers, CAS is still stateful, but the number of states is significantly smaller than that of per-flow fair queueing schemes (in orders of magnitude). Second, since CAS keeps states of traffic aggregates at core routers, it stores less information in the packets, with lag time $\delta$ only. In contrast, core-stateless scheduling generally needs to insert more information into packets. For example, CJVC (Core-stateless Jitter Virtual Clock) [58] needs to store four entries in each packet, with three of them relevant to scheduling (including the reserved rate for the flow). Since CAS stores less information in packets, the packet processing overhead is also relatively lower. Also, since core-stateless scheduling does not maintain any flow state at core routers, the admission control at core routers has to be based on traffic measurement or rate estimation [58], which has considerable overhead. CAS can use RSVP's extension for aggregation [11], which incurs smaller overhead at core routers. Third, core-stateless scheduling generally achieves the same delay performance as its corresponding per-flow scheduling. By taking advantage of multiplexing gains, CAS achieves tighter delay bounds than per-flow scheduling. Thus, CAS offers better delay performance. Finally, since CAS is not totally stateless at core routers, it has the advantage of isolating different traffic aggregates at core routers and confining the potential hazard problems (such as malicious traffic or denial-of-service attacks) within each aggregate, unaffecting other traffic aggregates.

To compare the performance of core-stateless scheduling with that of CAS, we repeated the first simulation in Section 3.3 using per-flow WFQ and GRCore, a work-conserving, core-stateless algorithm proposed in [59]. Under GRCore, edge routers perform per-flow scheduling (with such algorithms as VC and WFQ), and mark each packet with some per-flow state information (such as the flow's reserved rate and the packet's GRCore value). Core routers do not maintain any flow-related states. After receiving a packet, the core router first updates its GRCore value based on the

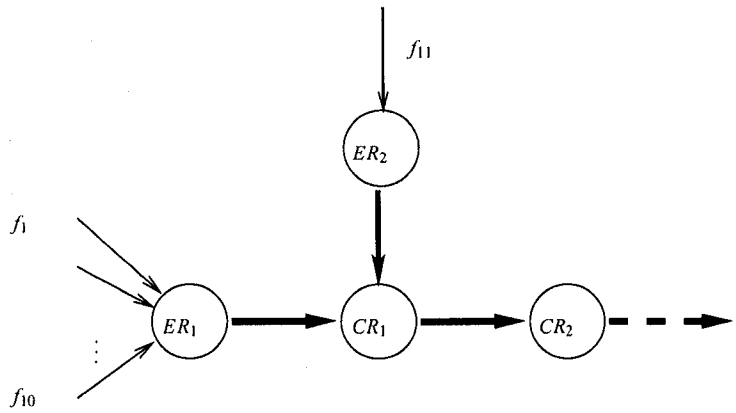



Figure 3.8: Network topology of the example

information carried in the packet. For packet $p_f^j$ at server $S_{i+1}$, the GRCore value is recursively defined as follows:

$$GRCore^{i+1}(p_f^j) = \begin{cases} GRC^i(p_f^j), & i = 0, \\ GRCore^i(p_f^j) + \beta^i + \tau^{i,i+1} + \max_{k\in[1,\ldots,i]} \frac{\ell_f^k}{r_f^k}, & i \geq 1. \end{cases} \tag{3.9}$$

Then, packets are scheduled using their GRCore values as deadlines/priorities. (Note that at an edge router, the GRCore value of a packet equals its GRC value, which is computed using Eq. (2.1)).

The simulation was conducted using the same network setup and parameters as described in Section 3.3. The results are plotted in Fig. 3.7. Comparison of Figs. 3.7 and 3.3 reveals a surprising result: *GRCore performs worse than both per-flow and aggregate scheduling.* We also ran the same simulation using WF$^2$Q and the results were similar. Interestingly, the delay results under GRCore using WFQ and WF$^2$Q are almost the same. This is because under GRCore, the deadlines of each packet under WFQ and WF$^2$Q are the same, and delays mainly occurred at the core routers. Thus, the difference in delay between WFQ and WF$^2$Q at the edge routers does not have much impact.

The reason why core-stateless does not perform well can be illustrated by the following example.

*Example*: As shown in Fig. 3.8, traffic comes into the network through two edge routers $ER_1$ and $ER_2$ before traversing core routers $CR_1$ and $CR_2$. Suppose 10 flows $(f_1, \cdots, f_{10})$ arrive through $ER_1$, each with reserved rate $r$, and one additional flow $(f_{11})$ arrives through $ER_2$, with reserved

rate $10r$. All the packets are of an identical size $S$, and all the links have identical capacity $C$ ($C >> r$) and propagation delay $\tau$.

Suppose all the routers are idle initially. Then, at time $t_0$, a burst of 20 packets $(p_{11}^1, p_{11}^2, \cdots, p_{11}^{20})$ from flow $f_{11}$ start arriving at $ER_2$. At the same time, 10 packets from the first 10 flows $(p_1^1, p_2^1, \cdots, p_{10}^1)$, one packet from each flow, also arrive simultaneously at $ER_1$. The first packet from the two bursts ($p_1^1$ and $p_{11}^1$) will both arrive at $CR_1$ at time $t = t_0 + \frac{S}{C} + \tau$.

We now examine the order of the outgoing packets at $CR_1$ under three scheduling schemes: per-flow, aggregate, and GRCore. WFQ is used as the scheduling algorithm in all three cases.

(i) Under per-flow scheduling, the 11 flows are scheduled by WFQ at $CR_1$, and hence, the output order is

$$p_{11}^1, p_{11}^2, \cdots, p_{11}^{10}, p_1^1, p_2^1, \cdots, p_{10}^1, p_{11}^{11}, \cdots, p_{11}^{20}.$$

That is, 10 packets from $f_{11}$ are transmitted first, then the packets from the first 10 flows, and then the next 10 packets from $f_{11}$.

(ii) Under aggregate scheduling, the first 10 flows are bundled into a single aggregate flow at $ER_1$ with reserved rate $10r$. At $CR_1$ this aggregate flow and $f_{11}$ are then scheduled by WFQ. Therefore, the output order is

$$p_1^1, p_{11}^1, p_2^1, p_{11}^2, \cdots, p_{10}^1, p_{11}^{10}, p_{11}^{11}, \cdots, p_{11}^{20}.$$

The packets from these two flows alternate before all the packets in the aggregate flow are transmitted.

(iii) Under GRCore, $ER_1$ and $ER_2$ schedule packets using per-flow WFQ, while $CR_1$ and $CR_2$ schedule packets based the GRCore values of the packets.

According to Eq. (3.9), the GRCore value for $p_1^1$ at $ER_1$ is $GRCore^1(p_1^1) = GRC^1(p_1^1) = t_0 + \frac{S}{r}$, so its GRCore value at $CR_1$ is $GRCore^1(p_1^1) + \frac{S}{C} + \tau + \frac{S}{r} = t + 2\frac{S}{r}$. Likewise, it can be shown that the GRCore values at $CR_1$ for $p_2^1, \cdots, p_{10}^1$ are all $t + 2\frac{S}{r}$, and the GRCore values at $CR_1$ for $p_{11}^1, p_{11}^2, \cdots, p_{11}^{20}$ are $t + 2\frac{S}{10r}, t + 3\frac{S}{10r}, \cdots, t + 21\frac{S}{10r}$.

Therefore, the output order at $CR_1$ is

$$p_{11}^1, p_{11}^2, \cdots, p_{11}^{19}, p_1^1, p_2^1, \cdots, p_{10}^1, p_{11}^{20}.$$

The first 19 packets from $f_{11}$ are transmitted before the packets from the first 10 flows, since they have smaller GRCore values.

From the above example, we can see that for the flows with lower reserved rates, aggregate scheduling has the best delay performance, while core-stateless scheduling performs the worst. The main reason for this is that under core-stateless scheduling, the packet deadline (such as the GRCore value) is computed rather conservatively, based on the flow's reserved rate. The actual arrival time of a packet at a core router is not considered in updating the deadline. At each core router, if other flows have burst, packets from low-rate flows are pushed back in the queue and thus have delay close to their deadlines. The fact that they may arrive earlier than other packets does not help, as long as the core router is not idle (in the above example, $p_1^1$ is transmitted after $p_{11}^{19}$ although it arrives at $CR_1$ much earlier). In this sense, core-stateless scheduling is biased against low-rate flows.

### 3.4.3 Coordinated scheduling

The idea of coordinating the scheduling of a packet at multiple nodes has also been explored in the literature. For example, the author of [61] proposed Coordinated EDF (CEDF) that coordinates the deadlines of a packet at multiple hops and yields a very small e2e delay. However, CEDF provides only statistical delay guarantees, and does not provide a natural way of assigning the deadline at each hop (the deadline at the first hop is randomly assigned). CAS is similar to CEDF in the sense that the scheduling inside an aggregate is based on EDF, and that it requires coordination among multiple nodes. However, by using the GRC value of a packet at the aggregator as its deadline at later hops, CAS provides a natural way of assigning deadline values to packets. In addition, CAS provides deterministic e2e delay guarantees.

In a related paper, the authors of [62] defined a general framework of *Coordinated Multi-hop Scheduling* (CMS), which covers many scheduling algorithms (including core-stateless algorithms [61, 58, 59]) exploring the coordination among different nodes. CAS's mechanism at the intra-aggregate level is also an example of CMS.

## 3.5 Summary of CAS's Features

The salient features of CAS are summarized as follows.

- *Scalable architecture*: by using traffic aggregation, CAS can support a large number of flows in core networks. Since CAS supports multiple aggregations, and the scale of traffic

aggregates (in terms of the number of flows in each aggregate) can be flexibly set. The architecture of aggregate scheduling also fits the Internet administration architecture well.

- *Superior performance*: as our analysis and simulation results have shown, CAS provides better performance than both per-flow fair queueing and VAS.

- *Low overhead*: first, since CAS belongs to aggregate scheduling, it has lower overhead than per-flow fair queueing (*e.g.*, lower state-maintenance overhead, simpler packet classification and scheduling). Second, with the optimization methods in Section 3.2, the extra overhead it has beyond VAS is marginal. Compared to VAS, CAS incurs a higher overhead at both the aggregator (computing and inserting lag values into packet headers) and core routers (packet sorting in each aggregate). For the former, the overhead is less than that in core-stateless fair queueing. For the latter, our simple multi-queue structure and adaptive algorithm have shown the extra overhead in packet scheduling to be marginal. Also, Stoica [58] has shown experimentally that the overall overhead of core-stateless fair queueing is not high—it "adds less than 5 $\mu$s overhead per enqueue operation, and about 2 $\mu$s per dequeue operation" on a 300MHz Pentium II machine. CAS has even smaller overhead than this.

- *Incremental deployment*: CAS facilitates incremental deployment. If some core routers do not implement CAS, then the queue at those nodes for each aggregate is FIFO. In other words, the scheduling at these routers becomes VAS. The performance will suffer, but still better than pure VAS. If the aggregator does not support CAS, as long as it supports fair queueing, it is still an aggregator of VAS. Then, the whole scheduling degrades to VAS, since the field to hold the lag value will have the default value for all packets, and the core routers will work in a FIFO fashion for each aggregate.

In addition to the features mentioned above, CAS is work-conserving and provides isolation between traffic aggregates.

## 3.6 Concluding Remarks

In this chapter, we proposed a novel coordinated aggregate scheduling (CAS) algorithm, which uses EDF within each aggregate and GR scheduling algorithms among traffic aggregates. The EDF scheduling within each aggregate is coordinated among multiple nodes. Under the assumption that

the incoming traffic to each aggregator conforms to the token bucket model, we proved that CAS provides tighter delay bounds for a flow than VAS. Moreover, CAS is shown to have many other salient features: for example, it is work-conserving and has small packet processing overhead. We have also shown by simulation that CAS is robust, performing better in terms of worst-case e2e delay than VAS, per-flow scheduling and core-stateless scheduling algorithms.

Table 3.1: Pseudocode of the adaptive algorithm

| | |
|---|---|
| $N_q$: | The current number of queues used by the aggregate; |
| $\delta_c$: | The default constant; |
| length[i]: | The length of queue i; |
| $VA_{max}$[i]: | The maximum value of VA in queue i; |
| $VA(p_A^j)$: | The virtual arrival time of packet $p_A^j$; |

```
1. //Upon arrival of a new packet p_A^j:
   i = 0;
   WHILE (VA(p_A^j) < VA_max[i] - δ AND i < N_q)
     i++;
   IF (i ≥ N_q) THEN
     N_q++;
     δ += δ_c;
   Insert p_A^j at the end of queue i;
   length[i]++;
   IF (VA(p_A^j) > VA_max[i]) THEN
     VA_max[i] = VA(p_A^j);

2. //Upon departure of a packet p_A^j:
   //Suppose p_A^j is from queue i;
   Remove p_A^j from queue i;
   length[i]--;
   IF (i = N_q-1 AND length[N_q-1] = 0) THEN
     N_q--;
     δ -= δ_c;
     VA_max[N_q-1] = 0;
```

Table 3.2: Parameters and their default values

| **Parameters** | **Values** |
|---|---|
| Tagged flow packet size | 400B |
| Cross traffic packet size | 1500B |
| Tagged flow rate | 32Kbps |
| Hop count | 10 |
| Burst size of tagged flows (red / blue) | 2 / 8 |
| Total number of tagged flows | 128 |
| Number of tagged flows in one aggregate | 16 |
| Link utilization | 55% |

# CHAPTER 4

# Differentiated BGP Update Processing for Improved Routing Convergence

Internet routers today can be overwhelmed by a large number of BGP updates triggered by events such as session resets, link failures, and policy changes. Such excessive updates can delay routing convergence, which, in turn, degrades the performance of delay- and jitter-sensitive applications. This chapter proposes a simple and novel idea of *differentiated processing* of BGP updates to reduce routers' load and improve routing convergence without changing the protocol semantics. Based on a set of criteria, BGP updates are grouped into different priority classes. Higher-priority updates are processed and propagated sooner, while lower-priority ones, not affecting routing decisions, can be delayed to both reduce routers' load and improve routing convergence. We first present a general methodology for update classification, update processing, and priority-state inference. By analyzing real BGP data obtained from Route Views, we show that our update classification is feasible and beneficial. We further propose two differentiated update processing (DUP) algorithms and evaluate them using the SSFNet BGP simulator on several realistic network topologies. The algorithms are shown to be very effective for large networks, yielding 30% fewer updates and reducing convergence time by 80%. Our scheme is simple and light-weight with little added processing overhead. It can be deployed incrementally, since BGP messages are not modified and every BGP router makes routing decisions independently.

## 4.1 Introduction

Real-time, multimedia applications such as IPTV, VoIP and Internet gaming are becoming popular. These delay- and jitter-sensitive applications impose more stringent requirements on the

underlying Internet routing system. In light of this trend, BGP (Border Gateway Protocol) routing issues have attracted significant attention from both the research and operator communities. A key problem associated with BGP is the excessive number of BGP updates possibly triggered by routing changes, such as session resets, link failures, and policy changes. For example, a recent study of a large tier-1 ISP shows that within just one minute, a "rich peering" router can experience hundreds of routing updates all at once partly due to the interaction between intra- and inter-domain routing [24].

There are several well-known schemes deployed to address this problem, including Minimum Route Advertisement Interval (MRAI), flap damping [63], Sender Side Loop Detection (SSLD), and Withdrawal Rate Limiting (WRATE) [64]. MRAI is a rate-limiting mechanism, enforcing a minimum inter-update interval between two neighbors (and for a specific destination prefix), in the hope that such a delay may help consolidate multiple related updates into fewer updates. Flap damping targets longer-term unstable routes, blocking routes changing too frequently over a relatively longer time period. Using a path-vector routing approach, BGP routers detect routing loops by checking if its own AS number appears in the AS path upon receiving a new route. SSLD, on the other hand, detects routing loops before sending the route to a neighbor BGP router. A rate-limiting mechanism, such as MRAI, is usually applied only to announcements, but not to withdrawals. However, some router vendors implement WRATE by applying MRAI to withdrawals as well, even though this is not recommended [64].

A related issue is the long convergence time caused by BGP path exploration. The authors of [25, 26, 27] have shown that the BGP convergence time is surprisingly long and depends on the length of the longest backup path. The convergence time is also shown to be proportional to the number of alternative routes to a given destination [28]. The prevalence of multi-homing in AS relationships (e.g., a customer AS peering with multiple providers ASes) [29, 30, 31] increases the number of backup routes in the Internet significantly, which, in turn, prolongs BGP convergence. Our recent study shows that the convergence time for one BGP Beacon [65] prefix is still surprisingly long—more than 30 minutes.

All the existing mechanisms, including MRAI and flap damping, are intended for all updates, except that in general, withdrawals are not subject to the influence of MRAI. In this chapter, we introduce the concept of *differentiated BGP update processing*, which classifies BGP updates and treats them according to their importance, which determines the update sending order and delay. We observe that BGP updates can be divided into two classes: the first class affects the routing

decisions of the receiving routers, possibly triggering more updates, while the second class does not affect the routing decisions of the receiving nodes, i.e., the best routes used are not changed. We regard the first class more important; but it is non-trivial to determine which updates belong to which class, since the routing decision and local policy of the receiving nodes are not directly available. We first define a method for classifying BGP updates. Updates in different classes will then be processed with different priorities. We also explore ways to process updates differently and propose two *Differentiated Update Processing* (DUP) algorithms.

In summary, we propose differentiation of updates depending on whether they are used in the forwarding tables of routers for related destination prefixes. If they are, they will more likely be processed with higher priority. The key ideas of our DUP algorithms are: (i) *Locally inferred routing preference*: when sending updates to a neighbor, a BGP router checks if the neighbor has sent updates for the same prefixes to itself. If so, it sends the updates with low priority. (ii) *Difference-based route selection*: when failure occurs and the best route to a destination is withdrawn, instead of selecting the next best available route, a BGP router first selects an interim route that shares the shortest common sub-route with the withdrawn route. The intuition behind the first idea is that if the neighbor also advertises a route, it must have an alternate route. The justification behind the second idea is that usually routes dissimilar to the withdrawn route are more likely valid during convergence.

The proposed scheme reduces the number of low-priority updates and the routing convergence time, thus reducing router (message processing, bandwidth) overhead, especially at an overloaded router in the core with rich peering. It reduces convergence time not only after a failure, but also during a new route propagation. At the same time, the scheme does not require any change to BGP protocol semantics (including the format of BGP messages and the final best route selection), thus facilitating incremental deployment. Moreover, it does not compromise reachability.

The authors of [66] proposed a Routing Control Platform (RCP), which uses a centralized routing control server to make route selection on behalf of each BGP router within a single AS, and distributes the routing decision to it. This RCP is to replace IBGP (Internal BGP) and solve many problems caused by its inefficiency. The authors showed that a prototype of such a system can be effectively implemented on a software router. Our scheme is simple and light-weight, and can be integrated into the RCP platform. Also, it can be easily implemented on software routers running XORP [67] or Zebra [68].

The rest of the chapter is organized as follows. Section 4.2 discusses related work on BGP

Table 4.1: Comparisons of BGP-enhancement schemes

| | Consistency assertion | Root-cause based schemes | Ghost flushing | Our scheme |
|---|---|---|---|---|
| change format of BGP messages | yes | yes | no | no |
| introduce extra messages | no | no | yes | no |
| reduce convergence time of updates for new prefixes | no | no | no | yes |

routing, putting our contributions in a comparative perspective. Section 4.3 presents the general idea of DUP and updates classification. Section 4.4 presents potential benefits of DUP using the Route Views data. Based on the general framework, Section 4.5 introduces the basic DUP algorithm. In Section 4.6 the basic algorithm is combined with a new route selection algorithm to further reduce the convergence time and the number of updates. Section 4.7 evaluates the DUP algorithms using simulation and compares their performance with the current BGP protocol and other BGP improvement schemes. Finally, Section 4.8 concludes.

## 4.2 Related Work

It is reported in [24, 69] that the current BGP may generate an excessive number of updates, especially when the network is overloaded, and the convergence of BGP may take too long to meet the requirements of real-time applications.

### 4.2.1 BGP Processing Overhead

The authors of [30] analyzed the rapid growth of BGP routing tables, as a result of several factors, such as load-balancing, the prevalence of multi-homing of small networks, and address fragmentation. The authors of [29] pointed out that not only is the Internet growing fast in size, it also becomes densely meshed at the inter-AS level. All these changes increase the BGP routing table size, thus increasing routers' processing overhead.

Studies have also shown that under certain (abnormal) conditions, there could be an excessive number of BGP updates. For instance, the authors of [70] studied the BGP behavior under the Slammer worm outbreak, and showed that during the attack, the number of BGP updates increased ten-fold, compared to that under normal condition. For some prefixes, the increase was by about 100 times. The authors of [69] studied the BGP behavior under heavy load and observed several weaknesses of the current BGP: its sensitivity to data congestion, global propagation of small local

changes, and slow convergence. The authors of [71] observed that in the time scale of minutes, BGP updates do not affect router's CPU load significantly, but in a shorter time scale of seconds, BGP can consume up to 100% of CPU cycles. The authors of [72] also showed that high update rates from multiple peers are harmful, prolonging the transit times of packets.

In addition to MRAI and flap damping, BGP Graceful Restart [73] is another mechanism deployed today that can reduce updates, but has limited applicability as it works only for short-lived session resets.

### 4.2.2 BGP Convergence Time

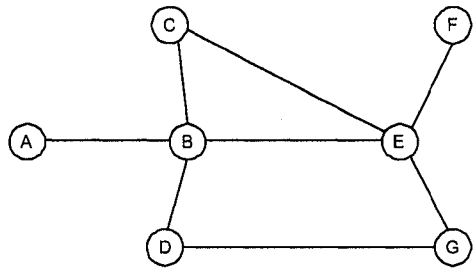
Using simulation, the authors of [74] demonstrated the effectiveness of MRAI in reducing the convergence time. They also observed that for a given topology, there exists an optimal MRAI timer value which minimizes the convergence time. However, the optimal value depends on the topology, so there is no universal optimal setting for the MRAI timer applicable for all routers and all types of routing changes.

In practice, Cisco routers use 30 and 5 seconds as the default MRAI timers for EBGP and IBGP sessions, respectively, while Juniper routers disable MRAI timer by default [75]. The study above showed that disabling MRAI timer may lead to large number of updates and long convergence time.

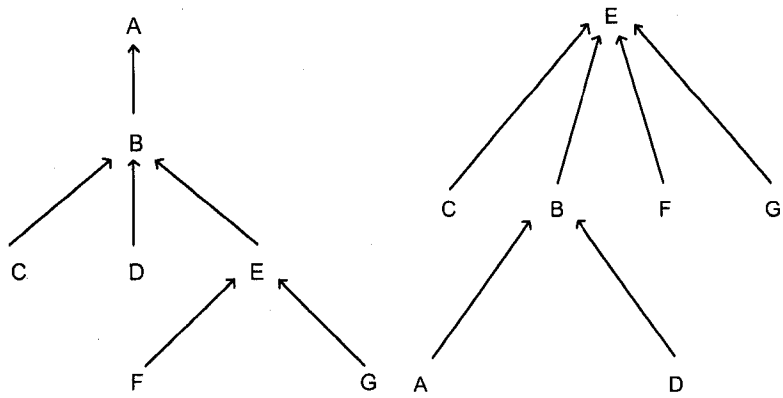
To reduce BGP update traffic and the associated overhead, and to decrease network convergence time, the authors of [76] proposed to add some *consistency assertion* checks to BGP update processing. From this checking, many updates are observed to contradict one another, and not all of them are valid. A set of assertion rules are defined to check the validity of updates, and block propagation of information on those routes that violate these rules, to other BGP peers. They have shown that by employing these rules, the number of BGP updates can be significantly reduced, and the route convergence time can be improved drastically.

The authors of [77, 78] extended the idea in [76] by embedding the root cause of a failure in updates, so a receiving node knows which candidate routes in its routing table are invalidated by the root cause, dramatically reducing the number of invalid routes and hence the convergence time. However, this scheme requires modification of BGP updates to embed the root cause information and slows deployment.

The authors of [79] proposed a different algorithm called "Ghost Flushing." The routes invalidated by a failure are called "Ghosts." For speedy removal of such invalid routes from the network,




(a) Network topology




(b) Forwarding-path tree for AS A (c) Forwarding-path tree for AS E

Figure 4.1: Forwarding-path tree of an AS

when a route is replaced by a less preferable one, if the route cannot be propagated because the MRAI timer has not expired, a withdrawal is sent immediately. By sending extra withdrawals, it is shown that invalid routes are removed much sooner and the convergence time is greatly reduced, especially when the original route is the only way to reach the destination—no valid alternative routes exist after the failure.

Table 4.1 compares our scheme with three existing schemes mentioned above. Our scheme does not require any modification to the BGP protocol semantics such as BGP message format, nor does it send extra messages. Moreover, unlike the above schemes which mainly focus on reducing convergence time after a failure, our scheme can also reduce convergence time during a new route propagation.

The author of [80] discusses various schemes (at different layers of network hierarchy) to achieve sub-second convergence and maintain high routes availability. The mechanisms capture the same differentiated processing idea as ours, but work mainly at the intra-domain level. In contrast, our scheme focuses on the inter-domain level.

## 4.3 General Methodology

### 4.3.1 Assumptions and Notations

Before delving into the details of our proposed approach, we introduce the following widely-used assumptions. Violation of these assumptions can be accommodated, especially in a single network where the policy information is known.

**A1.** In a peer-peer AS relationship, if ASes A and B are peers, A only sends B updates pertaining to itself and its customers; so does B. Routing updates learned from one peer will not be forwarded to other peers.

**A2.** In a customer-provider AS relationship, if A is a customer of B, A sends B only the updates pertaining to itself and its customers; B sends A the updates learned from all neighboring ASes.

**A3.** A BGP router prefers routes learned from customer to those learned from peers; it also prefers routes learned from peers to those learned from providers.

For simplicity, in addition to A3, we also assume that routing decisions are based on the AS path length by preferring shorter paths.

The following notation is used throughout the chapter. For a given BGP update, the sending router is called *sender*, and the receiving router *receiver*. If a router has multiple routes to reach a given destination AS, the one currently used is called *primary route*; other alternative routes are called *backup routes*. To avoid confusion, the term *peers* is used to indicate the two ASes with a peer-peer relationship between them, while neighboring BGP routers/ASes are called *neighbors*.

### 4.3.2 Per-prefix Forwarding-Path Tree

Before discussing BGP update classification, we introduce the concept of per-prefix *forwarding-path tree*[1]. When all the BGP routers in a network reach steady state, for a given destination prefix, at router level there is a forwarding-path tree inside the network. The destination itself is the root

[1]Here we assume that a router always send traffic for one prefix to the same neighbor. If this does not hold, the resulted structure would be a forwarding-path DAG (Directional Acyclic Graph). But this does not affect the following analysis.

of the tree, and its immediate neighbors are the first-level children, and so on. Directional links between different levels of nodes are trunks of the tree. Data packets heading for this destination prefix flow from the leaves to the root, along the trunks; while routing updates flow in the opposite direction, from the root to the leaves (Fig. 4.1). If updates are received through existing tree trunks, then we call them "on-tree" updates; otherwise, they are called "off-tree" updates. For each tree node, there is only one trunk reaching it from its parent node. Note that the tree structure is dictated by the routing in the network; when routing changes, the tree structure changes accordingly.

The key observation is that a BGP router can have many neighbors, thus receiving many updates regarding alternative routes to a given destination. However, for each destination prefix, there is only one on-tree update (regarding its primary route) from its parent node. Other updates are off-tree updates (regarding backup routes). For example, seven ASes are connected as shown in Fig. 4.1(a), each node representing an AS and also a BGP router. The forwarding-path trees for ASes A and E are illustrated in Figs. 4.1(b) and (c), respectively. In both trees the D-G link is an off-tree link.

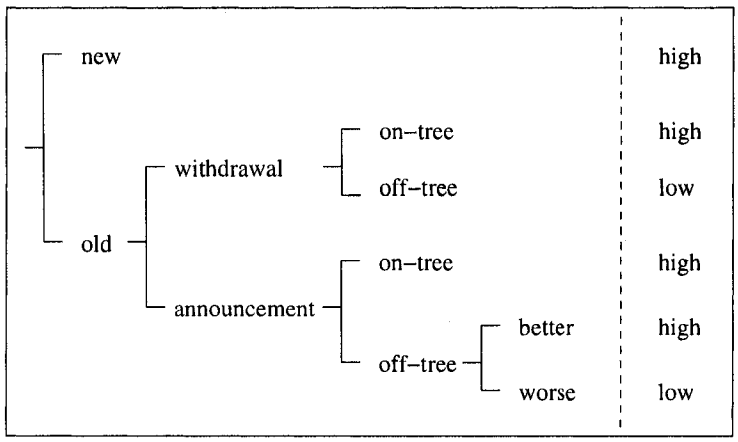



Figure 4.2: BGP update classification

In the current BGP, children nodes on the forwarding-path tree always know their parents, while parent nodes have no information about their children nodes (or no such information is used even if it is known at all). Our scheme attempts to gather such information and use it for BGP update processing.

### 4.3.3 Update Classification

The BGP update classification method is shown in Fig. 4.2. The updates are classified from the receiver's point of view. If the updates are about new destinations, which the receiver did not know before, they belong to a high-priority class. Otherwise, the updates are classified depending on whether they are on-tree in the forwarding-path tree of the associated destination prefix. We consider on-tree updates to be more important as they affect routing decisions.

As shown in Fig. 4.2, when an update is a withdrawal, then if it is on-tree, meaning that it withdraws a primary route of the receiver, then the update has high priority; otherwise, it has low priority since it withdraws a backup route which is not used by the receiver. When an update is an on-tree announcement, it implicitly replaces the primary route with a new route. Thus the update has high priority. If the update is off-tree, it has high priority only when it contains a better route than the primary route; otherwise, the update has low priority—since the route is not better than the primary route and will not be chosen.

### 4.3.4 Update Processing

Once updates are classified, the next question is how to differentiate their processing based on their priority class.

#### Receiver side—priority queues

Since the update classification is done from a receiver's perspective, the natural way is to process BGP updates using a priority queue on the receiver side. The advantage is that the receiver knows which class an update belongs to, by simply checking its forwarding table. However, to classify the updates into different priority queues, a BGP router has to check the content of the updates and do some pre-screening. This checking and pre-screening in a large part is repeating the default BGP update processing, thus incurring additional overhead. Also, this receiver-side method alone does not directly reduce the number of updates, as all updates are still transmitted to the receiver. Because of this, we focus on the sender-side scheme in the following discussion.

**Sender side—different delays**

An alternative way is to differentiate updates at the sender, using different delay timers (such as MRAI) for different classes of updates. There are two basic methods with different benefits. The first method is to process high-priority updates with default timers, while processing the low-priority ones with longer timers. This method can potentially reduce the number of low-priority updates transmitted; the second method is to process the low-priority updates with default timers, while processing the high priority ones with shorter timers. This method can potentially reduce the convergence time, since the high-priority updates experience a shorter delay at each hop.

However, since update classification is done from the receiver's perspective, the sender has no direct way to tell whether an outgoing update will affect the receiving router's forwarding table (or whether the update is an on-tree update for the receiver). Thus, the sender has to infer the class of updates. This class inference may incur some overhead on the sender side.

### 4.3.5 Update Class Inference

As mentioned above, when the sender side is involved in differentiated update processing, it has to infer the class of each update for the receiver. Discussed below are possible ways to achieve correct inference.

First, a router can infer the information externally: it can obtain the information from the data it receives from other routers, either implicitly (e.g., monitoring the data traffic passing through to determine if a neighbor is sending data packets to the related destination through it) or explicitly (e.g., letting the receivers of its updates send some feedback messages saying if the updates are being used as their primary routes). Note that inside a single AS, and hence for IBGP sessions, this information can be trivially obtained since the routing policy is consistent inside an AS.

A router can also infer the information internally, e.g., by checking its configuration and/or routing table. For example, when a router $R$ has an update for a destination prefix, it can examine if there is a route entry for the same prefix received from a neighbor in the routing table. If there is, then the neighbor is using a different route; otherwise, it is using the route learned from $R$. However, if routers filter out some outgoing updates based on local policies, then this method will overestimate the number of high priority updates. On the other hand, this scheme does not cause any reachability problem.

Since the external inference methods in general incur more overhead and require extra memory

Table 4.2: BGP update classification results

| $AS_1 \rightarrow AS_2$ | Total | Duplicates | Withdrawals (high) | Withdrawals (low) | Announcements (high) | Announcements (low) |
|---|---|---|---|---|---|---|
| 1239 → 7018 | 45689 | 1709 (4%) | 5679 (12%) | 8747 (19%) | 3850 (8%) | 25640 (56%) |
| 7018 → 1239 | 37298 | 15697 (42%) | 2934 (8%) | 4139 (11%) | 2363 (6%) | 12127 (33%) |
| 3561 → 3356 | 37718 | 4448 (12%) | 1748 (5%) | 10733 (28%) | 2617 (7%) | 18123 (48%) |
| 3356 → 3561 | 47664 | 6811 (14%) | 2663 (6%) | 9087 (19%) | 1526 (3%) | 27551 (58%) |
| 3549 → 5511 | 38188 | 0 (0%) | 3948 (10%) | 10516 (28%) | 3234 (8%) | 20365 (53%) |
| 5511 → 3549 | 26843 | 4979 (19%) | 1815 (7%) | 5309 (20%) | 755 (3%) | 13947 (52%) |
| 2497 → 1668 | 12983 | 3764 (29%) | 1928 (15%) | 1817 (14%) | 1939 (15%) | 3462 (27%) |
| 1668 → 2497 | 12276 | 183 (1%) | 1323 (11%) | 1128 (9%) | 1898 (15%) | 7354 (60%) |
| 3303 → 13237 | 15890 | 3189 (20%) | 373 (2%) | 3328 (21%) | 395 (2%) | 8565 (54%) |
| 13237 → 3303 | 34076 | 19051 (56%) | 1217 (4%) | 1809 (5%) | 2620 (8%) | 9025 (26%) |
| 1668 → 7018 | 12276 | 183 (1%) | 1676 (14%) | 762 (6%) | 1700 (14%) | 7552 (62%) |
| 7018 → 1668 | 354110 | 176547 (50%) | 11476 (3%) | 21569 (6%) | 17433 (5%) | 126843 (36%) |
| 3303 → 1239 | 15890 | 3189 (20%) | 1063 (7%) | 2582 (16%) | 1241 (8%) | 7719 (49%) |
| 1239 → 3303 | 194716 | 8796 (5%) | 10096 (5%) | 5267 (3%) | 67017 (34%) | 99715 (51%) |

to store the inferred data, in the following discussion, we focus on internal (or local) inference.

## 4.4 Empirical Data Analysis

To examine empirically the amount of BGP updates that can be classified as low-priority, we analyze the routing data collected by the Route Views project [81]. Each Route Views router peers with BGP routers in many ASes to collect BGP routing data. The data are collected in two forms: RIB files and update files. The RIB files contain the contents of the forwarding tables of all the BGP neighbors, and are collected every two hours; the update files contain new updates from the BGP neighbors every 15 minutes. The peering sessions between a Route Views router and its neighbors are different from the peering sessions between two Internet routers in two aspects: (i) the Route Views router is a listener only, i.e., it does not propagate its own routing data to its neighbors, (ii) in most cases, its neighbors send all updates to it, without any filtering. This implies that the Route Views router receives more updates than an Internet router normally does.

We chose the routing data collected by router RouteViews2 spanning five days, from 05/01/2006 to 05/05/2006. For each day, we used all the available RIB files (maximum 12). For each RIB file, we selected the update file collected right after the RIB file, such that the updates (route changes) are based on the RIB file just collected. Each RIB file and its matching update file constitute a *data set*.

Then, we chose seven AS pairs, all of which are peering with the router RouteViews2. For

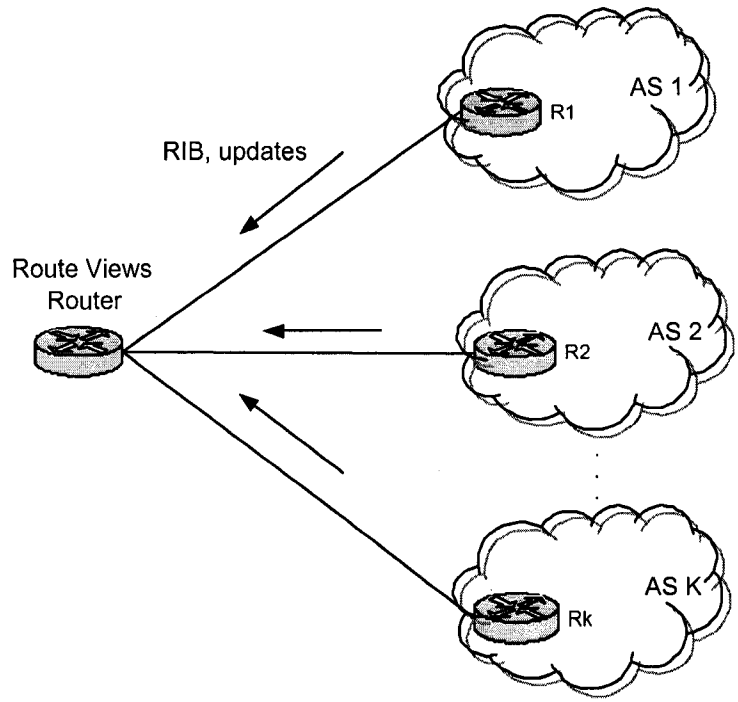



Figure 4.3: Illustration of Route Views peering sessions

each pair, we examined the updates between the two ASes and evaluated how many of the updates can be classified as low priority, according to the classification criteria discussed in Section 4.3. The chosen AS pairs cover both peer-peer and customer-provider AS relationships. For the peer-peer relationship, we study three tier-1 pairs (7018-1239, 3561-3356, and 3549-5511) and two tier-2 pairs (2497-1668 and 3303-13237); for the customer–provider relationship, we study two pairs (1668-7018 and 3303-1239). The AS relationship and tier classification are based on the data from [82].

### 4.4.1 Data sanitation

From the RIB files and the update files, we first inferred the updates exchanged between the two ASes of each pair. While the RIB files collected by the Route Views router reflect the contents of forwarding tables of the related BGP routers correctly, the update files are not exactly what has been exchanged between the two BGP routers—they contain more updates. Therefore, for each pair of ASes we studied, we need to filter out those updates that are not exchanged between the two ASes based on the assumptions stated in Section 4.3.

### 4.4.2 Analysis method

For a given data set and a pair of ASes, $AS_1$ and $AS_2$, we performed the analysis in the following four steps.

**S1:** From the RIB file, we extracted the forwarding table for $AS_1$ and $AS_2$ routers, $Rib_1$ and $Rib_2$. Then, from the update file, we extracted the updates sent from $AS_1$ and $AS_2$ routers, $Upd_1$ and $Upd_2$.

**S2:** From the AS relationship data [82], we collect the sets of customer ASes for $AS_1$ and $AS_2$, $Cust_1$ and $Cust_2$.

**S3:** From $Upd_1$ and $Cust_1$, we obtained all the updates learned through $AS_1$'s customer ASes, $Upd_1^{new}$, which is the actual updates sent from $AS_1$ to $AS_2$ in the real-world. Likewise, we obtained $Upd_2^{new}$.[2]

**S4:** From $Rib_1$ and $Upd_1^{new}$, we know exactly how the forwarding table of the $AS_1$ router evolves during a 15-minute span. Then, from $Upd_2^{new}$, we can compare whether an update from $AS_2$ is necessary by checking if the related route is for a new destination prefix for the $AS_1$ router, or whether the route used by $AS_1$ is through $AS_2$. If either of the two holds, the update is classified as high-priority; otherwise, it is classified based on the AS relationship between $AS_1$ and $AS_2$ and the AS relationship between $AS_1$ and its current next-hop AS for the related prefix (Assumption **A3**) and the AS-path lengths of $AS_1$'s current primary route and the route in $Upd_2^{new}$. The new update has high priority if its route is better than $AS_1$'s current route; otherwise it has low priority. The updates from $AS_1$ to $AS_2$ can be analyzed similarly.

### 4.4.3 Analysis results

As listed in Table 4.2, updates are classified into two categories: announcements and withdrawals. We found that (i) excluding duplicates, a large proportion of announcements (about 50% on average) can be classified as low priority, and thus processed separately. This confirms the value of our DUP scheme; (ii) a significant percentage of updates are pure duplicates, meaning

[2]Here we assume that $AS_1$ and $AS_2$ are peers, so that only customer routes are exchanged between them. If $AS_1$ and $AS_2$ are in the customer-provider relationship and $AS_1$ is a customer $AS_2$, then $Upd_2^{new} = Upd_2$ ($AS_2$ sends every update to $AS_1$).

that the updates are same as the previous ones regarding the same destinations. Also, the number of duplicates varies: some senders (such as those in ASes 7018 and 13237) send a huge number of duplicates, while others (such as those in ASes 1668 and 3549) send very few or no duplicates. We conjecture that this behavior depends on the implementation of the routers being used by different ASes and might be the result of routing changes associated with the non-transitive attributes, as mentioned in [69]. Note that the duplicates are not counted as part of low-priority updates; (iii) a significant portion of withdrawals are low priority as well, meaning that they are withdrawing backup, not primary routes.

## 4.5 DUP: Differentiated Update Processing

Based on the general discussion in Section 4.3, we design a *Differentiated Update Processing* (DUP) algorithm based on the sender-side scheme. To reduce overhead, a sender infers the priority of an update based on its local information. Also, since this inference is not always accurate, all the withdrawals are still treated the same, without any differentiation.[3] The algorithm takes the following two steps.

**Step 1**: it checks the AS relationship between a local $AS_1$ and a neighboring $AS_2$, and infers whether $AS_2$ is using $AS_1$'s route to forward traffic. For example, if $AS_1$ and $AS_2$ have a peer-peer relationship, then if $AS_1$ has no route from $AS_2$ (meaning that $AS_2$ has no route at all or has a route via its provider/peer AS), the route has high priority; else, the route has low priority. ($AS_2$'s current route must be via its own customer AS, which is favored over $AS_1$'s route since $AS_1$ is a peer AS of $AS_2$.)

If the AS relationship alone can not decide the priority of $AS_1$'s new route, it goes to Step 2 to compare the lengths of the two routes.

**Step 2**: the lengths of $AS_1$'s new route and the route from $AS_2$ are compared. If from $AS_2$'s point of view, the new route is shorter than $AS_2$'s current route, then it has high priority; else, it has low priority.

The details of the algorithm are listed in Table 4.3. Note that (i) in the algorithm we assumed known AS relationship information between a BGP router and its neighbor, which can be easily stored in a router as a configuration parameter or directly inferred from the configurations; (ii) the AS paths' length comparison is made from $AS_2$'s point of view: if at $AS_1$ the two paths have length

[3] In practice, withdrawals generally are not subject to the control of MRAI timer anyway.

Table 4.3: Pseudocode of DUP algorithm: DUP_Send()

| | |
|---|---|
| $ASP_1^{pre}$: | AS path of the previous route for D sent to $AS_2$; |
| $ASP_1^{new}$: | AS path of the newly-chosen route for D; |
| $ASP_2$: | AS path of the route for D received from $AS_2$; |
| len (): | length of an AS path; |

```
01:  // AS1 sends an update for D to AS2:
02:  If ASP2 = Null, Then
03:     sends the update with shorter MRAI timer;
04:  Else
05:     If AS1 is a peer or provider of AS2, Then
06:        sends the update with longer MRAI timer;
07:     Else // AS1 is a customer of AS2
08:        If ASP1^pre = NULL, Then
09:           sends the update with shorter MRAI timer;
10:        Else
11:           If len (ASP1^pre) ≠ len (ASP1^new) And
12:              len (ASP1^new) + 2 < len (ASP2), Then
13:              sends the update with shorter MRAI timer;
14:           Else
15:              sends the update with longer MRAI timer;
```

len ($ASP_1^{new}$) and len ($ASP_2$), then at $AS_2$ the lengths become len ($ASP_1^{new}$)+1 and len ($ASP_2$)-1; (iii) two MRAI timers are used in the DUP algorithm (one more than the current BGP protocol for each BGP neighbor), one for high-priority updates ($MRAI_{short}$) and the other for low-priority updates ($MRAI_{long}$). If timer $MRAI_{long}$ expires, the router will send low-priority updates and reset the timer; if timer $MRAI_{short}$ expires, the router will send high-priority updates and reset the timer. Depending on the purpose of the algorithm, two options can be implemented. The first option is to use the default MRAI value for $MRAI_{short}$, and a larger value for $MRAI_{long}$. The goal is to reduce the number of BGP updates exchanged between BGP routers, by holding low-priority updates longer. The second option is to use the default MRAI value for $MRAI_{long}$, and a smaller value for $MRAI_{short}$. The goal is to shorten the convergence time of BGP in the network, by speeding up the propagation of high-priority updates. In this chapter we focus on this second option.

Table 4.4: Pseudocode of Simpler DUP: DUP_Send()

| $ASP_2$: | AS path of the route for D received from $AS_2$; |
|---|---|

```
01:  // AS1 sends an update for D to AS2:
02:  If ASP2 = Null, Then
03:     sends the update with shorter MRAI timer;
04:  Else
05:     sends the update with longer MRAI timer;
```

### 4.5.1 Simpler DUP

The DUP algorithm above takes advantage of the knowledge of AS relationship between a BGP router and its neighbor when classifying the priority of an update. The classification process appears complex and may not be always accurate, as ASes do not always choose routes based on the guideline of AS relationships.

To overcome this issue, we introduce a simpler version of DUP without relying on the knowledge of AS relationships. It considers one thing only: whether a neighbor has already propagated route for the same prefix to the local AS. If the neighbor has not already done so, the update has high priority; otherwise, it has low priority. The details of the algorithm are listed in Table 4.4.

### 4.5.2 Priority misclassification

Since sender-side scheme is used, neither the DUP algorithm nor its simpler version matches exactly the update classification method in Fig 4.2. If a sending node has not received a route from its neighbor, it means (i) the route is for a new prefix; or (ii) the neighbor is already using the local AS's route; or (iii) the neighbor is using another route without notifying the local AS. While in the first two cases the outgoing update is on-tree, in the third case it is clearly off-tree. Thus our algorithms may set an update's priority higher and send it earlier than necessary. On the other hand, if the sending node has already received a route from the neighbor, then the outgoing update must be off-tree. The question is whether it is better than the neighbor's current route from the neighbor's perspective. The simpler version simply set the update as low priority, thus may delay the update longer than the default BGP. The DUP algorithm is more accurate, classifying the priority based on the AS relationship between the two neighbors and the AS path lengths of the new route and the route from the neighbor. However, if some ASes do not follow the guidelines of

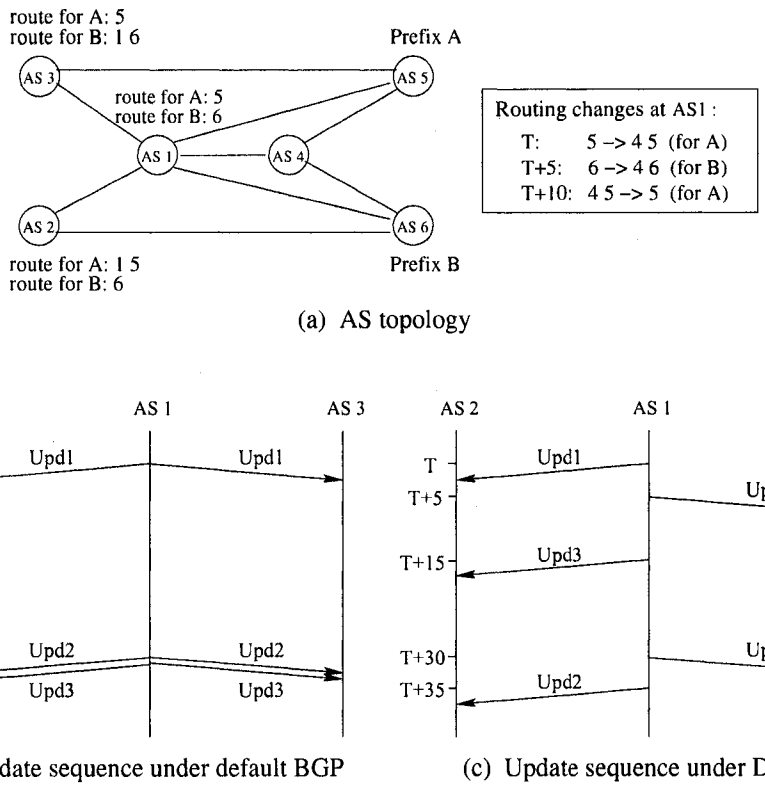



Figure 4.4: Illustration of DUP algorithm

AS relationships discussed in Section 4.3, it may misclassify the priority also. On the other hand, since in this case the neighbor already has a route, it does not affect the neighbor's connectivity. The only drawback is that the neighbor may use a less preferred route slightly longer. In addition, since we focus on speeding up the high priority update while sending the low priority ones using the default MRAI value, this is not an issue at all: low priority updates are not delayed longer than the default MRAI.

This simpler version does not consider AS relationships, nor does it compare AS path lengths. Thus it incurs less overhead. From simulation studies, it works quite well. Therefore, in the following discussion, we use the simpler version to represent the DUP algorithm.

The following example illustrates the advantages of our DUP algorithm. As shown in Fig. 4.4, both $AS_2$ and $AS_3$ are connected to $AS_1$. Prefix A is located in $AS_5$; prefix B is located in $AS_6$. Suppose to reach destination A, $AS_2$ uses $AS_1$'s path, while $AS_3$ does not; to reach destination B, $AS_3$ uses $AS_1$'s path, while $AS_2$ does not. Now suppose $AS_1$ first changes its path to A at time T, then changes its path to B at T+5, and changes its path to A again at T+10. Under the current BGP,

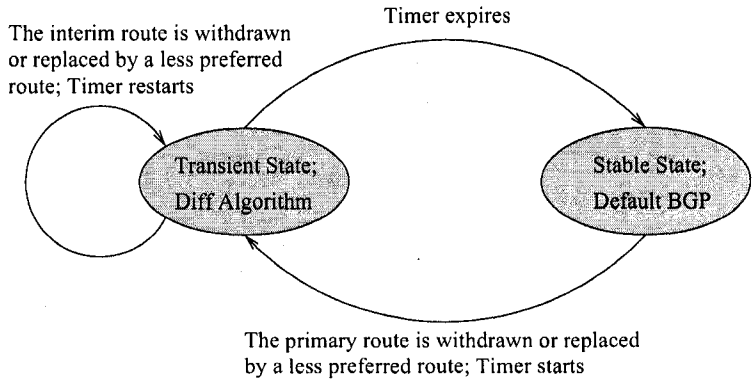



Figure 4.5: Transition diagram between stable and transient states

$AS_1$ uses one MRAI timer for each neighbor. It sends update for the first change ($Upd_1$) to both $AS_2$ and $AS_3$ at T, and sends updates for the second and third changes ($Upd_2$ and $Upd_3$) to both $AS_2$ and $AS_3$ at T+30 (after the MRAI timers expire). Under the DUP algorithm, in contrast, $AS_1$ uses two MRAI timers for each neighbors. It sends $Upd_1$ to only $AS_2$ at T: since $AS_3$ is not using $AS_1$'s route, the update to $AS_3$ has low priority and thus is delayed; then for the same reason, at T+5 (not T+30: since $AS_1$ does not send $Upd_1$ to $AS_3$ at T, the MRAI timer for high-priority update is not reset then) it sends $Upd_2$ to $AS_3$ only. At T+10, $Upd_3$ replaces $Upd_1$, so $Upd_3$ is sent to $AS_2$ at T+15 after the MRAI timer for high-priority update expires. $Upd_3$ is sent to $AS_2$ at T+30 after the MRAI timer for low-priority update expires, and $Upd_2$ is sent to $AS_3$ at T+35 after the MRAI timer for low-priority update expires. Therefore, the DUP algorithm not only sends important updates ($Upd_2$ to $AS_3$ and $Upd_3$ to $AS_2$) sooner, it also reduces the number of updates: under DUP, only five updates are sent. Since $Upd_2$ ($Upd_1$) is not used by $AS_2$ ($AS_3$), its delay does not affect routing decision.

## 4.6 DUP+: Enhanced DUP

Note that in the above discussion, we assumed that all the updates contain valid routes. While this assumption holds during the propagation of new routes, it does not hold in case of a network component failure which may invalidate some existing candidate routes, and these invalid routes may propagate through the network during the transient period after the failure. This propagation of invalid routes caused by BGP path exploration is the main culprit for the extraordinarily long convergence time after a failure, unnecessarily triggering excessive routing changes at the same

time.

Also, since invalid routes are propagated, the actual best route after a failure may not be shorter than some invalid routes being sent/received earlier; picking shorter routes only as the high-priority ones (as in the Step 2 of the DUP algorithm) is suboptimal. The update classification needs to be improved. Ideally we can distinguish valid routes from invalid ones, so that valid routes can receive higher priority than invalid routes, thus speeding up the convergence.

### 4.6.1 Difference-Based Route Selection

To reduce the convergence time and the number of updates exchanged after a network failure, we propose a new route-selection algorithm that selects a new route based on the difference between the candidate routes and the original (withdrawn) route: when the original route is withdrawn or replaced by a less preferred route (e.g., due to a network failure), the selected route is the shortest one with the *maximum difference* from the original route, which is not necessarily the best of the remaining routes.

As shown in Fig. 4.5, we define transient and stable states for a prefix. After an original route is withdrawn or replaced by a less preferred route (usually resulting from a network failure), the router enters the transient state. In this state, the routing decision process for the prefix does not select the best remaining route; instead, it selects an interim route with the minimum similarity to the original route—the one that shares the shortest common AS path segment with the original one. Then, the AS path of the original route is stored as *susceptible AS path segment*, since the failure occurred to this path. At the same time, a timer is started to indicate how long the router has stayed in the transient state. When the timer expires, the router switches from the transient state to the stable state, and the best route is recomputed using the default BGP route-selection algorithm. However, if the existing interim route is withdrawn or replaced again during the transient state, (i) the timer is restarted, and (ii) the existing interim route is used to update the susceptible AS path segment. This new algorithm used during the transient state is called *Diff*.

The details of the Diff algorithm are given in Table 4.5. Currently, the length of the timer ($t_p$ in the code) is set to 1.5 times the MRAI timer value. The idea is to set it long enough to catch consecutive updates from a single BGP neighbor. Also, during the same transient state, if the interim route is withdrawn or replaced by a less preferred route, the LCS (longest common subsequence) of the interim route and the susceptible AS path segment is computed (recursively) to get the new susceptible path segment, which has the effect of narrowing down the location of

Table 4.5: Pseudocode of Diff algorithm

| | |
|---|---|
| $ASP$: | AS path of route $R$; |
| $P$: | destination prefix of route $R$; |
| $ASP_{lcs}$: | susceptible AS path segment; |
| lcs ($asp_1$, $asp_2$): | longest common subsequence of two AS paths; |

```
01:  // Route R is withdrawn or replaced by a less preferred route :
02:  If P is in default stable state, Then
03:     ASP_lcs = ASP;
04:     start the timer t_p;
05:  Else
06:     ASP_lcs = lcs (ASP_lcs, ASP);
07:     restart the timer t_p;
08:  select the shortest route sharing the minimum LCS with ASP_lcs.
```

the failure. A variable $ASP_{lcs}$ is used to store the susceptible AS path segment.

The intuition behind Diff is that during the transient state, the routes are not stable and many invalid routes are propagated. These invalid routes are closer and more similar to the original routes; potential valid routes are more different from the original routes. Thus, during the transient state, we select an interim route which is more likely to be valid. Although this route is not necessarily the best remaining route, as long as it is valid, it guarantees the reachability of the destination. The propagation of this valid route also helps other routers to converge to a valid route faster. After the transient state is exited, most (if not all) of the invalid routes are removed from the routing table, and a new best route can be selected. Therefore, the Diff algorithm skips invalid backup routes and selects the shortest one with the largest difference from the original route, which is more likely to be a valid route. This significantly shortens the path exploration process and generates fewer routing updates.

Suppose the original route is $\{AS_1 \ldots AS_k\, AS_{k+1} \ldots AS_m\}$, and a failure occurs between $AS_k$ and $AS_{k+1}$. Then, all the invalid routes after the failure contain the path segment $\{AS_k\, AS_{k+1} \ldots AS_m\}$. By using difference-based route selection, the new algorithm potentially chooses a valid route faster, favoring routes that do not contain that path segment. On the other hand, a route containing segment $\{AS_k\, AS_{k+1} \ldots AS_m\}$ is not necessarily an invalid route, since there could be multiple peering sessions between $AS_k$ and $AS_{k+1}$.

Note that the Diff algorithm uses two criteria to select an interim route: maximum difference (which is equivalent to shortest LCS) and shortest AS path length: it chooses the route that shares

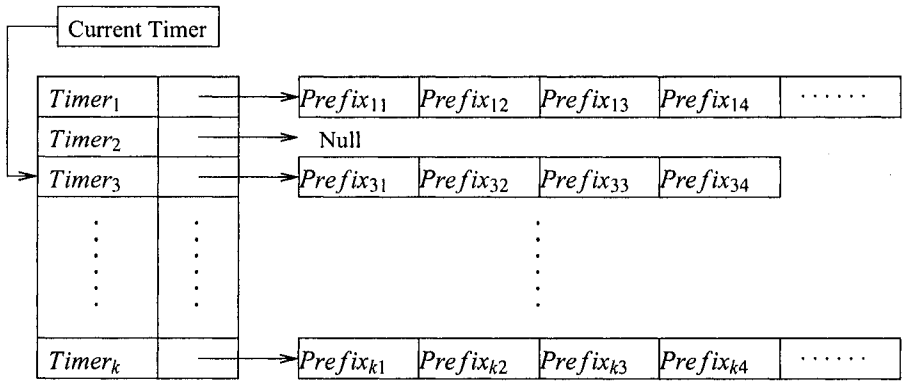



Figure 4.6: The global timer structure

the shortest LCS with the original route first. If multiple routes share the same shortest LCS, then the one with the shortest AS path will be chosen. In addition, the LCS algorithm here is different from the classic LCS algorithm in that the resulting common subsequence must start from the origin of the AS paths and must contain contiguous ASes. For example, given two routes, $\{AS_4\ AS_3\ AS_2\ AS_1\}$ and $\{AS_3\ AS_4\ AS_2\ AS_1\}$, both originating from $AS_1$, according to our algorithm, their LCS is $\{AS_2\ AS_1\}$, not $\{AS_3\ AS_2\ AS_1\}$ or $\{AS_4\ AS_2\ AS_1\}$. Our algorithm compares the AS numbers at the corresponding locations only, starting from the original AS.[4] Thus the computation complexity is linear with respect to the lengths of the AS paths.

Since Diff only changes how a route is selected, all the other mechanisms of the default BGP still remain, including loop detection. Therefore, the routers will converge under Diff if they do under the default BGP.

### Global timer scheme

In the discussion above, we assume that each prefix has a separate timer. We call it *individual timer scheme*. Although each timer is created on demand—only when a prefix is affected by changes in the network, and it is deleted whenever the routes for the prefix converge, the scheme may still incur significant overhead in practice when a failure affects a large number of prefixes.

To reduce the overhead incurred by the timers, we design a new *global timer scheme* that deploys a fixed number of global timers for all the prefixes. As shown in Fig. 4.6, the scheme

[4]If prepending is used by the two routes, redundant AS numbers must be removed from their AS paths before the comparison.

works as follows: when a BGP router starts, it automatically starts $K$ timers. The expiration time of the timers are set as $\frac{T}{K}, 2\frac{T}{K}, 3\frac{T}{K}, ..., (K-1)\frac{T}{K}$, and $T$, respectively, where $T$ is the full length of an expiration period. The gap between two neighboring timers' expiration time is $\frac{T}{K}$. There is a pointer locating the current timer, which is always the one with the longest expiration time. For each timer, there is also a queue associated with it. Whenever a prefix is affected by a change in the network, it is put in the queue associated with the current timer until the timer expires.

When a timer expires, (i) if its queue is not empty, all prefixes in the queue go through the route recomputation process to select the best routes, after which they are removed from the queue; (ii) the timer is reset with length $T$; (iii) the current timer pointer points to this newly reset timer.

Since the current timer is updated whenever a timer expires, its expiration time is always about $T$ time away (more accurately, the time value is between $\frac{(K-1)T}{K}$ and $T$). Also, a prefix is always put in the queue of the current timer. Therefore, for each prefix, its timer will expire in $t$, where $\frac{(K-1)T}{K} \leq t \leq T$.

Instead of a timer for each prefix, only a small number ($K$) of timers are maintained here, which are shared by all the prefixes. The larger $K$ is, the more closely the scheme simulates the individual timer scheme. Since $T$ equals 1.5 times of the MRAI timer value (as discussed above), we set $K$ as 9. Thus $\frac{T}{K}$ is one sixth of the MRAI timer value. From simulation tests, the two timer schemes work almost identically.

### 4.6.2 DUP+

Combining Diff with DUP, we can send potentially invalid routes using a longer MRAI timer while sending valid route using a shorter MRAI timer. This new algorithm is called DUP+, which is an extension of DUP. Under DUP+, the algorithm determines whether a prefix is in stable or transient state: if it is in stable state, the DUP algorithm will be executed; if it is in transient state, it will compare LCS values instead to determine the priority of an update. The details of the sender-side DUP+ algorithm are given in Tables 4.6. Note that DUP+ is a superset of DUP. In line 3 of Table 4.6, the corresponding code for DUP is called.

Table 4.6: Pseudocode of DUP+ algorithm: sender side

| | |
|---|---|
| $ASP_1^{new}$: | AS path of the newly-chosen route for prefix D; |
| $ASP_2$: | AS path of the route for D received from $AS_2$; |
| $ASP_{lcs}$: | susceptible AS path segment; |
| lcs ($asp_1$, $asp_2$): | longest common subsequence of two AS paths; |

```
01:  // AS1 sends an update for D to AS2:
02:  If the prefix is in stable state, Then
03:     call DUP_Send();
04:  Else
05:     If ASP2 = Null, Then
06:        sends the update with shorter MRAI timer;
07:     Else
08:        If lcs (ASP1^new, ASP_lcs) < lcs (ASP2, ASP_lcs), Then
09:           sends the update with shorter MRAI timer;
10:        Else
11:           sends the update with longer MRAI timer;
```

### Overhead

DUP+ introduces some extra overhead in terms of state maintenance and processing load. In Diff algorithm, once a prefix is in transient state, it needs to maintain its susceptible AS path segment. This consumes $2L$ bytes of memory for each path segment, where $L$ is the length of the AS path segment and is generally a single digit. On the other hand, these states are required only for the prefixes that are affected by failures, not for every prefix. By studying Route Views data, we found that typically an AS makes routing changes for less than 1000 prefixes during a 15-minute period. Thus the total required memory space is less than $2000L$ bytes. By using the global timer scheme, Diff also needs to maintain an array of global timers and their associated prefix queues; but the number of timers and their queues is a small fixed number. In addition, DUP+ maintains an extra MRAI timer for each neighbor.

DUP+ also incurs extra processing overhead. The LCS computation in Diff only occurs during failures, and its complexity is $O(n)$, where $n$ is the length of the longest common subsequence of the two AS paths. All the other functions are just simple modification of existing BGP, which contain only a few dozen lines of code. Thus the extra overhead is small.

Table 4.7: Parameters and their default values

| Parameters | Values |
|---|---|
| Link delay | 0.01-0.1 (sec) |
| $MRAI_{long}$ (for low priority) | 30 (sec) |
| $MRAI_{short}$ (for high priority) | 15 (sec) |
| MIN_PROC_TIME | 0.01 (sec) |
| MAX_PROC_TIME | 0.5, 0.1 (sec) |
| Number of advertisers | 3 |
| Length of timer $t_p$ (used by Diff algorithm) | 1.5 * $MRAI_{long}$ |
| Number of global timers | 9 |

## 4.7 Evaluation

To demonstrate the benefits of the DUP+ algorithm,[5] we evaluate it using SSFNet's BGP simulator [83], a Java-based simulator widely used for studying BGP performance (e.g., [74, 76, 78]).

### 4.7.1 Simulation Design

We studied two scenarios: new route propagation and link failure. The performance metrics used are valid network convergence time, average valid convergence time, and the number of updates exchanged.

First, we define valid convergence time, based on which the other two definitions follow:

**Definition 4.7.1 (Valid convergence time).** *The valid convergence time of a router is the length of the time interval ($t_e$, $t_c$], where $t_e$ is the time when the origin router sends out the first update messages, and $t_c$ is the time instant after which the router always has valid routes (through which the destination is reachable).*

Note that a router may switch from one valid route to another after its valid convergence time being reached.

**Definition 4.7.2 (Valid network convergence time).** *The valid network convergence time is the length of the time interval ($t_e$, $t_{nc}$], where $t_e$ is the time the origin router sends out the first update messages, and $t_{nc}$ is the time instant after which all the routers in the network always have valid routes.*

[5] Since DUP+ is a superset of DUP, we used only DUP+ in the evaluation.

The valid network convergence time is in fact the worst valid convergence time among all the nodes.

**Definition 4.7.3 (Average valid convergence time).** *The average valid convergence time is the average over the valid convergence times of all the nodes in the network.*

Compared to the commonly-used definition for convergence time [74], the valid (network) convergence time more accurately captures the reachability of a node (the network) to the destination and the impact of routing changes on application traffic. Similar definitions have also been proposed, such as *next-hop convergence time* [79] and *data-plane convergence time* [84]. Compared to them, the valid convergence time is easier to measure.

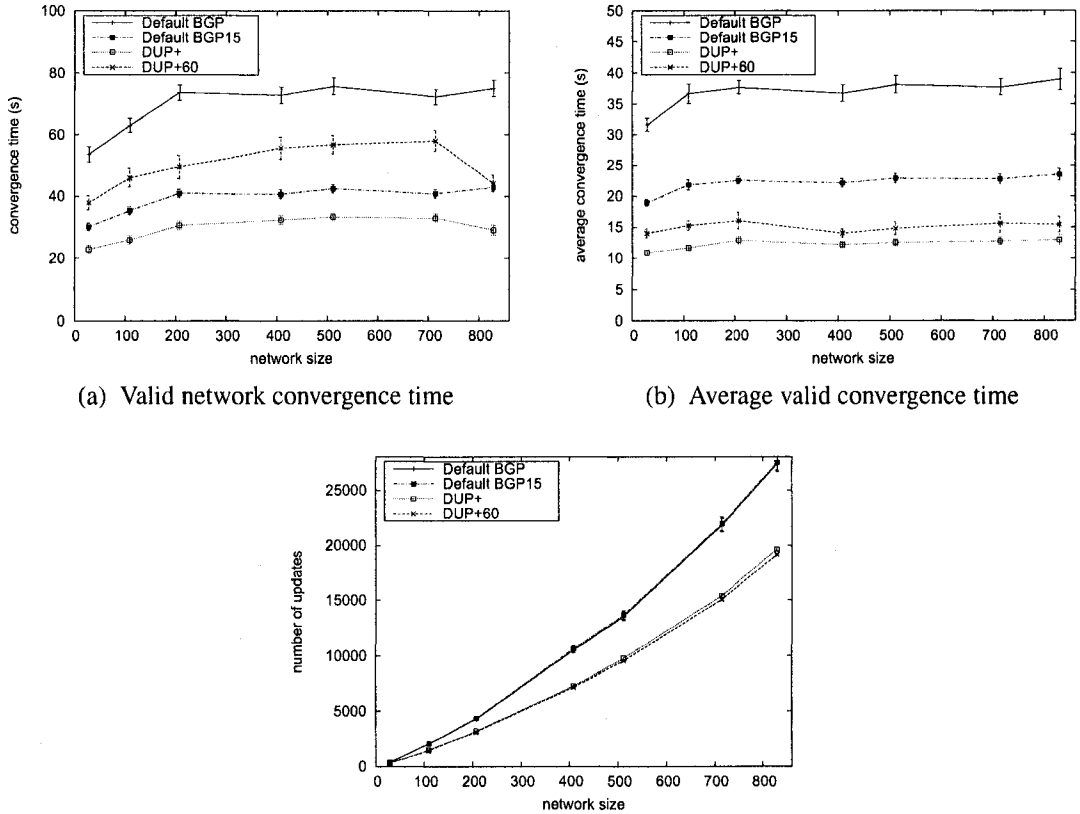
Table 4.7 describes the simulation parameters and their default values. The link delay was randomly set between 0.01s and 0.1s. In the SSFNet simulator, the CPU processing delay for each packet is simulated to be a random value between two thresholds—MIN_PROC_TIME and MAX_PROC_TIME. They were set to 0.01s and 0.5s, respectively. (We also used 0.1s for MAX_PROC_TIME and found that the relative performance of DUP+ is very similar). For simplicity, we always set $MRAI_{short}$ to be half of $MRAI_{long}$.

To test some realistic network topologies, we used the multi-AS topology generating package from SSFNet [85], which contains seven different topologies based on the Internet BGP routing table. The number of nodes contained in the topologies ranges from 29 to 830. For each topology, we chose $N$ nodes as advertisers, which announce their own IP prefixes to the network. Among all the nodes in each topology, we only chose those with a small number of neighboring nodes (less than or equal to four) in the simulation, meaning that they are more likely to be at the edge of the network. The default value of $N$ is 3. For each simulation scenario, we randomly picked six instances of advertisers; for each of them, we conducted six independent runs with different random seeds. Therefore, there are 36 runs for each scenario. The results are summarized as the average value of the 36 runs, and are presented using 95% confidence interval.

### 4.7.2 Results

#### New route propagation

We first tested the case that new routes are propagated, and compared DUP+ with the default BGP. To study the interactions among routing changes, the three advertisers started at different times with a delay of a few seconds between them. We focused on the performance of the last




(a) Valid network convergence time

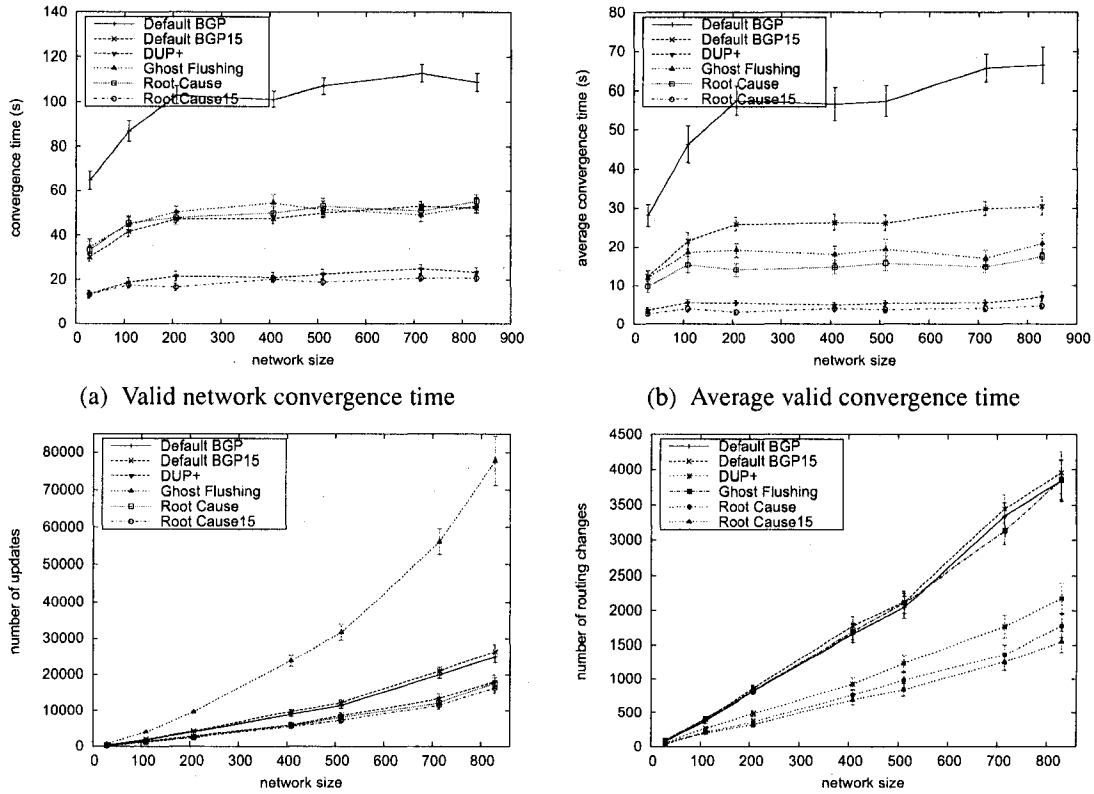
(b) Average valid convergence time

(c) Number of updates

Figure 4.7: Performance comparison between default BGP and DUP+: new route propagation

advertiser; the other two were considered as the generators of cross traffic.

As shown in Fig. 4.7, the DUP+ algorithm yields very short valid network convergence time and average valid convergence time—only 40% or less of those for default BGP. At the same time, the number of updates is also smaller, saving about 30% of updates. This clearly shows the benefits of DUP+. For comparison, we also ran default BGP with MRAI timer of 15 seconds (called "default BGP15"), and DUP+ with MRAI timer of 30 (60) seconds for high (low) priority updates (called "DUP+60"). As the results show, DUP+60 still outperforms default BGP, and its average convergence time is even shorter than "default BGP15". The clearly shows the benefits of differentiated processing.




(a) Valid network convergence time

(b) Average valid convergence time

(c) Number of updates

(d) Number of routing changes

Figure 4.8: Performance comparison between default BGP, DUP+, Ghost Flushing and Root Cause: failure case

### Link failure

We also evaluated the DUP+ algorithm under the link failure scenario, and compared it with the default BGP, Ghost Flushing, and Root-Cause based scheme. In addition to the default MRAI value of 30 seconds, we also used the value of 15 seconds for the default BGP (called "default BGP15") and Root Cause (called "Root Cause15") to see how well they perform with a shorter MRAI. For each topology, we randomly picked a node with a small number of peers as a test node. The test node first advertised some network prefixes to all its neighbors. For a given prefix, it advertised different AS path lengths to different neighbors by prepending. After the network initially converged, we broke the link between the test node and one of its neighbors, and observed how the algorithms perform. As in the new route propagation case, two additional nodes started advertising their prefixes shortly before the failure, injecting cross traffic.

As shown in Fig. 4.8, DUP+ has shorter valid network/average convergence time than not only both Ghost Flushing and default BGP (as much as 80%), but also Root Cause, which is a little surprising. The reason is that although Root Cause selects only valid route after failure by removing invalid route very fast, it propagates valid route the same way the default BGP does; in contrast, DUP+ propagates valid route with a shorter timer, which overcomes that fact that it is slower in selecting valid route. By using a shorter MRAI value of 15 seconds, we can see that "Root Cause15" indeed performs the best, but DUP+ is very close to it.

Under DUP+ the number of updates is very small as well. Again the number is very close to that for Root Cause. Although Ghost Flushing has shorter convergence time than the default BGP, the number of updates under it is surprisingly large. By checking the simulation data, we found that most of the updates are the extra withdrawals triggered by mechanisms of the Ghost Flushing algorithm.

In addition to the above three performance metrics, we also examined the number of routing changes under each algorithm and found the DUP+ algorithm to incur 50% fewer routing changes than the default BGP and Ghost Flushing, leading to more stable routes. In contrast, it is not as good as Root Cause since it is not as efficient in selecting valid routes; but the difference is rather small.

In summary, not only does DUP+ outperform the default BGP and Ghost Flushing, its performance exceeds or comes close to that of the Root-Cause based scheme as well. This shows that the performance of the current BGP can be significantly improved using our scheme without changing BGP message format required by Root-Cause based approaches.

## 4.8 Concluding Remarks

In this chapter, we presented a simple and novel way of differentiating BGP updates based on their impact on inferred routing decisions of the receiver. It is shown to make significant improvements in both reducing the routers' overhead of processing an excessive number of BGP updates, as well as reducing routing convergence time. The proposed scheme is simple to implement, requires no modification of BGP protocol semantics, and can be deployed incrementally.

So far, we have mainly focused on EBGP (External BGP) sessions (for BGP neighbors belonging to different ASes). Inside a large AS, there are also many IBGP (Internal BGP) sessions. However, the default MRAI timer for IBGP is only 5 seconds, much shorter than that for EBGP.

In addition, all the IBGP routers in the same AS are either directly peering with each other, or peering through route-reflectors. Therefore, routing delay inside an AS is generally shorter than that between ASes. Thus, our scheme does not modify the processing of updates between IBGP nodes.

# CHAPTER 5

# Impact of BGP Routing Changes on Application Traffic

## 5.1 Introduction

As discussed in the previous chapter, BGP has been shown to suffer from long convergence delays and to generate many updates (some containing routes already invalidated by failures). Such routing disruptions can have significant impact on emerging, popular real-time applications like VoIP and IPTV. For example, recent work [86] showed that BGP routing changes degrade performance of VoIP applications. It is thus important to understand the general impact of BGP routing changes on application traffic. Specifically, we would like to answer the following questions: (1) how much impact do routing changes have on application traffic? (2) which routing changes cause more impact? and (3) can we devise new metrics to measure the impact, instead of just using update count, to improve routing decisions?

We take a two-step approach to this problem. First, we study it at the AS level, examining how many ASes are affected by a given update. To do this, we first obtained AS relationship information containing the structure of the Internet hierarchy. Based on this hierarchy the number of ASes affected is analyzed. We then collect BGP data from multiple vantage points. At each vantage point, given an update, the current routing table of that vantage point, and the underlying AS-level hierarchy, we infer the number of ASes in the network affected by this routing change. Note that an update may be a withdrawal or an announcement of an alternative route. A withdrawal has more severe impact since the affected ASes may have no alternative routes, while an announcement only forces the affected ASes to change their routes. We can also correlate the updates observed at multiple vantage points and narrow down the source of a routing change, from which one can obtain a more accurate set of affected ASes.

The second step is to study failure impact at the traffic level, examining how much application traffic is affected by a given routing change. Ideally we need to measure the interdomain traffic matrices and map them onto the BGP routing hierarchy to determine the amount of traffic affected by a BGP routing change. However, such Internet-wide traffic matrices are difficult to obtain [87], because ISPs generally are unwilling to disclose traffic-volume statistics of their networks. For this reason, we used case studies to focus on the other readily available traffic information including a local ISP's traffic statistics and the rankings of the top Internet sites. We therefore focus on the impact of routing changes on a regional ISP in addition to popular Internet Web sites. Note that our framework enables any ISP to perform intelligent route selection to reduce traffic impact caused by routing disruptions based on local information.

In summary, our main contributions are the following. i) We present a novel algorithm to infer the set of affected ASes from a single vantage point based on observed routing updates. ii) We evaluate the algorithm using empirical BGP data and find that updates from top tiers affect more ASes. iii) Using a local ISP Network's traffic and routing data, we study the impact of routing changes on user traffic and find that top prefixes with high traffic volume in general have few routing changes, in particular much fewer routing withdrawals. iv) We propose a new algorithm of delaying adoption of certain routing changes, so that the amount of traffic shift will be minimized. We test the idea by using the local ISP's routing data and find that it can save as much as 30% of routing changes for top prefixes with high traffic volume.

The rest of the chapter is organized as follows. Section 5.2 describes an analytical method for studying the AS-level impact of BGP routing changes and presents the results of AS-level analysis using the RouteViews data [81]. Section 5.3 discusses the traffic-level impact of routing changes using traffic data from the local ISP Network. We describe related work in Section 5.4 and conclude in Section 5.5.

## 5.2 AS-Level Analysis

The first step is to infer which ASes are affected by a given routing change based on the AS-relationship hierarchy among all the ASes. An AS is deemed *affected* if its forwarding decision is changed because of a routing update. The basic idea of the analysis algorithm is to identify all ASes with alternative routes and thus, will not be affected by the routing change. Here we assume that the alternative routes have priority over the changed route to provide a conservative estimate.

AS relationships are classified into three types: customer-provider, peer-peer, and sibling-sibling. For simplicity, we ignore the sibling-sibling relationship, since two ASes with this relationship can be merged into a single AS.

We rely on the following widely-used assumptions [88].

**A1.** If ASes A and B are peers, A only sends B updates pertaining to itself and its customers; so does B. Routing updates learned from one peer will not be forwarded to other peers.

**A2.** If A is a customer of B, A sends B only updates pertaining to itself and its customers; B sends A the updates learned from all neighboring ASes.

**A3.** A BGP router prefers routes learned from its customers to those learned from its peers, and it prefers peer routes to provider routes.

### 5.2.1 Analysis Algorithm

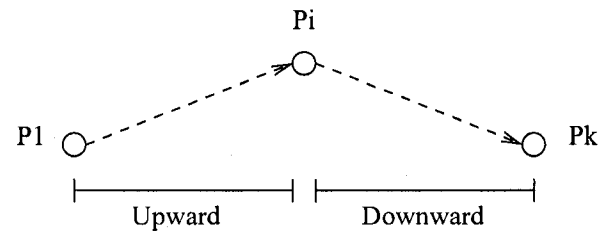
The algorithm below is conservative and relies on two underlying assumptions: 1) an observed failure occurs at a local AS's immediate link; 2) if an AS has an alternative path, that path is preferred compared to the path through the local AS. With these two assumptions and the inferred AS relationships, the algorithm underestimates the number of affected ASes and provides a lower bound for it. More accurate estimate can be obtained by inferring the cause for the update, but our simple algorithm serves our purpose of identifying highly disruptive routing updates.

The analysis algorithm works as follows:

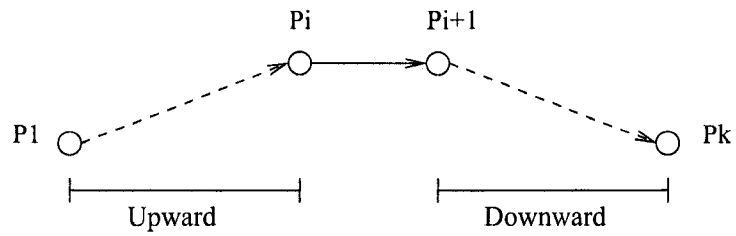
1. For an AS (we call *local AS* the AS that receives an update), it identifies the AS's customer list, single-homed customer list, and its current RIB from the AS relationship data and the RIB or BGP table data.

2. Given an update UPD for prefix P, it checks the current route for P in the RIB, $R_{old}$. Suppose $R_{old}$'s AS path is $\{p_1, p_2, \ldots, p_k\}$.

3. It divides $R_{old}$'s AS path into two segments: the upward and downward segments. As shown in Fig. 5.1, the path $\{p_1, p_2, \ldots, p_k\}$ can be in two forms: the first form is $\{p_1, p_2, \ldots, p_i, \ldots, p_k\}$, where $p_j$ is a customer of $p_{j+1}$ for $1 \leq j \leq i-1$, and $p_j$ is a provider of $p_{j+1}$ for $i \leq j \leq k-1$; the second form is $\{p_1, p_2, \ldots, p_i, p_{i+1}, \ldots, p_k\}$, where $p_j$ is a customer of $p_{j+1}$ for $1 \leq j \leq i-1$, $p_j$ is a provider of $p_{j+1}$ for $i+1 \leq j \leq k-1$, and $p_i$ is a peer of



(a) Case 1: one top-tier AS



(b) Case 2: two top-tier ASes

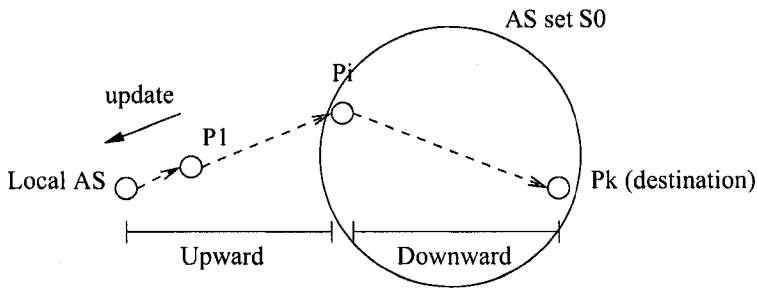
Figure 5.1: BGP path segments

$p_{i+1}$. In the first form, the downward segment contains $\{p_i, \ldots, p_k\}$; in the second form, it contains $\{p_{i+1}, \ldots, p_k\}$.

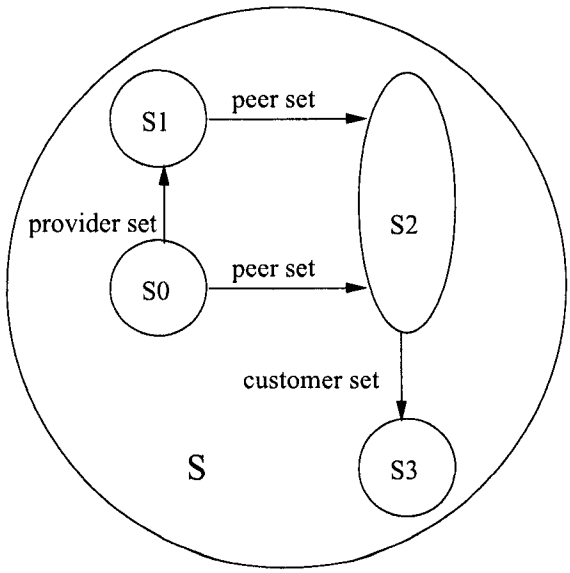
4. If UPD is a withdrawal, then it checks all the ASes on $R_{old}$'s downward segment (we call it set $S_0$). First, obtain set $S_1$, which contains all the providers of those ASes; then, obtain set $S_2$, which contains all the peers of the ASes in sets $S_0$ and $S_1$; next, obtain set $S_3$, which contains all the customers of the ASes in sets $S_0$, $S_1$, and $S_2$. Let set $S = S_0 \cup S_1 \cup S_2 \cup S_3$. In essence, set $S$ contains all the ASes that have alternative routes to reach the destination. ASes in set $S$ are not affected, but all the other single-homed customers of the local AS (if UPD is from a peer or provider) or all the other ASes (if UPD is from a customer) are. Fig.5.2 illustrates the relationship of local AS, the update received, and the various AS sets.

   Note that (i) sets $S_0$, $S_1$, $S_2$, and $S_3$ exclude the local AS itself; (ii) the affected AS set includes the local AS, since routing changes are observed there; (iii) the algorithm uses only the downward segment, because by checking all the providers, peers and the customers, the upward segment will be covered automatically. Also, if the algorithm uses the ASes in the upward segment, it can no longer check their providers and peers; otherwise, the resulting AS path are not "valley-free".

5. If UPD is an announcement, and its AS path is $\{q_1, q_2, \ldots, q_i\}$. Similar to steps 3 and 4,




(a) Relationship of local AS, received update and AS set $S_0$




(b) Relationship of various AS sets

Figure 5.2: Relationship of the local AS and various AS sets

we call the set of ASes in the downward segment of the path set $S'_0$, and eventually obtained set of ASes set $S'$. There are two cases here: 1) if $R_{old}$'s AS path is NULL, then UPD is a new route. All the ASes in set $S'$ are not affected, but all the other single-homed customers (if UPD is from peers or providers) or all the other ASes (if UPD is from customers) are affected; 2) if $R_{old}$'s route is not NULL, we obtain two sets from the two AS paths: $S$ and $S'$. Then, ASes in set $S$ or $S'$ are not affected, but all the other single-homed customers (if both UPD and $R_{old}$ are from peers or providers) or all the other ASes (UPD and $R_{old}$ are from customers) are. If UPD is from peers or providers and $R_{old}$ is from customers, then all the other ASes are affected; if UPD is from customers and $R_{old}$ is from peers or providers, then all the other single-homed customers are. So, $R_{old}$ is the deciding factor; UPD is irrelevant.

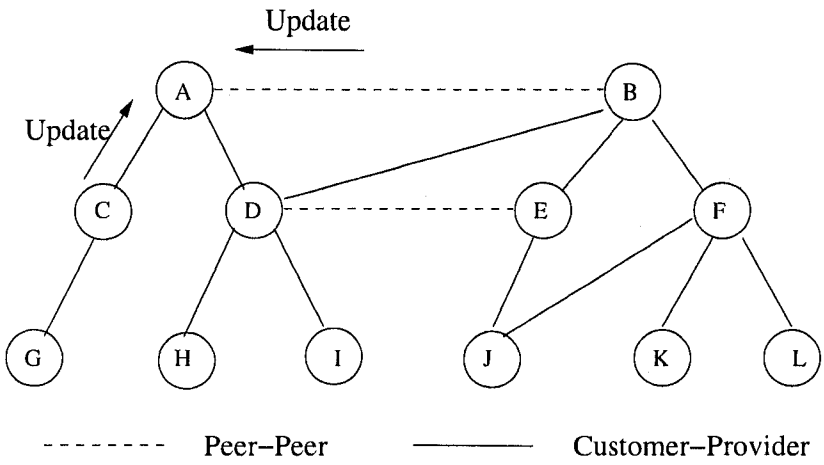



Figure 5.3: An example of a routing update's affected ASes.

To better understand the algorithm, let's consider the following example. Suppose we have an AS hierarchy illustrated in Fig. 5.3: A and B are tier-1 ASes; C, D, E and F are tier-2 ASes; and the rest are tier-3 ASes. For AS A, its single-homed customers are C and G; its customers are C, D, G, H, and I. If A receives a withdrawal from B for a prefix located in L, then the original route for the prefix is {B, F, L}. Next, we obtain $S_0$ = {B, F, L}, $S_1$ = {B, F}, $S_2$ = NULL (since A is the local AS, it is excluded from the set $S_2$). Then, $S_3$ = {D, E, H, I, J, K}, and $S$ = {B, F, L, D, E, H, I, J, K}. Since B is a peer of A, only the single-homed customers of A that are not included in set $S$ are affected, which are C and G (A itself is also affected). Note that here the algorithm is conservative and assumes the failure occurs between A and B, in which case all the ASes in set $S$ are not affected. If the failure occurs between B and F, for example, then ASes in set {B, D, E, H, I, J} are affected as well. If the failure occurs between F and L, then F and K too are affected.

Using the same example, suppose A receives a withdrawal from C for a prefix located in G, then the original route for the prefix is {C, G}. Next, we obtain $S_0$ = {C, G}, $S_1$ = {C} (since A is the local AS, it is excluded from the set $S_1$), $S_2$ = NULL. Then, $S_3$ = NULL, and $S$ = {C}. Since C is a customer of A, all the other ASes reachable by A but not included in set $S$ are affected, which are {A, B, D, E, F, H, I, J, K, L}. Similarly, the algorithm is conservative and assumes that the failure occurs between A and C, in which case all the ASes in set $S$ are not affected. If the failure occurs, for example, between C and G, then C is affected as well.

Table 5.1: AS relationship data comparison

| | Gao | SARK | CAIDA |
|---|---|---|---|
| Total number of relationship | 56006 | 55863 | 47140 |
| Customer-provider | 47147 | 53029 | 42606 |
| Peer-peer | 8611 | 2834 | 4284 |
| Sibling-sibling | 247 | - | 250 |

### 5.2.2 AS Relationship Inference

For the above analysis, we first need to obtain the AS relationship data for the global Internet. We were able to find the data from three sources: 1) data based on the new AS relationship inference (called Gao's) algorithm in [89], which is an updated version of Gao's original algorithm [88]; 2) data based on SARK's AS relationship inference algorithm [82] and 3) CAIDA's data [90]. For the first two data sources, we first collected routing tables from about three dozen public *route servers* [91] and obtained a large number of AS paths. Then, we applied the two inference algorithms to the collected AS paths to obtain the AS relationship data. For the third data source, we did not have the algorithm but obtained the AS relationship data directly from CAIDA.

As listed in Table 5.1, a majority of the relationships are the customer-provider type, just as expected. Sibling-sibling relationships are rather rare; SARK's algorithm does not have this classification at all. Furthermore, these three data sets do not exactly agree with one another. Their differences are captured in Table 5.2. As can be seen, between two data sets, less than 90% of relationships are shown to match each other. Note that "Gao-extra" means the number of relationship entries appear in the data set obtained from Gao's algorithm, but not the other data set in comparison.

To resolve the discrepancy, we used only Gao's algorithm and CAIDA's data, since these two are shown to be more accurate due to fewer inconsistent relationships. We first applied Gao's algorithm to obtain the AS relationship information. We then selected a subset of the AS relationship results that match CAIDA's data. Next, we used the subset as an initial input for Gao's algorithm to improve accuracy. We used the output as our AS relationship data.

Table 5.2: Difference in AS relationship inference

| | Match | Different | Gao-extra | SARK-extra | CAIDA-extra |
|---|---|---|---|---|---|
| Gao vs. SARK | 46372 (82.8% / 83%) | 9263 (16.5% / 16.6%) | 370 (0.66%) | 228 (0.4%) | - |
| Gao vs. CAIDA | 43903 (78.4% / 93.1%) | 3102 (5.5% / 6.6%) | 9000 (16%) | - | 135 (0.29%) |
| SARK vs. CAIDA | 41507 (74.3% / 88%) | 5441 (9.7% / 11.5%) | - | 8915 (16%) | 195 (0.41%) |

### 5.2.3 Analysis Results

Using RouteViews routing data from May 2006, we observed routing updates during a two hour period at 36 vantage points and analyzed their impact in terms of the number of affected ASes. These 36 ASes and their tiers are listed in Table 5.3. In summary, there are 6 tier-1 ASes, 26 tier-2 ASes, and 4 tier-5 ASes. The tier classification is based on the commonly used definition [82], where tier 1 refers to ISPs without any providers and tier 5 corresponds to stub networks without any customers.

Since multiple updates can be triggered by a single event, prior to analyzing the impact of the updates, we grouped the updates observed at a single vantage point into events in the following two steps. Note that the goal of this grouping is not to accurately identify routing events, but to minimize the number of updates that potentially have the same impact on ASes in the network. To reduce the processing time, we choose a larger time value than those chosen in the literature [92, 65, 91] as grouping threshold. In practice, such time value can be adjusted based on need.

**S1.** If two updates, $upd_1$ and $upd_2$, are for the same prefix, and the timestamps of the two are within a certain threshold (240 seconds), then we group them together by removing $upd_1$.

**S2.** If two updates, $upd_1$ and $upd_2$, are for different prefixes, but (i) they occur within a certain time threshold (10 seconds), (ii) they share the same AS path, and (iii) their corresponding AS paths in the routing table are also identical, then we group them into an event. The reason we use a small number 10 is that we assume the updates for those prefixes are most likely propagated in the same BGP message.

After the grouping, we analyzed the impact of updates by using the events obtained above,

Table 5.3: AS numbers and their tiers

| Tier | Number of ASes | AS Numbers |
|---|---|---|
| 1 | 6 | 1239, 2914, 3356, 3549, 701, 7018 |
| 2 | 26 | 1221, 1668, 2152, 2497, 286, 293, 2905, 3257, 3303, 3557, 5413, 5511, 6453, 6539, 6762, 6939, 7660, 852, 11537, 11608, 11686, 12956, 13237, 14107, 16150, 22388 |
| 5 | 4 | 3277, 3130, 10876, 12682 |

and derived the affected AS set for each event. The analysis method was described in Section 5.2. The impact is measured by the number of ASes affected by each event. We used three measures: average, maximum, and minimum number of affected ASes.

The results are summarized as follows. Table 5.4 presents the results based on the tier classification of the vantage points. The events observed at the top tiers are, in general, shown to have larger impact on the network. This is expected, as top-tier ASes have a large number of customer ASes and routing events are more likely to come from customer ASes.

Table 5.5 presents the results based on the information of which type of neighbor (a provider, peer, customer, or sibling) an update is received from. In some cases an update is generated by the local vantage point itself, which is labeled with "Local" in the table. Local updates are, in general, shown to have the greatest impact, since they affect all the ASes reachable from the local AS. An event received from a peer or provider affects only single-homed customers of the local AS, thus resulting in smaller impact. In contrast, an event received from a customer or sibling has larger impact, since it also affects ASes that are not customers of the local AS.

Table 5.6 combines the results of the first two tables, and lists the results based on both AS tiers and neighbor types. The results are more detailed, and confirm the observation from the first two tables. For example, since tier-5 nodes have only providers and peers, any events received from providers or peers only affect the AS itself.

Table 5.4: Impact based on tiers of vantage points

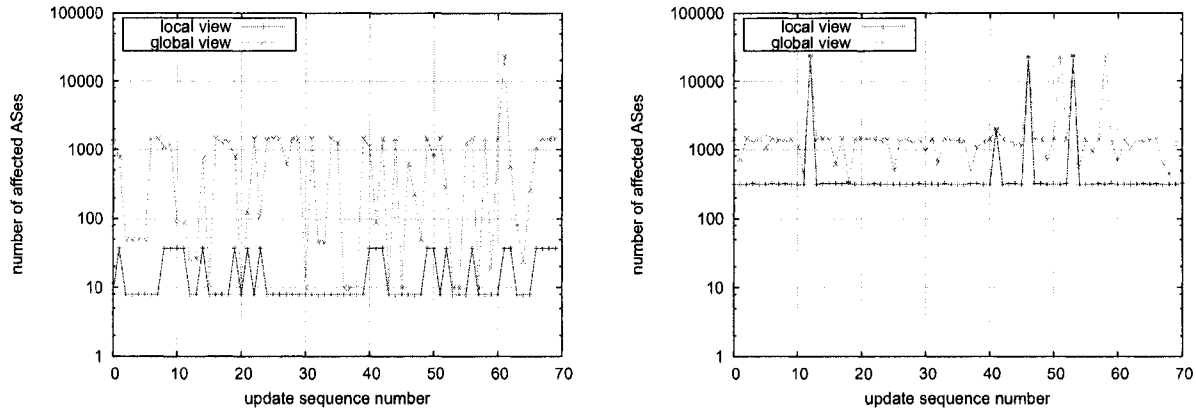
| Tier | Average | Maximum | Minimum |
|---|---|---|---|
| 1 | 800 | 23355 | 46 |
| 2 | 200 | 23355 | 1 |
| 5 | 1 | 1 | 1 |

Table 5.5: Impact based on neighbor type

| Neighbor Type | Average | Maximum | Minimum |
|---|---|---|---|
| Local | 23180 | 23355 | 22994 |
| Peer | 101 | 335 | 1 |
| Provider | 8 | 32 | 1 |
| Customer | 3808 | 23355 | 1830 |
| Sibling | 3252 | 23355 | 1994 |

Table 5.6: Impact based on both tier and neighbor types

| Tier | Neighbor Type | Average | Maximum | Minimum |
|---|---|---|---|---|
| 1 | Local | 23042 | 23275 | 22994 |
| 1 | Peer | 172 | 335 | 46 |
| 1 | Provider | 0 | 0 | 0 |
| 1 | Customer | 3562 | 23355 | 1830 |
| 1 | Sibling | 6011 | 23355 | 3296 |
| 2 | Local | 23339 | 23355 | 23324 |
| 2 | Peer | 14 | 32 | 2 |
| 2 | Provider | 10 | 32 | 1 |
| 2 | Customer | 4481 | 23355 | 1987 |
| 2 | Sibling | 2606 | 23355 | 1994 |
| 5 | Local | 0 | 0 | 0 |
| 5 | Peer | 1 | 1 | 1 |
| 5 | Provider | 1 | 1 | 1 |
| 5 | Customer | 0 | 0 | 0 |
| 5 | Sibling | 0 | 0 | 0 |




(a) stub AS (b) tier-1 AS

Figure 5.4: Comparison of local and global views: tier-1 AS has smaller discrepancy, but the trend can be predicted accurately for both.

### 5.2.4 Correlation of Events Collected from Multiple Vantage Points

To obtain a global view of the impact of a routing change, we aggregated the impact results from all the vantage points available and correlated the events triggered by the same routing change. For example, if two events observed at ASes A and B are triggered by the same routing change, and if the sets of impacted ASes at A and B are $S_A$ and $S_B$, respectively, then the combined set of impacted ASes should be $S_A \cup S_B$. If events at A and B are both received from customer ASes, or both from peer/provider ASes, then $S_A$ and $S_B$ should contain different ASes. If A receives the update from a customer, while B receives the update from a peer/provider, then potentially $S_B$ could be a subset of $S_A$. If A receives an update from B, then $S_A$ is definitely a subset of $S_B$.

The results show that for lower-tier ASes or ASes near the edge of the Internet, the difference between local and global views is significant, as shown in Fig. 5.4 (a). For tier-1 ASes, the difference is generally smaller, as shown in Fig. 5.4 (b). The large discrepancy for stub ASes' local views and global views is largely accounted for by our conservative estimation. However, it is important to note that the trend of larger impact in local views usually is accurately reflected in similarly scaled-up larger impact in the global views as explained below. Such trend prediction is useful to help guide BGP route selection to reduce impact on traffic.

We studied how the results from local views correlate with those from global views. We used an estimated average impact value to distinguish the updates with larger impact from those with

| Source IP | Destination IP | Protocol Type | Source AS | Destination AS | Octets | Packets |
|---|---|---|---|---|---|---|

Figure 5.5: Netflow record format

smaller impact. This estimated average is computed as the minimum of two values to help exclude outliers: half of the maximum number of impact ASes, and 10 times the mininum number of impact ASes. The results indicate that the correlation is very strong—when the impact of an update is above the estimate from the local view, more than 85% of the cases the impact is above estimated average from the global view as well. Thus, impact prediction using local view alone provides accurate estimation of global impact without acquiring additional external data.

## 5.3 Traffic-Level Analysis

Previously, we have described a simple and accurate algorithm for predicting the impact of a routing update in terms of the number of affected ASes, whose forwarding decisions are impacted by the routing update. To make the impact metric more complete, we now analyze another aspect of the impact based on limited local traffic information to infer the amount of global traffic affected due to the observed routing update.

### 5.3.1 Analysis of Local ISP's Traffic Data

Complementing the AS-level analysis in the previous section, we also analyzed the impact of BGP routing changes on traffic by using traffic data obtained from a regional ISP serving various universities and organizations. *Netflow* [93] is a Cisco standard for collecting flow-based network traffic statistics from network devices. A typical Netflow entry is presented in Fig. 5.5. It includes the source and destination IPs, protocol type (*e.g.*TCP or UDP), source and destination AS numbers, and traffic volume of the flow in terms of byte and packet counts. Using the local ISP's Netflow data collected in 2006, we first classified the prefixes into local or ISP's own prefixes and external prefixes. For this, we identified a list of ASes that belong to local networks: the local ISP itself and its customer ASes. Then, from RouteViews' routing table data, we mapped prefixes to ASes. Prefixes belonging to the local ASes are classified as local prefixes; the others are external prefixes. Based on this classification, traffic is divided into three categories: external to local, local to external, and local to local.

Table 5.7: Top external prefixes: traffic source (bytes)

| Prefixes | ASes | Description |
|---|---|---|
| 72.14.202.0/23 | 15169 | Google |
| 68.142.64.0/18 | 22822 | Limelight Networks, Inc. |
| 130.199.0.0/16 | 43 | Brookhaven National Laboratory |
| 192.76.177.0/24 | 12 | New York University |
| 128.223.0.0/16 | 3582 | University of Oregon |
| 209.73.188.0/23 | 36088 | AltaVista Company/Yahoo |
| 64.236.0.0/16 | 1668 | AOL Transit Data Network |
| 208.49.80.0/22 | 18607 | UNS (UseNetServer) |
| 64.233.162.0/23 | 15169 | Google |
| 128.91.0.0/16 | 55 | University of Pennsylvania |

Then, for each of the three categories, we aggregated the traffic based on source and destination prefixes, respectively, to identify the top prefixes that generate and receive most traffic. We first sorted the source or destination prefixes based on their traffic volume (in terms of both byte and packet counts). Tables 5.7 and 5.8 list the top 10 external prefixes sending traffic to the local ASes, in terms of the number of bytes and packets, respectively. As can be seen, 8 of the 10 prefixes in the two tables are identical and most are university networks or Google, reflecting the type of the customers of the local ISP. Tables 5.9 and 5.10 list the top 10 external prefixes receiving traffic from local ASes, in terms of the number of bytes and packets, respectively. They are quite different from the last two lists. For example, in Table 5.9, 7 out of the 10 prefixes belong to local commercial ISPs. At the same time, the two lists are quite different, only 3 out of 10 overlap. Such traffic analysis provides a basis for understanding the amount of traffic a given routing change to an external destination may influence.

Next, from the local ISP's routing update files (for the same time period as Netflow data), we linked the number of updates to the traffic volume for each prefix to examine the correlation between the routing changes and traffic volume for various destination prefixes from the perspective of that local ISP network. The results depicted in Fig. 5.6 show that (i) the majority of traffic is between local and external networks; (ii) the prefixes with a higher traffic volume tend to have fewer routing changes confirming a previous study [94] performed more than five years ago.

Fig. 5.6(a) shows the traffic volume for the prefixes. To better illustrate the trend, we used

Table 5.8: Top external prefixes: traffic source (packets)

| Prefixes | ASes | Description |
|---|---|---|
| 72.14.202.0/23 | 15169 | Google |
| 68.142.64.0/18 | 22822 | Limelight Networks, Inc. |
| 218.85.128.0/17 | 4134 | XiaMen JiMei Sweet Potato Netbar |
| 130.199.0.0/16 | 43 | Brookhaven National Laboratory |
| 192.76.177.0/24 | 12 | New York University |
| 128.223.0.0/16 | 3582 | University of Oregon |
| 64.236.0.0/16 | 1668 | AOL Transit Data Network |
| 64.233.162.0/23 | 15169 | Google |
| 209.73.188.0/23 | 36088 | AltaVista Company/Yahoo |
| 69.25.156.0/23 | 32385 | VoEx, Inc. |

*average* values: we first sorted the prefixes according to their traffic volume (in number of bytes), and then, from every 100 data points we computed their average value. Thus, each point in the figure represents the average value of 100 data points. For Figs. 5.6(b)-(d), in addition to the average values, we also computed the maximum and minimum values for each 100 data points. In all four of them, the prefixes are in the same order for ease of comparison. Fig. 5.6(b) shows the trend for the number of updates. In Figs. 5.6(c) and (d), updates are further classified into route announcements and withdrawals. The prefixes with a higher traffic volume tend to have fewer routing changes. The trend for the number of withdrawals (in Fig. 5.6(d)) is more evident. Since withdrawals have more impact on traffic than announcement, understanding its trend is more important.

We also analyzed the correlation of routing updates with the number of packets being sent and received based on individual prefixes. The results follow the same trend as in Fig. 5.6 based on byte count.

### 5.3.2 Analysis of Top Internet Sites

To examine the impact of routing changes on Internet traffic from the perspective of the global Internet, we also analyzed the impact on the top Internet sites (ranked by traffic volume) in the US and around the globe. We use the following methodology to accomplish the task. We (i) obtained

Table 5.9: Top external prefixes: traffic destination, bytes

| Prefixes | ASes | Description |
|---|---|---|
| 69.208.0.0/12 | 7132 | SBC Internet Services |
| 68.40.0.0/18 | 33668 | Comcast Cable Communications |
| 68.40.192.0/18 | 33668 | Comcast Cable Communications |
| 68.42.64.0/18 | 33668 | Comcast Cable Communications |
| 66.249.64.0/19 | 15169 | Google |
| 68.40.128.0/18 | 33668 | Comcast Cable Communications |
| 129.79.0.0/16 | 87 | Indiana University |
| 70.224.0.0/11 | 7132 | SBC Internet Services |
| 131.225.0.0/16 | 3152 | Fermi National Accelerator Lab |
| 68.72.0.0/13 | 7132 | SBC Internet Services |

the top US 100 and top global 500 sites ranked by Alexa [95]; and (ii) by using the DNS server, we translated domain names into IP addresses. To get more IP addresses, we also prepended common domain name prefixes (such as "mail.", "video.", and "news.") to the obtained domain names; (iii) from the individual IP addresses, we obtained the matching IP prefixes from RouteViews' routing table data, which are the smallest subnets covering the IP addresses; (iv) we also translated the IP prefixes to the corresponding AS numbers using the RouteViews RIB data. A route entry contains the destination IP prefix and its AS path. The last AS number of an AS path is used as the AS number for the corresponding prefix; (v) finally, we used a month (April 2007) of routing update data from RouteViews, and checked how many of the updates are for the top prefixes and top ASes.

The results are presented in Table 5.11. Only about 0.045% of updates are shown to be for the top global 100 prefixes, while about 1.70% are for the top global 100 ASes. This clearly illustrates that from a global Internet perspective, popular destination prefixes usually experience fewer routing changes. To put those numbers into perspective, in the table we also listed a group of reference prefixes and reference ASes. The reference prefixes are obtained as follows: using the same routing update data from RouteViews, we first ranked the prefixes according to the number of routing changes associated with them. We then picked the prefix that ranked in the middle and chose a number of consecutive prefixes starting from it. The number equals the corresponding top prefix category. For example, to compare with the top 100 global sites that covers 585 prefixes, we used 585 prefixes. Note that since the RouteViews data do not distinguish US prefixes from

Table 5.10: Top external prefixes: traffic destination, packets

| Prefixes | ASes | Description |
|---|---|---|
| 161.53.0.0/16 | 2108 | Croatian Academic and Research Network (CARNet) |
| 72.14.202.0/23 | 15169 | Google |
| 68.142.64.0/18 | 22822 | Limelight Networks, Inc. |
| 152.63.0.0/16 | 701 | UUNET |
| 69.25.156.0/23 | 32385 | VoEx, Inc. |
| 69.208.0.0/12 | 7132 | SBC Internet Services |
| 192.76.177.0/24 | 12 | New York University |
| 216.239.36.0/23 | 15169 | Google |
| 66.249.64.0/19 | 15169 | Google |
| 68.40.0.0/18 | 33668 | Comcast Cable Communications |

Table 5.11: Impact on top Internet sites

| | Top Prefixes (%) | Top ASes (%) | Reference Prefixes (%) | Reference ASes (%) |
|---|---|---|---|---|
| Top US 100 | 0.051 | 1.08 | - | - |
| Top Global 100 | 0.045 | 1.70 | 0.086% | 4.74% |
| Top Global 200 | 0.078 | 3.22 | 0.136% | 6.12% |
| Top Global 300 | 0.129 | 5.29 | 0.169% | 8.37% |
| Top Global 400 | 0.172 | 7.74 | 0.209% | 9.61% |
| Top Global 500 | 0.212 | 10.21 | 0.241% | 10.22% |

foreign prefixes, we did not have a reference prefix and AS lists for top U.S. 100 sites. As can be seen, the top prefixes or ASes have fewer routing changes than the reference lists. However, if such routing changes occur, they will affect a significantly larger amount of traffic. As described later, we devise an intelligent routing selection improvement to reduce such routing-induced traffic disruptions.

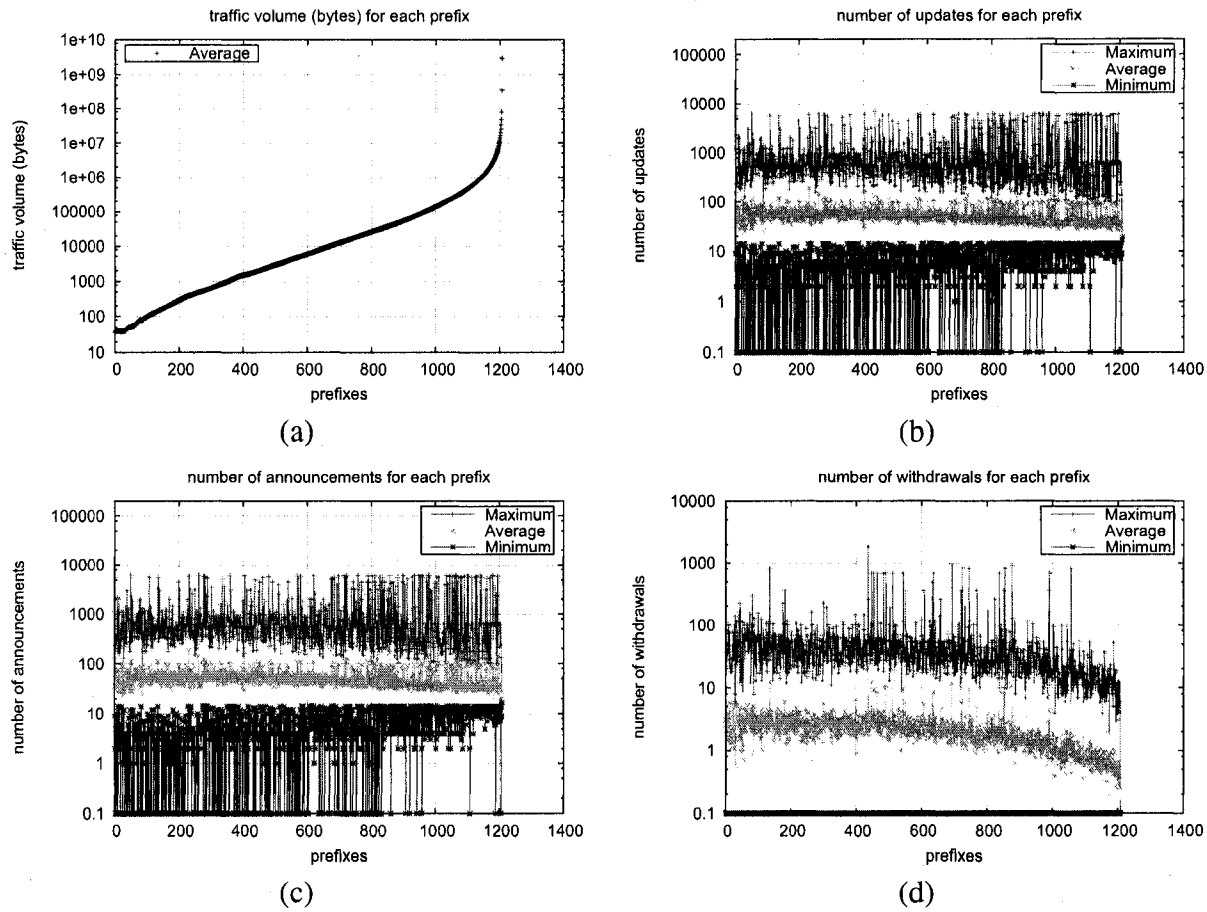



Figure 5.6: Relationship between traffic volume (bytes) and number of updates: external to local, source-based comparison.

### 5.3.3 Routing Stability of ISPs

One metric to rate different ISPs is based on the stability of routes carried by the ISP. If an ISP continuously announce many routing changes, it means that the ISP does not have stable routes to reach the corresponding destinations. To compare the routing stability of various ISPs, we used the routing update files collected by both RouteViews and RIPE for the month of April in 2007. The vantage points covers 71 ASes located in the US, Europe, and Asia. We describe below our simple method to compare routing stability of large ISPs based on observed routing changes. Note that if a routing change is due to failure close to the origin prefix, all networks will be impacted. Thus, counting updates is a simple and effective way to compare ISP's routing stabilities.

(i) First, we checked the RIB files for those ASes and removed the ASes that have very small

RIB files (implying that they contained default routes rather than a full BGP table). For the remaining 63 ASes, we divided each file into multiple sub-files, one for each AS.

(ii) Then, we removed the extraordinarily large sub-files, which may be triggered by the session resets between the vantage points and the collecting routers of RouteViews and RIPE. For this, we first checked how many routing updates the files contain. If a file contains too many routing updates (more than half of a full BGP table size), we checked how many prefixes are affected by those routing updates. If the number is greater than a threshold, a half of the routing table size for that AS, then we checked the files collected immediately before and after the current file. We combined one of the files with the current file, respectively. If for the combined file, the affected prefixes is close to the corresponding routing table size, then there is a session reset between the AS and RouteViews/RIPE router. We remove the number of routing changes that equals the routing table size from the final count for that AS.

Note that unlike the previous work [96] on identifying BGP table transfers, we did not try to infer exactly when the reset starts. We just needed to confirm that there exists a reset around the time the current update file was collected and subtracted that amount of routing changes from the final calculation. We also assumed that a session reset occurs within 30 minutes, and it span over at most two consecutive update files. This is verified from previous observations.

(iii) Next, all of the files for the same AS will be combined together to count the total number of updates for that AS. Note that some ASes have multiple peering sessions with RouteVeiws/RIPE routers. We calculated the average number of updates for the ASes in this case.

(iv) We now compare the number of updates for different ASes, and plot them in Fig. 5.7. The stability of different ASes is found to vary significantly. Most ASes have 4000K—10,000K routing changes, though. The most stable and least stable ASes are shown in Table 5.13.

For comparison, we also removed all large files identified in Step (ii) and plotted a second curve. The goal was to see how much the results were affected by some large bursts of routing updates. The two curves are found to be similar. The results also show that for the ASes with many routing changes, they are continuously unstable—the instability is not caused by those large bursts of routing changes. Such an analysis is useful for making provider ISP selection decisions, as any network would prefer to obtain transit services from ISPs providing stable routing to most destinations.

Table 5.12: Tiers of vantage points

| Tiers | Number of ASes |
|---|---|
| 1 | 8 |
| 2 | 44 |
| 3 | 5 |
| 5 | 6 |

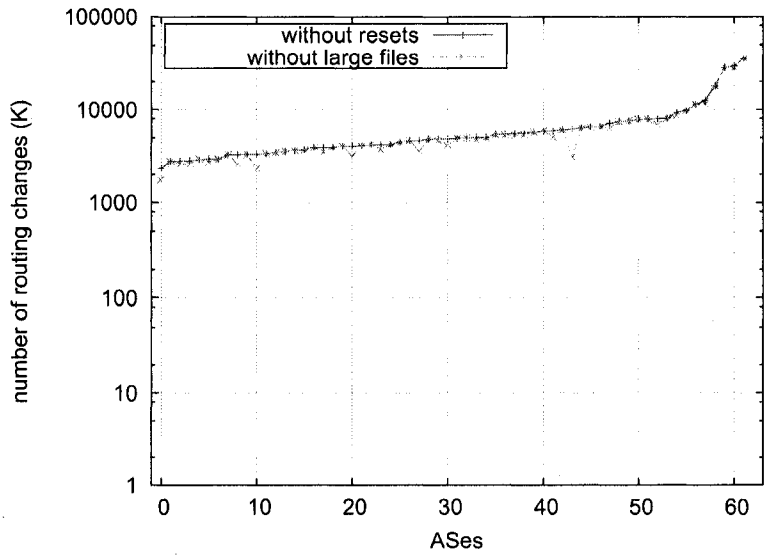



Figure 5.7: AS stability comparison

### 5.3.4 Enhanced Routing-Decision Process

Based on the above analysis, some routing changes may significantly affect application performance, so we designed an enhanced version of the BGP routing decision process to take into account of the impact of a routing change. The basic idea is that if the difference between the existing route and a new route is not significant, and if the affected traffic volume is large, then the routing change will not be adopted immediately. The key is that the existing route should still be valid after the new route is received.

To evaluate the effectiveness of this idea, we examined the local ISP's data and analyzed the updates for the top 500 prefixes with a high traffic volume. If the ISP receives a better route (either with a higher local preference number or a shorter AS path length) that replaces the current route, we assumed that the replaced route is still valid. Then the algorithm delays the adoption of the

new route for 12 hours (This is to simulate the case that the adoption is delayed until midnight when traffic volume tends to turn low, so that the impact on traffic is minimized). The results show that most of the updates contain duplicate routes that are identical to the previous ones (or routes with the same AS path but different attributes). For the updates that do contain different AS paths than the previous routes, about 30% of routing changes for the top 10 prefixes can be saved if the changes are not adopted immediately. For most of the time, within two hours those changes were replaced by new changes. For top 500 prefixes, the average savings is 14%. The details are shown in Table 5.14.

We also checked how many prefixes virtually have no routing changes (or all of the routing changes are identical). As shown in Table 5.14, more than 30% of the top prefixes have no routing changes, as their respective providers are usually well managed. The maximum number of routing changes for a prefix is also shown. The top prefixes tend to have few changes; however, when they do experience changes, the impacted traffic is more significant and thus can benefit from a more traffic disruption aware route selection scheme.

## 5.4 Related Work

Our work is motivated by several past measurement studies on BGP dynamics [65, 97] and the observation that most studies do not effectively quantify how BGP routing change affects traffic at the global Internet scale.

A recent study [98] of BGP dynamics has shown that as the Internet is becoming larger, BGP routing changes also become more intense. Our work verifies and builds on a previous study by Rexford *et al.* [94] which shows that most BGP routing changes stem from a small number of unpopular destinations; for the popular destinations that generate large volume of traffic, their routes are very stable. However, their observation is from a single (although large) AS's point of view, which limits the significance of their results. Moreover their study focused on only the first question mentioned in our introduction without further understanding how to reduce potentially disruptive impact on traffic.

Kushman *et al.* [86] recently analyzed the performance of VoIP applications and found that BGP's slow convergence has significant negative impact on the quality of cross-domain VoIP calls. Specifically, BGP's slow convergence also prevents users from reestablishing the broken VoIP calls quickly. This motivates our work on traffic disruption aware route selection schemes.

Our work also relates to various recent proposals on improving BGP's route decision process, such as PGBGP [99] by more cautiously adopting anomalous routes. Our proposal on modifying routing decisions can be incrementally deployed using routing control platform such as RCP [66] currently in use by large ISPs.

## 5.5 Concluding Remarks

In this chapter we performed the first comprehensive study on the impact of BGP routing changes on network traffic at the global scale. We tackled the problem in two steps: first, we analyzed how many ASes are affected by a given routing change observed at a vantage point. By using RouteViews routing data, we showed that routing changes observed at top-tier ASes tend to have impact on more ASes. Also, routing changes received from customer or local networks tend to have impact on more ASes. We then analyzed the impact at traffic level. By using a local regional ISP's traffic data, we were able to show that the top prefixes generating or receiving more traffic tend to have fewer routing changes. We also examined the impact of routing changes on some top Internet sites, and found that the number of routing changes for those sites is small. Finally, we propose an improved route selection scheme to reduce routing induced disruption on traffic.

Table 5.13: The most and least stable ASes based on routing stabilities.

| AS Number | AS degree | Num of Customers | Num of Providers | Num of Peers | Tier |
|---|---|---|---|---|---|
| Most stable ASes | | | | | |
| 3303 (Swisscom) | 599 | 155 | 15 | 432 | 2 |
| 6079 (RCN Corporation) | 142 | 53 | 13 | 76 | 2 |
| 15444 (Netservices, UK) | 251 | 5 | 21 | 225 | 2 |
| 7660 (APAN, JP) | 37 | 22 | 5 | 10 | 2 |
| 2018 (TENET) | 5 | 0 | 4 | 1 | 5 |
| Least stable ASes | | | | | |
| 19151 (WV FIBER LLC) | 366 | 81 | 5 | 280 | 2 |
| 3549 (Global Crossing, Ltd.) | 798 | 742 | 0 | 56 | 1 |
| 3741 (Internet Solutions, South Africa) | 75 | 26 | 9 | 40 | 2 |
| 6730 (SUNRISE, Switzerland) | 603 | 172 | 31 | 400 | 2 |
| 12956 (Telefonica, Spain) | 152 | 48 | 20 | 84 | 2 |

Table 5.14: Statistics of updates for top prefixes (ranked by traffic volume in bytes)

| | Number of prefixes with no changes | Max. number of changes per prefix | Total number of routing changes | Total number of saved changes | Saving ratio (%) |
|---|---|---|---|---|---|
| Top 10 | 8 (40%) | 20 | 136 | 42 | 31 |
| Top 20 | 14 (35%) | 20 | 246 | 54 | 22 |
| Top 100 | 70 (35%) | 46 | 2046 | 256 | 12.5 |
| Top 500 | 327 (32.7%) | 3289 | 21857 | 3118 | 14 |

# CHAPTER 6

# Conclusions and Future Work

## 6.1 Main Contributions

Emerging real-time, multimedia Internet applications such as voice-over-IP (VoIP), IPTV, and video conferencing have attracted millions of users. Typical examples include Vonage and Skype's VoIP systems and Cisco's TelePresence. These applications require the network to provide better QoS support. In this dissertation, we tackled the problem of providing QoS in the Internet by exploring two main functions of data networks: packet scheduling and routing. We studied the performance of aggregate scheduling by both analysis and simulation, and proposed enhancements to the current BGP routing protocol to reduce both its convergence time and the number of routing updates. We also analyzed the impact of BGP routing changes on application traffic. The main contributions of our work are summarized in the following paragraphs.

We first studied the performance of aggregation scheduling in Chapters 2 and 3. Aggregate scheduling combines the scalability of DiffServ and service guarantees of IntServ. We first studied a vanilla version of aggregate scheduling in Chapter 2. By using the GR algorithms to schedule traffic aggregates, we showed not only the existence of e2e delay bounds, but also the fact that in many cases the bounds are smaller than those of per-flow scheduling. In addition, by using in-depth simulation we not only confirmed the analytical results, but also demonstrated the advantages of aggregate scheduling. Aggregate scheduling is very robust and can exploit statistical multiplexing gains. It performs better than per-flow scheduling in most cases.

In Chapter 3, we extended the previous work and proposed a coordinated version of aggregation scheduling — Coordinated Aggregate Scheduling (CAS), where multiple scheduling nodes along a path calculate the scheduling deadline at each node collectively. The computation of the deadline

of a packet at each intermediate node is coordinated between the node itself and its upstream nodes. This new algorithm makes the delay of a flow independent of the burstiness of other flows in the same aggregate, yielding smaller e2e delays. This performance improvement is also confirmed by our simulation results.

To provide better QoS, intelligent packet scheduling alone is not sufficient. The routing system has to be stable and be able to converge to a new path quickly when network changes occur. In Chapters 4 and 5, we studied the routing aspect of data networks and proposed ways of enhancing the performance of the BGP routing protocol. In Chapter 4, we proposed a novel method of differentiated update processing. BGP updates are classified based on whether their routes are used in the forwarding tables of the receiving routers for related destination prefixes. High-priority updates will be processed sooner while low-priority ones will be delayed. This new algorithm is shown to be able to reduce both the BGP convergence time and the amount of routing updates. We also proposed a difference-based route selection algorithm, which can significantly reduce the amount of unnecessary routing exchange and speed up BGP convergence during network failures.

In Chapter 5, we studied the impact of BGP routing changes on application traffic at both AS level and traffic level. At AS level, we showed that routing changes observed at top-tier ASes tend to affect more ASes. Also, routing changes received from a customer network or local network tend to have impact on more ASes. At traffic level, we were able to show that the top prefixes generating/receiving more traffic volume tend to have fewer routing changes. We also examined the impact of routing changes on some top Internet sites. The results showed that the number of routing changes for those sites is small.

## 6.2 Future Work

Despite the recent progress in understanding BGP routing dynamics, there still remain a number of important issues that warrant further research. We discuss below a few of them.

### Further evaluation of the enhanced routing-decision process

In Chapter 5 we proposed a novel idea of delaying the selection of certain routes to minimize the number of routing changes and traffic shifts. The effectiveness of this enhanced route-decision process needs to be examined more thoroughly. For example, a more detailed algorithm is needed to specify under what conditions the delaying mechanism takes action. Simulation and/or experi-

ments are also needed to evaluate the performance of such a algorithm.

## More intelligent AS relationship inference algorithm

Our work in Chapter 5 relies on the accuracy of the AS relationship data. Although many inference algorithms have been proposed in literature [89, 88, 82, 90], the relationships inferred by them are, in general, based on some simplistic heuristics such as "valley-free" rule and degree-based rule (an AS with a larger degree of connectivity is more likely to be a provider of its neighboring AS with a smaller degree, not vice versa). Therefore, the inferred relationship does not accurately reflect the complicated business relationship between ASes in the real world. To make matters worse, the data sources for the inference algorithms are rather limited. Public BGP data repositories such as RouteViews and RIPE provide detailed BGP RIB and update information from dozens of BGP vantage points. However, compared to the scale of the global Internet, their coverage is still very limited. For this reason, a lot of backup routes and lower-tier peer-peer links are invisible. Later algorithms such as those in [89, 90] tried to solve this problem by taking into account other information sources such as the Internet registry data. However, (i) the registry data are often out-dated and (ii) the format of the data is not uniform, so it is difficult to extract useful information from them automatically and efficiently. As a result, the relationships inferred by different algorithms are not consistent with each other. Moreover, there is not any easy way to verify which result is correct. Therefore, how to design an accurate AS relationship inference algorithm remains an open question.

## Automatic BGP route aggregation

Driven by the expansion of the Internet and the prevalence of multi-homing, the size of the BGP table in Internet core routers is increasing very fast, which, in turn, increases the processing time and overhead of user traffic. At the same time, a larger routing table also demands more memory in routers. The popularity of Virtual Private Networks (VPNs), where routers need to maintain a separate routing table for each supported VPN, further exacerbates the problem. To solve this problem, the current BGP routers use CIDR (Classless Inter-Domain Routing), which aggregates multiple BGP routes into a single route, but the aggregation requires manual configuration, which is tedious and prone to error. Recently, several algorithms [100, 101, 102] have been proposed to help reduce the BGP table size. However, they either require setting up and maintaining a new type

of routing entity such as policy atom and virtual prefix, or temporarily discard (and later retrieve) some alternative routes to save memory space. Therefore, these algorithms are complicated and difficult to deploy. One idea that merits further exploration is to *automatically* aggregate multiple related BGP routing entries into a single entry without any human intervention. This is compatible with the current BGP's CIDR and easy to deploy. Our preliminary test showed that more than 25% of BGP routes can be easily aggregated by using this scheme.

# APPENDICES

# APPENDIX A

# End-to-End Delay Bounds for Traffic Aggregates under Guaranteed-Rate Scheduling Algorithms

## A.1 Proof of Corollary 2.2.1

*Proof.* Similar to Theorem 2.2.1, from Lemma 2.2.1 we have

$$GRC^{i+1}(p_A^{j'}) \leq GRC^i(p_f^j) + \frac{\sum_{k\neq f}\tilde{\sigma}_k + \ell_f^{max}}{R} + \alpha^i.$$

Replacing $\sum_{k\neq f}\tilde{\sigma}_k$ by (2.27), we obtain

$$GRC^{i+1}(p_A^{j'}) \leq GRC^i(p_f^j) + \frac{\sum_{k\neq f}\sigma_k + L_{max}^i + 2\ell_f^{max} + 2\ell_A^{max}}{R} + \alpha^i. \tag{A.1}$$

Thus for the two-level hierarchical H-WF$^2$Q aggregator $S_i$, it is a fair aggregator with aggregation constant $\gamma^i = \frac{\sum_{k\neq f}\sigma_k + L_{max}^i + 2\ell_f^{max} + 2l_A^{max}}{R}$.

Then, similar to the proof of Theorem 2.2.2, the e2e delay $d_f^j$ satisfies

$$d_f^j \leq \frac{\sigma_f}{r_f} + \gamma^1 + (K-3)\frac{\ell_A^{max}}{R} + \frac{\ell_f^{max}}{r_f} + \alpha_H^1 + \sum_{i=2}^{K}\alpha^i. \tag{A.2}$$

Also, since $S_1$ is a hierarchical aggregator, from Fact 2.2.1 we know that its scheduling constant is larger than that of the stand-alone server:

$$\alpha_H^1 \leq \alpha^1 + \frac{\ell_A^{max} + (L_{max}^1 - \ell_A^{max})\frac{R}{C^1}}{R} \leq \alpha^1 + \frac{L_{max}^1}{R}. \tag{A.3}$$

Rearranging the terms in Eq. (A.2), we obtain Eq. (2.28). □

## A.2 Proof of Theorem 2.2.3

*Proof.* For the convenience of description, we regard the other ($N$-1) flows in aggregate flow $A$ as a virtual flow $\bar{f}$, and prove the lemma for any time $t$ in flow $\bar{f}$'s backlogged period.

Consider the two-level hierarchical H-PFQ scheduler $S_i$. At the upper-level scheduler, aggregate flow $A$ is treated as a single flow conforming to the token bucket model $(\sigma_A, \rho_A)$, where $\sigma_A = \sum_{k=1}^{N} \sigma_k$ and $\rho_A = \sum_{k=1}^{N} \rho_k$; at the lower-level variable-rate scheduler, $N$ flows are aggregated into $A$. *As pointed out in [52], by defining virtual times in unit of bits instead of in unit of seconds, many properties of constant-rate packet schedulers still hold for variable-rate packet schedulers.*

Consider any backlogged period $(\tau, t]$ of flow $\bar{f}$ at $S_i$. For any flow $k$ and time $t$, from Eqs. (2.24) and (2.25) we obtain:

$$W^i_{k,GPS}(0,t) - W^i_k(0,t) \leq \lambda^i_k,$$
$$W^i_k(0,t) - W^i_{k,GPS}(0,t) \leq \delta^i_k.$$

Thus, for flow $f$ at the lower-level scheduler $S_{i,l}$,

$$\begin{aligned} W^i_f(\tau,t) &= W^i_f(0,t) - W^i_f(0,\tau) \\ &\leq [W^i_{f,GPS}(0,t) - W^i_{f,GPS}(0,\tau)] + \lambda^i_f + \delta^i_f \\ &= W^i_{f,GPS}(\tau,t) + \lambda^i_f + \delta^i_f. \end{aligned} \tag{A.4}$$

Let $W^i_l(\tau,t)$ be the total service provided by $S_{i,l}$, a variable-rate scheduler, during $(\tau,t]$. Since flow $\bar{f}$ is backlogged during this period, $f$ at most gets its reserved share at the corresponding GPS server at $S_{i,l}$, and $W^i_l(\tau,t)$ reaches $S_{i,l}$'s capacity during the period. Thus,

$$W^i_{f,GPS}(\tau,t) \leq \frac{r_f}{R} \cdot W^i_l(\tau,t)$$

For $t = t_k$, the time when $k^{th}$ ($k \geq 1$) packet in aggregate $A$ is sent out, we have

$$\begin{aligned} W^i_l(\tau,t_k) &= W^i_A(\tau,t_k), \qquad k \geq 1 \\ \Longrightarrow W^i_{f,GPS}(\tau,t_k) &\leq \frac{r_f}{R} \cdot W^i_A(\tau,t_k), \quad k \geq 1. \end{aligned}$$

For $t_k \leq t \leq t_{k+1}$ ($k \geq 0$ and assume $t_0 = \tau$), since $W^i_{f,GPS}(\tau,t) = W^i_{f,GPS}(\tau,t_k)$ and $W^i_A(\tau,t) =$

$W_A^i(\tau, t_k)$, we obtain,

$$W_{f,GPS}^i(\tau,t) \leq \frac{r_f}{R} \cdot W_A^i(\tau,t), \quad t_k \leq t \leq t_{k+1}, k \geq 0.$$

Therefore,

$$W_{f,GPS}^i(\tau,t) \leq \frac{r_f}{R} \cdot W_A^i(\tau,t), \quad t \geq \tau. \tag{A.5}$$

Similarly, for aggregate $A$ at the upper-level scheduler, we have

$$\begin{aligned} W_A^i(\tau,t) &= W_A^i(0,t) - W_A^i(0,\tau) \\ &\geq [W_{A,GPS}^i(0,t) - W_{A,GPS}^i(0,\tau)] - \lambda_A^i - \delta_A^i \\ &= W_{A,GPS}^i(\tau,t) - \lambda_A^i - \delta_A^i. \end{aligned} \tag{A.6}$$

From Eqs. (A.4), (A.5), and (A.6) we obtain

$$\begin{aligned} W_{\bar{f}}^i(\tau,t) &= W_A^i(\tau,t) - W_f^i(\tau,t) \\ &\geq [W_A^i(\tau,t) - \frac{r_f}{R} W_A^i(\tau,t)] - [\lambda_f^i + \delta_f^i] \\ &\geq (1 - \frac{r_f}{R})[W_{A,GPS}^i(\tau,t) - \lambda_A^i - \delta_A^i] - [\lambda_f^i + \delta_f^i]. \end{aligned} \tag{A.7}$$

Since $\bar{f}$ is backlogged during $(\tau, t]$, so must $A$ during this period, and hence

$$W_{A,GPS}^i(\tau,t) \geq R \cdot (t - \tau). \tag{A.8}$$

Since $\tau$ is the start time of the backlogged period, the backlog of $\bar{f}$ at $\tau$ is 0. Thus the backlog of $\bar{f}$ at time $t$, $Q_{\bar{f}}^i(t)$, is

$$\begin{aligned} Q_{\bar{f}}^i(t) &= \sum_{k \neq f} A_k(\tau,t) - W_{\bar{f}}^i(\tau,t) \\ &\leq \sum_{k \neq f} [\sigma_k + \rho_k(\tau,t)] - (1 - \frac{r_f}{R}) W_{A,GPS}^i(\tau,t) + (1 - \frac{r_f}{R})[\lambda_A^i + \delta_A^i] + [\lambda_f^i + \delta_f^i] \\ &\leq [\sum_{k \neq f} \sigma_k + (R - r_f)(t - \tau)] - (1 - \frac{r_f}{R}) \cdot R \cdot (t - \tau) + (1 - \frac{r_f}{R})[\lambda_A^i + \delta_A^i] + [\lambda_f^i + \delta_f^i] \\ &\leq \sum_{k \neq f} \sigma_k + (1 - \frac{r_f}{R})[\lambda_A^i + \delta_A^i] + [\lambda_f^i + \delta_f^i]. \end{aligned} \tag{A.9}$$

Note that Eq. (A.9) holds for any time $t$. Similar to the proof of Theorem 3 in [36], we obtain the total burstiness of the ($N$-1) outgoing flows as:

$$\sum_{k\neq f}\tilde{\sigma}_k \leq \sum_{k\neq f}\sigma_k + [\lambda_f^i + \delta_f^i] + (1-\frac{r_f}{R})[\lambda_A^i + \delta_A^i].$$

□

## A.3 Proof of Theorem 2.3.1

*Proof.* Let $p_A^{j'}$ be the packet corresponding to packet $p_f^j$ of flow $f$. From Lemma 2.3.1 we have

$$GRC^{1,h}(p_A^{j'}) \leq GRC^{1,l}(p_f^j) + \alpha^{1,l}.$$

Also, since virtual server $S_{1,h}$ and $S_2, \cdots, S_{K-1}$ are GR servers for aggregate flow $A$, from Lemma 2.1.2 we have

$$\begin{aligned}
GRC^2(p_A^{j'}) &\leq GRC^{1,h}(p_A^{j'}) + \frac{\ell_A^{max}}{R} + \alpha^{1,h},\\
GRC^3(p_A^{j'}) &\leq GRC^2(p_A^{j'}) + \frac{\ell_A^{max}}{R} + \alpha^2,\\
&\vdots\\
GRC^{K-1}(p_A^{j'}) &\leq GRC^{K-2}(p_A^{j'}) + \frac{\ell_A^{max}}{R} + \alpha^{K-2},
\end{aligned}$$

where $\alpha^{1,h} = \alpha^1$. Adding them up, and with $p_A^{j'} = p_f^j$, we obtain

$$\begin{aligned}
GRC^{K-1}(p_f^j) &= GRC^{K-1}(p_A^{j'})\\
&\leq GRC^{1,l}(p_f^j) + (K-2)\frac{\ell_A^{max}}{R} + \alpha^{1,l} + \sum_{i=1}^{K-2}\alpha^i.\\
\Longrightarrow D^{K-1}(p_f^j) &\leq GRC^{K-2}(p_f^j) + \alpha^{K-1}\\
&\leq GRC^{1,l}(p_f^j) + (K-2)\frac{\ell_A^{max}}{R} + \alpha^{1,l} + \sum_{i=1}^{K-1}\alpha^i
\end{aligned}$$

Thus, from the definition of GR server, for flow $f$, the first ($K$-1) servers can be viewed as a virtual GR server with scheduling constant $\alpha^* = (K-2)\frac{\ell_A^{max}}{R} + \alpha^{1,l} + \sum_{i=1}^{K-1}\alpha^i$.

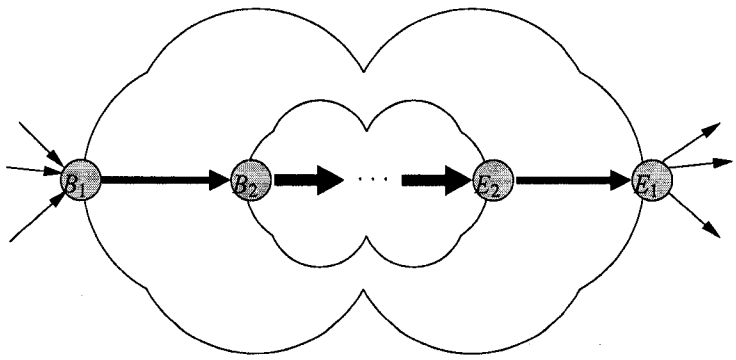



Figure A.1: Multiple recursive aggregation

Then for the last hop, since $S_K$ is also a GR server for flow $f$, we have

$$GRC^K(p_f^j) \leq GRC^{1,I}(p_f^j) + \alpha^* + \frac{\ell_f^{max}}{r_f}.$$

Thus we obtain

$$\begin{aligned} D^K(p_f^j) &\leq GRC^K(p_f^j) + \alpha^K \\ &\leq GRC^{1,I}(p_f^j) + (K-2)\frac{\ell_A^{max}}{R} + \alpha^{1,I} + \frac{\ell_f^{max}}{r_f} + \sum_{i=1}^{K} \alpha^i \\ \Longrightarrow d_f^j &\leq D^K(p_f^j) - A^1(p_f^j) \\ &\leq [GRC^{1,I}(p_f^j) - A^1(p_f^j)] + (K-2)\frac{\ell_A^{max}}{R} + \alpha^{1,I} + \frac{\ell_f^{max}}{r_f} + \sum_{i=1}^{K} \alpha^i. \end{aligned}$$

□

# A.4 Proof of Theorem 2.3.2

We first prove two lemmas, which consider two basic cases of multiple aggregations. One is pure recursive aggregation, as shown in Fig. A.1, and the other is pure sequential aggregation as shown in Fig. A.2.

**Lemma A.4.1 (Recursive Aggregation).** *Suppose flow $f$ takes $K$ hops of GR servers, and there are $M$ aggregators along the path. Suppose all the aggregators are rate-controlled fair aggregators. For each aggregator $B_i$, there is a corresponding deaggregator $E_i$ at a later hop, $1 \leq i \leq M$. $R_i$ is the GR for aggregate flow $A_i$ which starts from $B_i$ and ends at $E_i$. If $1 \leq B_1 < B_2 < \cdots B_{N-1} <$*

*$B_N < E_N < E_{N-1} \cdots < E_2 < E_1 \leq K$ (i.e., aggregation is done recursively, as shown in Fig. A.1), the e2e delay for packet $p_f^j$, $d_f^j$, satisfies:*

$$\begin{aligned} d_f^j \leq\ & [GRC^1(p_f^j) - A^1(p_f^j)] \\ & + [K - 1 - (E_1 - B_1 - 1)]\frac{\ell_f^{max}}{r_f} \\ & + \sum_{i=1}^{M-1} [(E_i - B_i - 1) - (E_{i+1} - B_{i+1} - 1)]\frac{\ell_{A_i}^{max}}{R_i} \\ & + (E_M - B_M - 1)\frac{\ell_{A_M}^{max}}{R_M} + \sum_{i=1}^{M} \frac{\ell_{A_i}^{max}}{R_i} + \sum_{n=1}^{K} \alpha^n. \end{aligned} \tag{A.10}$$

*Proof.* First, consider aggregate flow $A_M$. From Eq. (2.34) we know

$$D^{E_M}(p_{A_{M-1}}^j) \leq GRC^{B_M}(p_{A_{M-1}}^j) + (E_M - B_M)\frac{\ell_{A_M}^{max}}{R_M} + \frac{\ell_{A_{M-1}}^{max}}{R_{M-1}} + \sum_{n=B_M}^{E_M} \alpha^n.$$

Thus, for aggregate flow $A_{M-1}$, the set of nodes from $B_M$ to $E_M$ can be viewed as a virtual GR server with scheduling constant $\alpha^{B_M} = (E_M - B_M)\frac{\ell_{A_M}^{max}}{R_M} + \frac{\ell_{A_{M-1}}^{max}}{R_{M-1}} + \Sigma_{n=B_M}^{E_M} \alpha^n$.

Then, recursively for aggregate flow $A_i$ $(M-1 \geq i \geq 1)$, we have

$$D^{E_i}(p_{A_{i-1}}^j) \leq GRC^{B_i}(p_{A_{i-1}}^j) + [E_i - B_i - (E_{i+1} - B_{i+1})]\frac{\ell_{A_i}^{max}}{R_i} + \frac{\ell_{A_{i-1}}^{max}}{R_{i-1}} + \sum_{n=B_i}^{B_{i+1}-1} \alpha^n + \sum_{n=E_{i+1}+1}^{E_i} \alpha^n + \alpha^{B_{i+1}}.$$

Thus for aggregate flow $A_{i-1}$ $(M-1 \geq i \geq 1)$, the set of nodes from $B_i$ to $E_i$ can be viewed as a virtual GR server with scheduling constant

$$\alpha^{B_i} = [E_i - B_i - (E_{i+1} - B_{i+1})]\frac{\ell_{A_i}^{max}}{R_i} + \frac{\ell_{A_{i-1}}^{max}}{R_{i-1}} + \sum_{n=B_i}^{B_{i+1}-1} \alpha^n + \sum_{n=E_{i+1}+1}^{E_i} \alpha^n + \alpha^{B_{i+1}}.$$

Here we define $A_0 = f$ and $R_0 = r_f$.

Since $S_1$ and $S_K$ are GR servers for flow $f$, from Theorem 2.1.3, we have

$$D^K(p_f^j) \leq GRC^1(p_f^j) + [K - 1 - (E_1 - B_1)]\frac{\ell_f^{max}}{r_f} + \sum_{n=1}^{B_1-1} \alpha^n + \sum_{n=E_1+1}^{K} \alpha^n + \alpha^{B_1}.$$

Thus, by replacing $\alpha^{B_i}(1 \leq i \leq M)$ recursively, we obtain

$$
\begin{aligned}
D^K(p_f^j) \;\leq\; & GRC^1(p_f^j) + [K-1-(E_1-B_1)]\frac{\ell_f^{max}}{r_f} + \sum_{n=1}^{B_1-1}\alpha^n + \sum_{n=E_1+1}^{K}\alpha^n \\
& + [E_1-B_1-(E_2-B_2)]\frac{\ell_{A_1}^{max}}{R_1} + \frac{\ell_f^{max}}{r_f} + \sum_{n=B_1}^{B_2-1}\alpha^n + \sum_{n=E_2+1}^{E_1}\alpha^n \\
& \vdots \\
& + [E_{M-1}-B_{M-1}-(E_M-B_M)]\frac{\ell_{A_{M-1}}^{max}}{R_{M-1}} + \frac{\ell_{A_{M-2}}^{max}}{R_{M-2}} + \sum_{n=B_{M-1}}^{B_M-1}\alpha^n + \sum_{n=E_M+1}^{E_{M-1}}\alpha^n \\
& + [E_M-B_M]\frac{\ell_{A_M}^{max}}{R_M} + \frac{\ell_{A_{M-1}}^{max}}{R_{M-1}} + \sum_{n=B_M}^{E_M}\alpha^n.
\end{aligned}
$$

Rearranging the terms, we obtain

$$
\begin{aligned}
D^K(p_f^j) \;\leq\; & GRC^1(p_f^j) + [K-1-(E_1-B_1-1)]\frac{\ell_f^{max}}{r_f} \\
& + \sum_{i=1}^{M-1}[(E_i-B_i-1)-(E_{i+1}-B_{i+1}-1)]\frac{\ell_{A_i}^{max}}{R_i} + [E_M-B_M-1]\frac{\ell_{A_M}^{max}}{R_M} + \sum_{i=1}^{M}\frac{\ell_{A_i}^{max}}{R_i} + \sum_{n=1}^{K}\alpha^n \\
\Longrightarrow d_f^j \;\leq\; & [GRC^1(p_f^j) - A^1(p_f^j)] + [K-1-(E_1-B_1-1)]\frac{\ell_f^{max}}{r_f} \\
& + \sum_{i=1}^{M-1}[(E_i-B_i-1)-(E_{i+1}-B_{i+1}-1)]\frac{\ell_{A_i}^{max}}{R_i} + (E_M-B_M-1)\frac{\ell_{A_M}^{max}}{R_M} + \sum_{i=1}^{M}\frac{\ell_{A_i}^{max}}{R_i} + \sum_{n=1}^{K}\alpha^n.
\end{aligned}
$$

□

Since $B_i$ and $E_i$ are, respectively, the aggregator and deaggregator for aggregate flow $A_i$, they schedule packets of aggregate flow $A_{i-1}$. Thus, the number of hops/servers that schedule packets of $A_i$ is $(E_i-B_i-1)-(E_{i+1}-B_{i+1}-1)$ for $1 \leq i < M$, or $E_i-B_i-1$ for $i=M$. Thus, Eq. (A.10) can be rewritten as Eq. (2.36).

**Lemma A.4.2 (Sequential Aggregation).** *Suppose flow $f$ takes $K$ hops of GR servers, and there are $M$ aggregators along the path. Suppose all the aggregators are rate-controlled fair aggregators. For each aggregator $B_i$, there is a corresponding deaggregator $E_i$ at a later hop, $1 \leq i \leq M$. $R_i$ is the GR for aggregate $A_i$, which starts from $B_i$ and ends at $E_i$. If $1 \leq B_1 < E_1 \leq B_2 < E_2 \leq \cdots \leq B_M < E_M \leq K$ (i.e., aggregation is done sequentially, as shown in Fig. A.2), then the e2e*

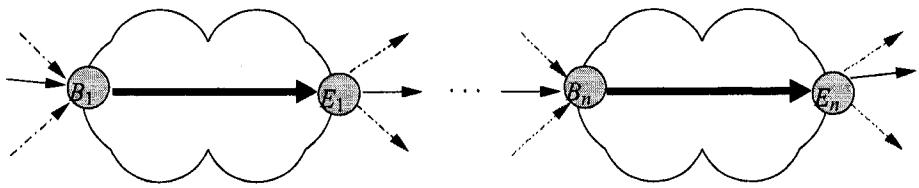



Figure A.2: Multiple sequential aggregation

*delay bound for packet $p_f^j$ is:*

$$d_f^j \leq [GRC^1(p_f^j) - A^1(p_f^j)] + [(K-1) - \sum_{i=1}^{M}(E_i - B_i - 1)]\frac{\ell_f^{max}}{r_f} + \sum_{i=1}^{M}(E_i - B_i - 1)\frac{\ell_{A_i}^{max}}{R_i} + \sum_{i=1}^{M}\frac{\ell_{A_i}^{max}}{R_i} + \sum_{n=1}^{K}\alpha^n. \quad \text{(A.11)}$$

*Proof.* For each aggregate flow $A_i$ ($1 \leq i \leq M$), from Eq. (2.34), we have

$$D^{E_i}(p_f^j) \leq GRC^{B_i}(p_{A_i}^j) + (E_i - B_i)\frac{\ell_{A_i}^{max}}{R_i} + \frac{\ell_f^{max}}{r_f} + \sum_{n=B_i}^{E_i}\alpha^n$$

Thus for flow $f$, the set of nodes from $B_i$ to $E_i$ can be view as a virtual GR server with scheduling constant $\alpha^{B_i} = (E_i - B_i)\frac{\ell_{A_i}^{max}}{R_i} + \frac{\ell_f^{max}}{r_f} + \sum_{n=B_i}^{E_i}\alpha^n$.

From Theorem 2.1.3, we obtain

$$\begin{aligned} D^K(p_f^j) &\leq GRC^1(p_f^j) + [K-1-\sum_{i=1}^{M}(E_i - B_i)]\frac{\ell_f^{max}}{r_f} + \sum_{n=1}^{B_1-1}\alpha^n + \sum_{n=E_M+1}^{K}\alpha^n + \sum_{j=1}^{M-1}\sum_{n=E_j+1}^{B_{j+1}-1}\alpha^n + \sum_{i=1}^{M}\alpha^{B_i} \\ &= GRC^1(p_f^j) + [K-1-\sum_{i=1}^{M}(E_i - B_i)]\frac{\ell_f^{max}}{r_f} + \sum_{n=1}^{K}\alpha^n + \sum_{i=1}^{M}(E_i - B_i)\frac{\ell_{A_i}^{max}}{R_i} + \sum_{i=1}^{M}\frac{\ell_f^{max}}{r_f} \\ &= GRC^1(p_f^j) + [(K-1) - \sum_{i=1}^{M}(E_i - B_i - 1)]\frac{\ell_f^{max}}{r_f} + \sum_{i=1}^{M}(E_i - B_i - 1)\frac{\ell_{A_i}^{max}}{R_i} + \sum_{i=1}^{M}\frac{\ell_{A_i}^{max}}{R_i} + \sum_{n=1}^{K}\alpha^n. \end{aligned}$$

□

Since $B_i$ and $E_i$ are, respectively, the aggregator and deaggregator for aggregate flow $A_i$, they schedule packets of flow $f$. Thus, the number of hops that schedule packets of $A_i$ is $E_i - B_i - 1$ ($1 \leq i \leq M$). Thus, Eq. (A.11) can be rewritten as Eq. (2.36).

With these two lemmas, we can now prove Theorem 2.3.2.

*Proof.* The idea is to prove by induction on the levels of aggregation. Lemmas A.4.1 and A.4.2 have proved the basic cases for the theorem. We use them as two basic building blocks. We only need to show that if all the component regions of the e2e path satisfy the theorem, the whole aggregation region built based on these component regions satisfies the theorem as well. Again, we consider two cases:

*Case 1: Recursive aggregation*

Suppose flow $f$ takes $K$ hops of GR servers. Along the path, there is an aggregation region from $B_1$ to $E_1$ for aggregate flow $A_1$ with guaranteed rate $R_1$. This aggregation region contains another one from $B_2$ and $E_2$ for aggregate flow $A_2$, with guaranteed rate $R_2$. In other words, $1 \leq B_1 < B_2 < E_2 < E_1 \leq K$. If the region from $B_2$ to $E_2$ satisfies

$$D^{E_2}(p^j_{A_1}) \leq GRC^{B_2}(p^j_{A_1}) + \sum_{k=B_2+1}^{E_2} \frac{\ell^{max}_{\hat{A}_k}}{\hat{R}_k} + \sum_{k=2}^{M} \frac{\ell^{max}_{A_k}}{R_k} + \sum_{n=B_2}^{E_2} \alpha^n,$$

then the e2e delay $d^j_f$ satisfies

$$d^j_f \leq [GRC^1(p^j_f) - A^1(p^j_f)] + \sum_{i=2}^{K} \frac{\ell^{max}_{\hat{A}_i}}{\hat{R}_i} + \sum_{i=1}^{M} \frac{\ell^{max}_{A_i}}{R_i} + \sum_{n=1}^{K} \alpha^n.$$

*Case 2: Sequential aggregation*

Suppose flow $f$ takes $K$ hops of GR servers and there are $M$ aggregation regions along the path, each between $B_i$ and $E_i$ ($1 \leq i \leq M$). The guaranteed rate for aggregate flow $A_i$ is $R_i$, which starts from router $B_i$ and ends at router $E_i$. If $1 \leq B_1 < E_1 \leq B_2 < E_2 \leq \cdots \leq B_M < E_M \leq K$ (i.e., the aggregation regions are sequentially placed), and each aggregation region between $B_i$ and $E_i$ satisfies

$$D^{E_i}(p^j_f) \leq GRC^{B_i}(p^j_f) + \sum_{k=B_i+1}^{E_i} \frac{\ell^{max}_{\hat{A}_k}}{\hat{R}_k} + \sum_{k=M_i}^{N_i} \frac{\ell^{max}_{A_k}}{R_k} + \sum_{n=B_i}^{E_i} \alpha^n,$$

where $M_i$ and $N_i$ are the levels of aggregation in the $i^{th}$ aggregation region, then the e2e delay $d^j_f$ satisfies

$$d^j_f \leq [GRC^1(p^j_f) - A^1(p^j_f)] + \sum_{i=2}^{K} \frac{\ell^{max}_{\hat{A}_i}}{\hat{R}_i} + \sum_{i=1}^{M} \sum_{k=M_i}^{N_i} \frac{\ell^{max}_{A_k}}{R_k} + \sum_{n=1}^{K} \alpha^n.$$

The proof of the two cases are similar to those of Lemmas A.4.1 and A.4.2, respectively. Note that the aggregator and deaggregator for aggregate flow $A_i$ do not schedule packets on $A_i$. They

schedule packets of the constituent flows of $A_i$ instead. With these two cases, the theorem is proven, since any kind of multiple aggregations is built by combining the recursive and sequential aggregations. □

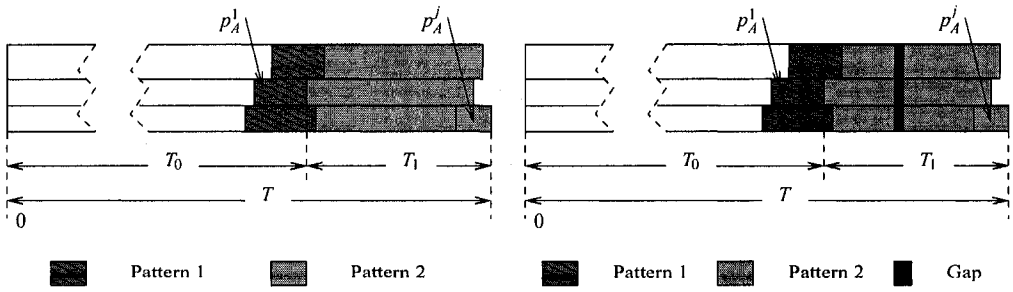
# APPENDIX B

# Coordinated Aggregate Scheduling for Improving End-to-End Delay Performance

## B.1 Proof of Lemma 3.1.1

*Proof.* Since the busy periods of all flows in aggregate $A$ coincide with one another at $S_i$, packets leave the aggregator in the correct order of their GRC values. Therefore, no packet reordering is needed at $S_{i+1}$. Without loss of generality, we consider a busy period of aggregate $A$ at $S_{i+1}$. We also assume that the first busy period of aggregate $A$ at $S_i$ starts at time 0. Note that the busy periods of aggregate $A$ at $S_i$ and $S_{i+1}$ may not coincide with each other. In other words, a busy period at $S_i$ may be split into multiple busy periods at $S_{i+1}$; multiple busy periods (or multiple segments of busy periods) at $S_i$ may be merged into a single busy period at $S_{i+1}$.

Let $p_A^j$ be the $j^{th}$ packet of aggregate $A$ in the busy period.




(a) No gaps between busy periods (b) Gaps between busy periods

Figure B.1: Illustration of scheduling mechanisms of CAS

*Case 1*: $j = 1$. Let $p_A^1 = p_f^m$. Since $p_A^1$ is the first packet of the busy period,

$$GRC^{i+1}(p_A^1) \leq A^{i+1}(p_A^1) + \frac{\ell_A^1}{R}.$$

Also, since $S_i$ is a GR server,

$$\begin{aligned} A^{i+1}(p_A^1) &\leq GRC^i(p_A^1) + \alpha^i \\ \Longrightarrow GRC^{i+1}(p_A^1) &\leq GRC^i(p_A^1) + \frac{\ell_A^1}{R} + \alpha^i \\ &\leq GRC^i(p_f^m) + \frac{\sum_{k=1}^{N} \ell_k^{max}}{R} + \alpha^i. \end{aligned} \tag{B.1}$$

*Case 2*: $j > 1$. Depending on the relationship between the busy periods of $A$ at $S_i$ and $S_{i+1}$, this case can be further divided into two scenarios.

(i) In this scenario we consider the case when a busy period of $A$ at $S_i$ is split into multiple busy periods at $S_{i+1}$ (a special case is that a busy period at $S_i$ corresponds to one at $S_{i+1}$). Suppose $p_A^1$ is the first packet of a busy period at $S_{i+1}$, which is a segment of a busy period at $S_i$, and suppose $p_A^1 = p_g^l$. Then for $p_A^j$,

$$\begin{aligned} GRC^{i+1}(p_A^j) &= A^{i+1}(p_A^1) + \frac{\sum_{k=1}^{j} \ell_A^k}{R} \\ &\leq GRC^i(p_A^1) + \frac{\sum_{k=1}^{j} \ell_A^k}{R} + \alpha^i. \end{aligned} \tag{B.2}$$

At the same time, suppose $p_A^j = p_f^m$, then

$$\begin{aligned} GRC^i(p_A^j) &= \frac{\sum_{k=1}^{m} \ell_f^k}{r_f} \\ \Longrightarrow GRC^{i+1}(p_A^j) - GRC^i(p_A^j) &\leq GRC^i(p_A^1) + \frac{\sum_{k=1}^{j} \ell_A^k}{R} + \alpha^i - \frac{\sum_{k=1}^{m} \ell_f^k}{r_f}. \end{aligned} \tag{B.3}$$

We now prove that

$$GRC^i(p_A^1) + \frac{\sum_{k=1}^{j} \ell_A^k}{R} - \frac{\sum_{k=1}^{m} \ell_f^k}{r_f} \leq \frac{\sum_{k=1}^{N} \ell_k^{max}}{R}. \tag{B.4}$$

Fig. B.1(a) illustrates the idea of the proof by using an aggregate containing three flows. Let $T_0 = GRC^i(p_A^1)$ and $T = GRC^i(p_A^j) = \frac{\sum_{k=1}^{m} \ell_f^k}{r_f}$. Fig. B.1(a) shows the relationship among the GRC

values of the packets at the aggregator $S_i$. Since $p_A^1$ is the first packet in the busy period (which consists of packets in the shaded area), its GRC value at $S_i$, $GRC^i(p_A^1)$, must be the smallest among all the packets in the same busy period. Also, let $SP = \sum_{k=1}^{j} \ell_A^k$, the total shaded area (both patterns 1 and 2). We obtain

$$\begin{aligned} GRC^i(p_A^1) - \frac{\sum_{k=1}^{m} \ell_f^k}{r_f} &= T_0 - T \\ &= -T_1 \\ \Longrightarrow GRC^i(p_A^1) + \frac{\sum_{k=1}^{j} \ell_A^k}{R} - \frac{\sum_{k=1}^{m} \ell_f^k}{r_f} &= \frac{\sum_{k=1}^{j} \ell_A^k}{R} - T_1 \\ &= \frac{SP}{R} - T_1 \\ &\leq \frac{SP_1}{R} \\ &\leq \frac{\sum_{k=1}^{N} \ell_k^{max}}{R}, \end{aligned} \tag{B.5}$$

where $SP_1$ is the pattern 1's shaded area only. The last inequality holds since $SP_1$ contains at most one packet from each flow in the aggregate. Therefore, from Eqs. (B.3) and (B.4), we obtain

$$\begin{aligned} GRC^{i+1}(p_A^j) &\leq GRC^i(p_A^j) + \frac{\sum_{k=1}^{N} \ell_k^{max}}{R} + \alpha^i \\ &= GRC^i(p_f^m) + \frac{\sum_{k=1}^{N} \ell_k^{max}}{R} + \alpha^i. \end{aligned} \tag{B.6}$$

(ii) In the previous scenario we assume that a busy period of $A$ at $S_{i+1}$ must belong to a *single* busy period at $S_i$. In other words, although a busy period at $S_i$ can be split into multiple busy period at $S_{i+1}$, the opposite is not true. In this scenario, we consider the case when a busy period at $S_{i+1}$ indeed consists of multiple busy periods at $S_i$ (or multiple segments of busy periods at $S_i$). As shown in Fig. B.1(b), there will be gaps between the busy periods at $S_i$. Therefore, in the calculation of $GRC^i(p_A^j)$, there will be an extra delay corresponding to the gaps, and its value will be larger than the sum of the corresponding packet sizes.

We prove that the lemma still holds even with the gaps. Similarly to the proof in scenario (i), we obtain

$$GRC^i(p_A^j) = GRC^i(p_f^m)$$

$$= \frac{\sum_{k=1}^{m} \ell_f^k}{r_f} + T_f$$

$$\Longrightarrow GRC^{i+1}(p_A^j) - GRC^i(p_A^j) \leq GRC^i(p_A^1) + \frac{\sum_{k=1}^{j} \ell_A^k}{R} + \alpha^i - (\frac{\sum_{k=1}^{m} \ell_f^k}{r_f} + T_f), \tag{B.7}$$

where $T_f$ is the total length of gaps between the busy periods of flow $f$ before $p_f^m$. As shown in Fig. B.1(b), $T_0 = GRC^i(p_A^1)$, and $T = \frac{\sum_{k=1}^{m} \ell_f^k}{r_f} + T_f$. Also, let $SP = \sum_{k=1}^{j} \ell_A^k + T_{gap} \cdot R$, the total shaded area (both patterns 1 and 2), where $T_{gap}$ is the total length of gaps in the shaded area. Then,

$$GRC^i(p_A^1) - (\frac{\sum_{k=1}^{m} \ell_f^k}{r_f} + T_f) = -T_1$$

$$\Longrightarrow GRC^i(p_A^1) + \frac{\sum_{k=1}^{j} \ell_A^k}{R} - (\frac{\sum_{k=1}^{m} \ell_f^k}{r_f} + T_f) < \frac{\sum_{k=1}^{j} \ell_A^k + T_{gap} \cdot R}{R} - T_1$$

$$\leq \frac{\sum_{k=1}^{N} \ell_k^{max}}{R}. \tag{B.8}$$

From Eqs. (B.7) and (B.8), the lemma still holds. □

## B.2 Proof of Theorem 3.1.1

*Proof.* We prove the theorem in two steps: in the first step, we prove that the theorem holds for an extreme case, in which the busy periods of all flows in the same aggregate coincide with one another at the aggregator. In this case, no packet reordering is needed at downstream nodes since packets leave the aggregator in the correct order of their GRC values. In the second step, we prove that the extreme case is the worst-case scenario and thus, the theorem holds for all other cases.

**Step 1**: In this step, no packet reordering is needed. Let $p_A^{j'}$ be the packet in the aggregate flow $A$ corresponding to the packet $p_f^j$ in flow $f$. From Theorem 3.1.1, we have

$$GRC^2(p_A^{j'}) \leq GRC^1(p_f^j) + \frac{\sum_{k=1}^{N} \ell_k^{max}}{R} + \alpha^1.$$

Since $S_2, \ldots, S_{K-1}$ are all GR servers for aggregate flow $A$ with guaranteed rate $R = \sum_{k=1}^{N} r_k$, from Lemma 2.1.2, we have

$$GRC^3(p_A^{j'}) \leq GRC^2(p_A^{j'}) + \frac{\ell_A^{max}}{R} + \alpha^2,$$

$$GRC^4(p_A^{j'}) \leq GRC^3(p_A^{j'}) + \frac{\ell_A^{max}}{R} + \alpha^3,$$

$$\vdots$$

$$GRC^{K-1}(p_A^{j'}) \leq GRC^{K-2}(p_A^{j'}) + \frac{\ell_A^{max}}{R} + \alpha^{K-2}.$$

Adding them up, we obtain

$$GRC^{K-1}(p_A^{j'}) \leq GRC^1(p_f^j) + \frac{\sum_{k=1}^{N} \ell_k^{max}}{R} + (K-3)\frac{\ell_A^{max}}{R} + \sum_{i=1}^{K-2} \alpha^i.$$

Since $p_f^j = p_A^{j'}$, the departure time of packet $p_f^j$ from $S_{K-1}$ is

$$\begin{aligned} D^{K-1}(p_f^j) &= D^{K-1}(p_A^{j'}) \\ &\leq GRC^{K-1}(p_A^{j'}) + \alpha^{K-1} \\ &\leq GRC^1(p_f^j) + \frac{\sum_{k=1}^{N} \ell_k^{max}}{R} + (K-3)\frac{\ell_A^{max}}{R} + \sum_{i=1}^{K-1} \alpha^i. \end{aligned}$$

From the definition of the GR server, the first ($K$-1) servers for flow $f$ can be viewed as a virtual GR server with scheduling constant $\alpha^* = \frac{\sum_{k=1}^{N} \ell_k^{max}}{R} + (K-3)\frac{\ell_A^{max}}{R} + \sum_{i=1}^{K-1} \alpha^i$. Since $S_K$ is also a GR server for flow $f$ with guaranteed rate $r_f$, from Lemma 2.1.2, we have

$$GRC^K(p_f^j) \leq GRC^1(p_f^j) + \frac{\ell_f^{max}}{r_f} + \alpha^*.$$

Therefore, the departure time of packet $p_f^j$ from $S_K$ is

$$\begin{aligned} D^K(p_f^j) &\leq GRC^K(p_f^j) + \alpha^K \\ &\leq GRC^1(p_f^j) + \frac{\ell_f^{max}}{r_f} + \alpha^* + \alpha^K \\ &= GRC^1(p_f^j) + \frac{\sum_{k=1}^{N} \ell_k^{max}}{R} + (K-3)\frac{\ell_A^{max}}{R} + \frac{\ell_f^{max}}{r_f} + \sum_{i=1}^{K} \alpha^i. \end{aligned} \tag{B.9}$$

Then, the e2e delay $d_f^j$ satisfies

$$\begin{aligned} d_f^j &= D^K(p_f^j) - A^1(p_f^j) \\ &\leq [GRC^1(p_f^j) - A^1(p_f^j)] + \frac{\sum_{k=1}^{N} \ell_k^{max}}{R} + (K-3)\frac{\ell_A^{max}}{R} + \frac{\ell_f^{max}}{r_f} + \sum_{i=1}^{K} \alpha^i. \end{aligned} \tag{B.10}$$

From [4], for flow $f$ conforming to the token bucket model $(\sigma_f, \rho_f)$ and with reserved rate $r_f$ ($r_f \geq \rho_f$), $GRC^1(p_f^j) - A^1(p_f^j) \leq \frac{\sigma_f}{r_f}$. Thus, the theorem follows.

**Step 2**: Now, we prove that the extreme case discussed in the previous step is the worst case for e2e delay. Suppose that among the $N$ flows in aggregate $A$, all other flows' busy periods start at the same time at $S_i$, while flow $f$ remains idle. Then, the delay performances of all the other flows are not negatively affected. In fact, when flow $f$ is idle, the other flows can take advantage of the extra bandwidth and thus have a smaller e2e delay.

However, for flow $f$, since its bandwidth may be used by other flows when it is idle, it may be delayed by the packets from other flows when its busy period starts. We prove that even for flow $f$, the above case is indeed the worst case. We further consider two scenarios:

(i) When flow $f$ is idle, the actual bandwidth used by other flows in aggregate $A$ is not more than the reserved rate $R$. For each scheduling system in which flow $f$ is idle while other flows are in busy period, we construct a corresponding reference system in which flow $f$ is always in busy period as long as other flows are in their busy periods. We prove that the e2e delay of flow $f$ in the original system is not worse than that in the new reference system.

The basic idea is shown in Fig. B.2. Suppose the busy period of flow $f$ (the one with shaded patterns) starts at $t_0$, while all the other flows in the same aggregate start at 0, the beginning of aggregate $A$'s busy period. In the reference system, the idle period of flow $f$ between 0 and $t_0$ is filled with a sequence of packets of flow $f$, which meet the flow $f$'s maximum packet size requirement. The total packet size equals $r_f \cdot t_0$. Thus, in both systems the packets of $f$ after $t_0$ have identical GRC values. At the same time, the finishing time of packets of other flows remain the same, meaning that those packets get smaller proportionally to the extra bandwidth they take from $f$ in the original system.

Now, let us examine the delay of flow-$f$ packets. On one hand, in both systems the packets of $f$ have identical GRC values. On the other hand, in the original system, the packets of other flows have GRC values greater than those in the reference system, since the packets arriving before the first packet of $f$ are larger than those in the reference system. Therefore, the priorities of flow $f$'s packets in the original system are relatively higher than those in the reference system, so the packets of flow $f$ in the original system have delays no worse than those in the reference system.

Since the reference system belongs to the extreme case we considered in the previous step, this proves that the extreme case is indeed the worst case.

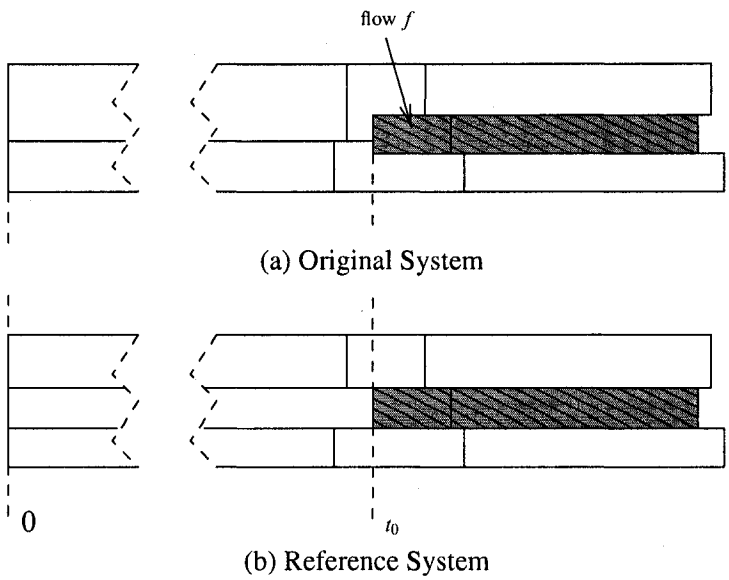



Figure B.2: A reference system

(ii) When flow $f$ is idle, the actual bandwidth used by other flows in aggregate $A$ is more than the reserved rate $R$.

We proof that for any packet from flow $f$, its e2e delay is no worse than the delay bound in the extreme case. Suppose $p_A^j$ is from flow $f$.

If at a downstream node $S_i$, the packet get into its correct place (no packets ahead of it has a larger deadline value), then the delay experienced by $p_A^j$ at the downstream nodes of $S_i$ will be exactly the same as in Step 1. At the same time, the delay experienced by $p_A^j$ up until $S_i$ is also the same, since the delay at $S_i$ will make up the delay discrepancy in upstream nodes. In short, the delay bound of $p_A^j$ will be the same as in the extreme case.

Otherwise, the packet always lags behind its correct place all the way. Then at any give node $S_i$, if the queue for aggregate $A$ is empty, from WF$^2$Q we get $D^i(p_A^j) - A^i(p_A^j) \leq \frac{\ell_A^j}{R} + \Delta_A^i$, where $\Delta_A^i = (\frac{1}{R} - \frac{1}{C^i})\ell_A^{max} + \frac{L_{max}^i}{C^i}$. Thus the delay bound of $p_A^j$ at node $S_i$ is $\frac{\ell_A^j}{R} + (\frac{1}{R} - \frac{1}{C^i})\ell_A^{max} + \frac{L_{max}^i}{C^i}$; If the queue for the aggregate is not empty, and there is one packet $p_A^n$ ahead of $p_A^j$ with larger deadline value, then the delay of the $p_A^j$ is $\frac{\ell_A^n}{C^i}$ longer, so the total delay is less than

$$\frac{\ell_A^j}{R} + (\frac{1}{R} - \frac{1}{C^i})\ell_A^{max} + \frac{L_{max}^i}{C^i} + \frac{\ell_A^n}{C^i} \leq \frac{\ell_A^j}{R} + \frac{\ell_A^{max}}{R} + \frac{L_{max}^i}{C^i}. \tag{B.11}$$

(A detailed proof is as follows: Suppose the remaining size of $p_A^n$ is $\ell_r$. Consider a reference system in which packet $p_A^j$ arrives $\frac{\ell_r}{C^i}$ later than in the original system, i.e., $\hat{A}^i(p_A^j) = A^i(p_A^j) + \frac{\ell_r}{C^i}$. Then in the reference system, $p_A^j$ will enter an empty queue, thus its delay will be $\hat{d} = \frac{\ell_A^j}{R} + (\frac{1}{R} - \frac{1}{C^i})\ell_A^{max} + \frac{L_{max}^i}{C^i}$. Since in both the reference and original system $p_A^j$ departs at the same time, but in the original system it arrives $\frac{\ell_r}{C^i}$ earlier, thus the delay in the original system is $\frac{\ell_r}{C^i}$ longer,

$$\begin{aligned} d &= \hat{d} + \frac{\ell_r}{C^i} \\ &\leq \frac{\ell_A^j}{R} + (\frac{1}{R} - \frac{1}{C^i})\ell_A^{max} + \frac{L_{max}^i}{C^i} + \frac{\ell_r}{C^i} \\ &\leq \frac{\ell_A^j}{R} + \frac{\ell_A^{max}}{R} + \frac{L_{max}^i}{C^i}, \end{aligned} \tag{B.12}$$

since $\ell_r \leq \ell_A^n \leq \ell_A^{max}$. )

If a group of packets from flow $f$ are served consecutively at $S_i$, then suppose the first packet of this group is $p_A^j$, for $p_A^{j+k}$ (k $\geq$ 1), from the definition of GRC value,

$$\begin{aligned} GRC^i(p_A^{j+k}) &\geq GRC^i(p_A^{j+k-1}) + \frac{\ell_f^{j+k}}{R} \\ &\geq GRC^i(p_A^j) + \frac{\sum_{m=1}^{k} \ell_f^{j+m}}{R}. \end{aligned} \tag{B.13}$$

Now consider a packet $p_f^j$. If at a node a packet from other flows is served between it and its predecessor $p_f^{j-1}$, we consider it has a delay bound $\frac{\ell_A^j}{R} + \frac{\ell_A^{max}}{R} + \frac{L_{max}^i}{C^i}$ at that node and disregard the node from the following discussion. Then there are only those nodes that $p_f^j$ is consecutively served after its predecessor. This is illustrated in Fig.B.3. Note that this is done independent of the delay experienced by $p_f^{j-1}$ at the disregarded nodes.

Then, for the remaining nodes, the delay can be proven similarly.

Let us redefine GRC value as follows:

$$GRC^i(p_f^j) = \begin{cases} 0, & j = 0 \\ \max\{A^i(p_f^j), GRC^i(p_f^{j-1})\} + \frac{\ell_f^j}{r_f}, & j \geq 1, \end{cases} \tag{B.14}$$

where $p_f^1$ is the first packet of a sequence of consecutive packets from flow $f$.

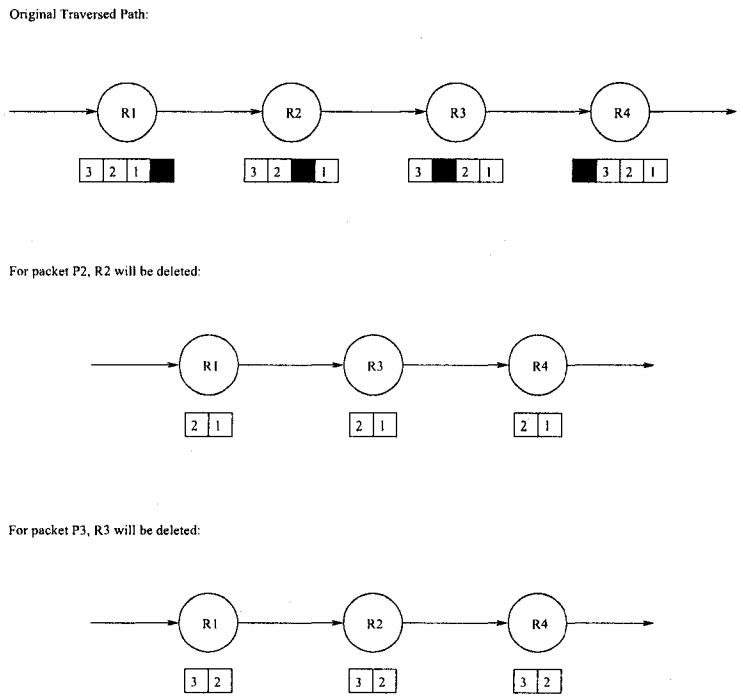



Figure B.3: An example of isolating nodes

Then, we can prove that WF$^2$Q is a GR server for flow $f$ with $\hat{\alpha}^i = \alpha^i + \frac{\ell_A^{max}}{R}$.

$$GRC^{i+1}(p_f^j) = GRC^i(p_f^j) + \frac{\ell_A^{max}}{R} + \hat{\alpha}^i. \tag{B.15}$$

Next, we can prove that (Since

$$\begin{aligned} GRC^i(p_f^j) &\leq A^i(p_f^j) + \frac{\ell_A^{max}}{R} \\ \Longrightarrow GRC^i(p_f^{j+k}) &\leq A^i(p_f^j) + \frac{\ell_A^{max}}{R} + \frac{\sum_{m=1}^{k} \ell_f^{j+m}}{R} \\ &\leq GRC^{i-1}(p_f^j) + \alpha^{i-1} + \frac{\ell_A^{max}}{R} + \frac{\sum_{m=1}^{k} \ell_f^{j+m}}{R} \\ &\leq GRC^{i-1}(p_f^{j+k}) + \alpha^{i-1} + \frac{\ell_A^{max}}{R}. \end{aligned} \tag{B.16}$$

)

This completes the proof. □

Also, as a general discussion, the reordering makes the traditional GRC value at the node meaningless, because the departure time of a packet at an intermediate node is sort of independent of its

actual arrival time at the node (due to the reordering). The performance of CAS is more dependent of the fairness property of the scheduling algorithm. Thus different scheduling algorithms could have (very) different performance. WF$^2$Q is the best we know.

## B.3 General proof that CAS is superior to VAS

The proof that CAS is no worse than VAS:

(i) If there is no reordering during the e2e transmission of packet $p_f^j$, then the delays in both cases are the same;

(ii) If reordering occurs, then the delay in CAS is no worse than the reference case in VAS.

Let us focus on two packets $p_g^k$ and $p_f^j$ at a node $S_i$. When the reordering occurs at node $S_i$, it does not affect the delay of the packets ahead and behind the two packets involved. Also, if $p_g^k$ and $p_f^j$ exchange places from $p_g^k, p_f^j$ to $p_f^j, p_g^k$, it does not mean $p_g^k$ leaves the aggregator later than it does in the original case. It is just that it meets $p_f^j$, which appears ahead of it in queue at node $S_i$. The delay of $p_g^k$ at upstream nodes $S_1, \cdots, S_{i-1}$ remains the same. Thus the GRC value of $p_g^k$ at nodes $S_1, \cdots, S_{i-1}$ are smaller than that in the reference system, which means it can potentially have smaller delay at those nodes. It actually has smaller delay at the aggregator since it is scheduled ahead of $p_f^j$.

Under token bucket traffic model, there are limits on how much traffic from other flows can be ahead of a given flow. Thus the extra delay caused by those traffic is limited. Continue the above example, other flows many have packets ahead of $p_g^k$, but the amount of traffic is limited; then, moving $p_f^j$ ahead of $p_g^k$ from behind it, it only adds one packet from flow $f$, which initially was not disturbing flow $g$ at all. In fact, the amount of disturbing traffic is far from its limit since flow $f$ is well below its potential.

# BIBLIOGRAPHY

# BIBLIOGRAPHY


[1] R. Braden, D. Clark, and S. Shenker, "Integrated services in the Internet architecture: an overview," RFC 1633, June 1994.

[2] B. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource ReSerVation Protocol (RSVP) — version 1 functional specification," RFC 2205, Sept. 1997.

[3] H. Zhang, "Service disciplines for guaranteed performance service in packet-switching networks," *Proc. IEEE*, vol. 83, no. 10, pp. 1374–1396, Oct. 1995.

[4] P. Goyal, S. S. Lam, and H. M. Vin, "Determining end-to-end delay bounds in heterogeneous networks," in *Proc. of NOSSDAV'95*, Apr. 1995, pp. 287–298.

[5] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The single node case," *IEEE/ACM Trans. Networking*, vol. 1, no. 3, pp. 344–357, June 1993.

[6] L. Zhang, "Virtual Clock: A new traffic control algorithm for packet-switched networks," *ACM Trans. Computer Systems*, vol. 9, no. 2, pp. 101–124, May 1991.

[7] S. Blake, D. L. Black, M. A. Carlson, E. Davies, Z. Wang, and W. Weiss, "An architecture for differentiated services," RFC 2475, Dec. 1998.

[8] B. Davie, A. Charny, J. C. R. Bennett, K. Benson, J.-Y. Le Boudec, B. Courtney, S. Davari, V. Firoiu, and D. Stiliadis, "An expedited forwarding PHB (Per-Hop Behavior)," RFC 3246, Mar. 2002.

[9] A. Charny and J.-Y. Le Boudec, "Delay bounds in a network with aggregate scheduling," in *Proc. of QofIS'00*, Oct. 2000, pp. 1–13.

[10] I. Stoica, "Stateless Core: A scalable approach for Quality of Service in the Internet," Ph.D. dissertation, CMU, CMU-CS-00-176, Dec. 2000.

[11] F. Baker, C. Iturralde, F. Le Faucheur, and B. Davie, "Aggregation of RSVP for IPv4 and IPv6 reservation," RFC 3175, Sept. 2001.

[12] J. Ehrensberger, "Resource demand of aggregated resource reservations," in *1st European Conf. on Universal Multiservice Networks (ECUMN)*, Oct. 2000, pp. 56–61.

[13] O. Schelén and S. Pink, "Aggregating resource reservation over multiple routing domains," in *Proc. of IEEE IWQoS'98*, May 1998, pp. 29–32.

[14] A. Terzis, L. Zhang, and E. L. Hahne, "Reservations for aggregate traffic: Experiences from an RSVP tunnels implementation," in *Proc. of IEEE IWQoS'98*, May 1998, pp. 23–25.

[15] B.-K. Choi, D. Xuan, R. Bettati, and W. Zhao, "Scalable QoS guaranteed communication services for real-time applications," in *Proc. of IEEE ICDCS'00*, Apr. 2000, pp. 180–187.

[16] H. Fu and E. W. Knightly, "Aggregation and scalable QoS: A performance study," in *Proc. of IEEE IWQoS'01*, June 2001, pp. 39–50.

[17] S. Berson and S. Vincent, "Aggregation of Internet integrated services state," in *Proc. of IEEE IWQoS'98*, May 1998, pp. 26–28.

[18] H. Zhang and S. Keshav, "Comparison of rate-based service disciplines," in *Proc. of ACM SIGCOMM'91*, Aug. 1991, pp. 113–121.

[19] T. Li and Y. Rekhter, "A provider architecture for differentiated services and traffic engineering (PASTE)," RFC 2430, Oct. 1998.

[20] D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell, and J. McManus, "Requirements for traffic engineering over MPLS," RFC 2702, Sept. 1999.

[21] H. Fu and E. W. Knightly, "A simple model of real-time flow aggregation," *IEEE/ACM Trans. Networking*, vol. 11, no. 3, pp. 422–435, June 2003.

[22] K. Dolzer, W. Payer, and M. Eberspächer, "A simulation study on traffic aggregation in multi-service networks," in *Proc. of IEEE Conf. on High Performance Switching and Routing*, June 2000, pp. 157–165.

[23] S. Vutukury and J. Garcia-Luna-Aceves, "A scalable architecture for providing deterministic guarantees," in *Proc. of IEEE IC3N'99*, Oct. 1999, pp. 534–539.

[24] R. Teixeira, A. Shaikh, T. Griffin, and J. Rexford, "Dynamics of hot-potato routing in IP networks," in *Proc. of ACM SIGMETRICS'04*, June 2004, pp. 307–319.

[25] C. Labovitz, G. R. Malan, and F. Jahanian, "Internet routing instability," in *Proc. of ACM SIGCOMM'97*, Sept. 1997, pp. 115–126.

[26] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian, "Delayed Internet routing convergence," in *Proc. of ACM SIGCOMM'00*, Aug. 2000, pp. 175–187.

[27] C. Labovitz, A. Ahuja, R. Wattenhofer, and S. Venkatachary, "The impact of Internet policy and topology on delayed routing convergence," in *Proc. of IEEE INFOCOM'01*, Apr. 2001, pp. 537–546.

[28] B. J. Premore, "An analysis of convergence properties of the Border Gateway Protocol using discrete event simulation," Ph.D. dissertation, Dartmouth College, Department of Computer Science, Technical Report TR2003-452, May 2003.

[29] G. Huston, "Analyzing the Internet BGP routing table," *The Internet Protocol Journal*, vol. 4, no. 1, pp. 2–15, Mar. 2001.

[30] T. Bu, L. Gao, and D. Towsley, "On characterizing BGP routing table growth," in *Proc. of IEEE Global Internet Symposium'02*, Nov. 2002.

[31] A. Akella, B. Maggs, S. Seshan, A. Shaikh, and R. Sitaraman, "A measurement-based analysis of multihoming," in *Proc. of ACM SIGCOMM'03*, Aug. 2003, pp. 353–364.

[32] W. Sun and K. G. Shin, "End-to-end delay bounds for traffic aggregates under guaranteed rate scheduling algorithms," *IEEE/ACM Trans. Networking*, vol. 13, no. 5, Oct. 2005.

[33] ——, "Coordinated aggregate scheduling for improving end-to-end delay performance," in *Proc. of IEEE IWQoS'04*, Montréal, CA, June 2004, pp. 77–86.

[34] W. Sun, Z. M. Mao, and K. G. Shin, "Differentiated BGP update processing for improved routing convergence," in *Proc. of IEEE ICNP'06*, Santa Barbara, CA, Nov. 2006, pp. 280–289.

[35] ——, "On understanding impact of BGP routing changes on application traffic," under submission to IEEE INFOCOM 2008.

[36] D. Stiliadis and A. Varma, "Latency-rate servers: A general model for analysis of traffic scheduling algorithms," *IEEE/ACM Trans. Networking*, vol. 6, no. 5, pp. 611–624, Aug. 1998.

[37] J. A. Cobb, "Preserving quality of service guarantees in spite of flow aggregation," in *Proc. of IEEE ICNP'98*, Oct. 1998, pp. 90–97.

[38] ——, "Preserving quality of service guarantees in spite of flow aggregation," *IEEE/ACM Trans. Networking*, vol. 10, no. 1, pp. 43–53, Feb. 2002.

[39] J. C. R. Bennett and H. Zhang, "Hierarchical packet fair queueing algorithms," in *Proc. of ACM SIGCOMM'96*, Aug. 1996, pp. 143–156.

[40] ——, "$WF^2Q$: Worst-case fair weighted fair queueing," in *Proc. of IEEE INFOCOM'96*, Mar. 1996, pp. 120–128.

[41] J.-Y. Le Boudec, "Application of network calculus to guaranteed service networks," *IEEE Trans. Inform. Theory*, vol. 44, no. 3, pp. 1087–1096, May 1998.

[42] H. Zhang and D. Ferrari, "Rate-controlled service disciplines," *J. of High Speed Networks*, vol. 3, no. 4, pp. 389–412, 1994.

[43] S. Keshav, *An Engineering Approach to Computer Networking*. Addison Wesley, 1997.

[44] "ns2 Simulator." [Online]. Available: http://www.isi.edu/nsnam/ns/

[45] R. A. Guérin and V. Pla, "Aggregation and conformance in differentiated service networks: A case study," *ACM Computer Communication Review*, vol. 31, no. 1, pp. 21–32, Jan. 2001.

[46] J. Sahni, P. Goyal, and H. M. Vin, "Scheduling CBR flows: FIFO or per-flow queueing?" in *Proc. of NOSSDAV'99*, June 1999.

[47] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson, "On the self-similar nature of Ethernet traffic (extended version)," *IEEE/ACM Trans. Networking*, vol. 2, no. 1, pp. 1–15, Feb. 1994.

[48] A. Popescu, "Traffic self-similarity (invited tutorial)," in *Proc. of IEEE Intl. Conf on Telecommunications (ICT2001)*, June 2001.

[49] R. Jain, *The Art of Computer Systems Performance Analysis.* John Wiley & Sons, 1991.

[50] F. Fitzek and M. Reisslein, "MPEG-4 and H.263 video traces for network performance evaluation," *IEEE Network*, vol. 15, no. 6, pp. 40–54, Nov./Dec. 2001.

[51] J. L. Rexford, A. G. Greenberg, and F. G. Bonomi, "Hardware-efficient fair queueing architectures for high-speed networks," in *Proc. of IEEE INFOCOM'96*, Mar. 1996, pp. 638–646.

[52] J. C. R. Bennett, D. C. Stephens, and H. Zhang, "High speed, scalable, and accurate implementation of packet fair queueing algorithms in ATM networks," in *Proc. of IEEE ICNP'97*, Oct. 1997, pp. 7–14.

[53] D. C. Stephens and H. Zhang, "Implementing distributed packet fair queueing in a scalable switch architecture," in *Proc. of IEEE INFOCOM'98*, Mar. 1998, pp. 282–290.

[54] F. M. Chiussi and V. Sivaraman, "Implementing fair queueing in ATM switches. I. a practical methodology for the analysis of delay bounds," in *GLOBECOM'97*, Nov. 1997, pp. 509–518.

[55] F. M. Chiussi and A. Francini, "Implementing fair queueing in ATM switches: the discrete-rate approach," in *Proc. of IEEE INFOCOM'98*, Mar. 1998, pp. 272 –281.

[56] J. Schmitt, M. Karsten, L. Wolf, and R. Steinmetz, "Aggregation of guaranteed service flows," in *Proc. of IEEE IWQoS'99*, June 1999, pp. 147–155.

[57] R. Brown, "Calendar queues: A fast O(1) priority queue implementation for the simulation event set problem," *Communications of the ACM*, vol. 31, no. 10, pp. 1220–1227, Oct. 1988.

[58] I. Stoica and H. Zhang, "Providing guaranteed services without per flow management," in *Proc. of ACM SIGCOMM'99*, Sept. 1999, pp. 81–94.

[59] J. Kaur and H. M. Vin, "Core-stateless guaranteed rate scheduling algorithms," in *Proc. of IEEE INFOCOM'01*, Apr. 2001, pp. 1484–1492.

[60] Z.-L. Zhang, Z. Duan, and Y. T. Hou, "Virtual time reference system: A unifying scheduling framework for scalable support of guaranteed services," *IEEE J. Select. Areas Commun.*, vol. 18, no. 12, pp. 2684–2695, Dec. 2000.

[61] M. Andrews and L. Zhang, "Minimizing end-to-end delay in high-speed networks with a simple coordinated schedule," in *Proc. of IEEE INFOCOM'99*, vol. 1, Mar. 1999, pp. 380–388.

[62] C. Li and E. W. Knightly, "Coordinated multihop scheduling: A framework for end-to-end services," *IEEE/ACM Trans. Networking*, vol. 10, no. 6, pp. 776–789, Dec. 2002.

[63] C. Villamizar, R. Chandra, and R. Govindan, "BGP Route Flap Damping," RFC 2439, Nov. 1998.

[64] Y. Rekhter and T. Li, "A border gateway protocol 4 (BGP-4)," RFC 1771, Mar. 1995.

[65] Z. M. Mao, R. Bush, T. G. Griffin, and M. Roughan, "BGP beacons," in *Proc. of Internet Measurement Conference (IMC)'03*, Oct. 2003.

[66] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, , and J. van der Merwe, "Design and implementation of a routing control platform," in *Proc. of USENIX NSDI'05*, May 2005.

[67] M. Handley, E. Kohler, A. Ghosh, O. Hodson, and P. Radoslavov, "Designing extensible IP router software," in *Proc. of USENIX NSDI'05*, May 2005.

[68] "GNU Zebra." [Online]. Available: http://www.zebra.org/

[69] L. Wang, X. Zhao, D. Pei, R. Bush, D. Massey, A. Mankin, S. F. Wu, and L. Zhang, "Observation and analysis of BGP behavior under stress," in *Proc. of Internet Measurement Workshop'02*, Nov. 2002, pp. 217–229.

[70] M. Lad, X. Zhao, B. Zhang, D. Massey, and L. Zhang, "Analysis of BGP update surge during slammer worm attack," in *IWDC'03*, Dec. 2003.

[71] S. Agarwal, C.-N. Chuah, S. Bhattacharrya, and C. Diot, "The impact of BGP dynamics on router CPU utilization," in *Proc. of Passive & Active Measurement Workshop*, Apr. 2004, pp. 278–288.

[72] A. Feldmann, H. Kong, O. Maennel, and A. Tudor, "Measuring BGP pass-through times," in *Proc. of Passive & Active Measurement Workshop*, Apr. 2004.

[73] S. R. Sangli, Y. Rekhter, R. Fernando, J. G. Scudder, and E. Chen, "Graceful restart mechanism for BGP," draft-ietf-idr-restart-10.txt, June 2004.

[74] T. G. Griffin and B. J. Premore, "An experimental analysis of BGP convergence time," in *Proc. of IEEE ICNP'01*, Nov. 2001, pp. 53–61.

[75] "Juniper MRAI." [Online]. Available: https://www.juniper.net/techpubs/software/junos/junos57/swconfig57-routing/html/bgp-summary32.html

[76] D. Pei, X. Zhao, L. Wang, D. Massey, A. Mankin, S. F. Wu, and L. Zhang, "Improving BGP convergence through consistency assertions," in *Proc. of IEEE INFOCOM'02*, June 2002, pp. 902–911.

[77] D. Pei, M. Azuma, N. Nguyen, J. Chen, D. Massey, and L. Zhang, "BGP-RCN: Improving BGP convergence through root cause notification," Computer Science Department, UCLA, Tech. Rep. TR-030047, Oct. 2003.

[78] J. Chandrashekar, Z. Duan, Z.-L. Zhang, and J. Krasky, "Limiting path exploration in BGP," in *Proc. of IEEE INFOCOM'05*, Mar. 2005.

[79] A. Bremler-Barr, Y. Afek, and S. Schwarz, "Improved BGP convergence via ghost flushing," in *Proc. of IEEE INFOCOM'03*, Apr. 2003, pp. 927–937.

[80] R. White, "High availability in routing," *The Internet Protocol Journal*, vol. 7, no. 1, pp. 2–14, Mar. 2004.

[81] "University of Oregon Route Views Project." [Online]. Available: http://www.routeviews.org/

[82] L. Subramanian, S. Agarwal, J. Rexford, and R. H. Katz, "Characterizing the Internet hierarchy from multiple vantage points," in *Proc. of IEEE INFOCOM'02*, June 2002, pp. 594–603.

[83] "Scalable Simulation Framework (SSF)." [Online]. Available: http://www.ssfnet.org/

[84] F. Hao and P. Koppol, "An Internet scale simulation setup for BGP," *ACM Computer Communication Review*, vol. 33, no. 3, pp. 43–57, July 2003.

[85] B. J. Premore, "Multi-AS topologies from BGP routing tables." [Online]. Available: http://www.ssfnet.org/Exchange/gallery/asgraph/index.html

[86] N. Kushman, S. Kandula, and D. Katabi, "Can you hear me now?! It must be BGP," *ACM Computer Communication Review*, 2007.

[87] H. Chang, S. Jamin, Z. M. Mao, and W. Willinger, "An empirical approach to modeling inter-AS traffic matrics," in *Proc. of Internet Measurement Conference (IMC)*, Oct. 2005.

[88] L. Gao, "On inferring autonomous system relationships in the Internet," *IEEE/ACM Trans. Networking*, vol. 9, no. 6, Dec. 2001.

[89] J. Xia and L. Gao, "On the evaluation of AS relationship inferences," in *Proc. of IEEE Globecom'04*, Nov. 2004.

[90] X. Dimitropoulos, D. Krioukov, M. Fomenkov, B. Huffaker, Y. Hyun, and K. C. abd George Wiley, "AS relationships: Inference and validation," CAIDA, Tech. Rep., 2006.

[91] A. Feldmann, O. Maennel, Z. M. Mao, A. Berger, and B. Maggs, "Locating internet routing instabilities," in *Proc. of ACM SIGCOMM'04*, Aug. 2004.

[92] J. Wu, Z. M. Mao, J. Rexford, and J. Wang, "Finding a needle in a haystack: Pinpointing significant BGP routing changes in an IP network," in *Proc. of USENIX NSDI'05*, May 2005.

[93] "Cisco Netflow." [Online]. Available: http://www.cisco.com/en/US/products/ps6601/products_ios_protocol_group_home.html

[94] J. Rexford, J. Wang, Z. Xiao, and Y. Zhang, "BGP routing stability of popular destinations," in *Proc. of Internet Measurement Workshop (IMC)*, Nov. 2002.

[95] "Alexa Traffic Rankings." [Online]. Available: http://www.alexa.com/

[96] B. Zhang, V. Kambhampati, M. Lad, D. Massey, and L. Zhang, "Identifying BGP routing table transfer," in *Proceedings of ACM SIGCOMM Workshop on Mining Network Data (MineNet-05)*, 2005.

[97] R. Oliveira, B. Zhang, D. Pei, Izhak-Ratzin, and L. Zhang, "Quantifying path exploration in the Internet," in *Proc. of Internet Measurement Conference'06*, 2006.

[98] J. Li, M. Guidero, Z. Wu, E. Purpus, and T. Ehrenkranz, "BGP routing dynamics revisited," *ACM Computer Communication Review*, vol. 37, no. 2, pp. 7–16, Apr. 2007.

[99] J. Karlin, S. Forrest, and J. Rexford, "Pretty Good BGP: Improving BGP by cautiously adopting routes," in *Proc. of IEEE ICNP'06*, Nov. 2006.

[100] P. Verkaik, A. Broido, K. Claffy, R. Gao, Y. Hyun, and R. van der Pol, "Complexity of global routing policies," CAIDA, Tech. Rep., 2004. [Online]. Available: http://www.caida.org/funding/routing/atoms/documents/atoms-widew0311.pdf

[101] X. Zhang, P. Francis, J. Wang, and K. Yoshida, "Scaling IP routing with the core router-integrated overlay," in *Proc. of IEEE ICNP'06*, Nov. 2006, pp. 147–156.

[102] E. Karpilovsky and J. Rexford, "Using forgetful routing to control BGP table size," in *Proc. of CoNEXT'06*, Dec. 2006.

[103] W. Sun and K. G. Shin, "TCP performance under aggregate fair queueing," in *Proc. of IEEE Globecom'04*, Dallas, TX, Dec. 2004, pp. 1308–1313.