CERIAS Tech Report 2003-37

MONITORING AND CONTROLLING QOS NETWORK DOMAINS: AN EDGE-TO-EDGE APPROACH

by Md Ahsan Habib

Center for Education and Research in
Information Assurance and Security,
Purdue University, West Lafayette, IN 47907-2086

MONITORING AND CONTROLLING QOS NETWORK DOMAINS: AN EDGE-TO-EDGE APPROACH

A Thesis

Submitted to the Faculty

of

Purdue University

by

Md Ahsan Habib

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

August 2003

To my parents

ACKNOWLEDGMENTS

It is my privilege to work with my advisor, Prof. Bharat Bhargava. I would like to express my sincere appreciation and gratitude to Prof. Bhargava for his guidance, support, and encouragement during the time of this thesis's work. Without his support, I would not be able to overcome many difficulties in my research and this thesis would not have been possible. He has provided an excellent working environment. I also thank him to let us work as an independent researcher.

Special thanks to Prof. Sonia Fahmy for her time, ideas, and active involvement with this research. I remember meeting her days after days to shape things up. Without her great help, I would not be able to finish my thesis by this time.

I would like to thank Prof. David Yau, Prof. Samuel Wagstaff, and Prof. Wojciech Szpankowski for serving as committee members, and providing insightful comments about my research. Discussion with them about this research helps me to address many problems from different angles.

I am very grateful to Mohamed Hefeeda for his constant encouragement and discussion about my thesis work. He provided me great assistance in every step of the thesis work. I want to thank Maleq Khan for his help to analyze many research problems with me. Without his help, I would not be able to provide a solid analysis of overlay-based distributed network monitoring. I want to thank Srinivas R. Avasarala and Venkatesh Prabhakar for their idea to measure delay. Many thanks to Yi Lu, Yuhui Zhong, Weichow Wang, Leszek Lillen, Florian Baumgartner, and Sarika Agarwal for their ideas and encouragement to finish this thesis.

Finally, I owe special gratitude to my family for continuous and unconditional support. Without their support and love for me, I would never achieve my current position.

This research is sponsored in part by the NSF grants ANI-0219110, CCR-001712, and CCR-001788, CERIAS grants, and IBM SUR grant.

TABLE OF CONTENTS

Page

Page

Page

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

Figure Page

Figure Page

ABSTRACT

Habib, Md Ahsan. Ph.D., Purdue University, August, 2003. Monitoring and Controlling QoS Network Domains: An Edge-to-Edge Approach. Major Professor: Bharat Bhargava.

This research studies and designs techniques for coordinated network monitoring, traffic conditioning, and flow control as integral components of the edge routers in a network domain. The enhanced edge routers yield secure network domains, and achieve better performance in terms of high data throughput, low delay, and low loss rates. The potential performance gain from the proposed techniques is critical for the current and emerging network services such as multimedia applications. Using simulation, we evaluate the edge router for data intensive applications such as FTP and delay sensitive applications such as Telnet and Web. The contributions of this thesis can be summarized as follows:

- *Network Monitoring.* Continuous monitoring of network activity is required to maintain confidence in the security of networks with quality of service (QoS) support. The flows are monitored for possible service level agreement violations and bandwidth theft attacks. We design and evaluate *tomography-based* and *overlay-based* network monitoring methodologies for efficient and scalable network monitoring. These schemes infer the internal characteristics of a network domain without involving the core routers. Our results show that we can monitor a network domain with $O(n)$ probes, where n is the number of edge routers. Monitoring mechanism can be used to detect denial of service (DoS) attacks at an early stage. A quantitative comparison among schemes to defeat DoS attacks is conducted, in which, we highlight the merits of each scheme and estimate the processing and communication overhead introduced by it. The comparison provides guidelines for selecting the appropriate scheme based on the requirements and how much overhead can be tolerated.

- *Policy Enforcement.* To improve the QoS, we enforce policy on the incoming flows and focus on congestion and unfairness in network resource allocation problems. We design an adaptive conditioner that considers congestion window size, round trip

time (RTT), retransmission time-out, and explicit congestion notifi cation information to mark and shape a flow. The conditioner also improves the fairness among long and short-RTT flows.

- *Flow Control.* We use the differentiated services framework and network tomography to detect the unresponsive flows. The flow detection mechanism is scalable, and it requires very low overhead. An adaptive congestion control framework is designed that follows TCP-like congestion control algorithm and regulates the unresponsive flows to alleviate the congestion.

1. INTRODUCTION AND RESEARCH DIRECTION

Continuous monitoring of network activity is required to maintain confi dence in the security of networks with quality of service (QoS) support. Moreover, the growth of multimedia applications and high speed networks increases the demand for QoS and congestion control in the Internet. Poor network performance by excessive delays and losses experienced by the users' applications, and/or lack of security is not acceptable. To ensure that a network domain is not experiencing attacks or service violations, we devise low overhead network monitoring schemes. To improve the QoS, we focus on congestion and unfairness in network resource allocation problem. Proper traffi c conditioning (marking, shaping) with unresponsive fbw control can solve the congestion and unfairness problem. All these solutions have to be scalable otherwise they will not be deployable in the heterogeneous Internet.

One solution to provide QoS is to increase the available bandwidth to avoid congestion. However, It does not provide proper resource utilization. This over-provisioning solution has been effective in its support of applications like FTP, HTTP, and e-mail. The problem is more than a bandwidth issue. The problem is that the traffi c has also changed in nature. There are many new types of applications, and they have very different operational requirements. Internet has moved into the global communication, and so is their applications. The Internet is driving the convergence of the telephone and Internet industries. Internet telephony (Voice over IP) and other multimedia applications require service guarantees, and have timing requirements. These applications require network services beyond the "best-effort" service that is delivered by IP networks.

1.1 Framework to Provide QoS

The Internet Engineering Task Force (IETF) [1] has proposed many service models and mechanisms to meet the demand for QoS. A survey of the proposed frameworks is provided

in [2]. Notably among them are the Integrated Services (IntServ) or RSVP model [3,4], the Differentiated Services (DiffServ) model [5, 6], and MPLS [7].

The IntServ model [3] provides guaranteed service in addition to the Best Effort (BE) service. The RSVP was proposed as a signaling protocol for applications to reserve resources [4]. The applications first set up paths and reserve resources before data are transmitted. The difficulty with the IntServ architecture is that the amount of state information depends on the number of flows. This places a huge storage and processing overhead on the routers. Therefore, this architecture does not scale well in the Internet core. In addition, all routers must implement RSVP, admission control, packet classification, and scheduling for the framework come into reality. Stoica *et al.* [8] show that guaranteed services can be provided without per flow state information at the core. This work uses dynamic packet state to convey reservation information from routers to routers.

Because of the difficulty in implementing and deploying IntServ and RSVP, the DiffServ is introduced in early 1998. In DiffServ framework, packets are marked differently to provide different services. Priorities are set on incoming traffic in the TOS byte of the IP header to indicate that a special service is required such as low delay, high throughput, or low loss rate. The edge routers at the boundary of a network domain is responsible for bandwidth reservations. In order to receive differentiated services from its Internet Service Provider (ISP), the users must have a Service Level Agreement (SLA) with its ISP. A SLA specifies the service classes supported and the amount of traffic allowed in each class. The core routers provide a forwarding mechanism based on the traffic class to achieve the service differentiation. The amount of state information stored at the core is proportional to the number of classes rather than the number of flows. Differentiated service is therefore more scalable. Packet classification, marking, policing and shaping operations are only needed at the edge routers. The ISP core routers need only to implement simple forwarding mechanism. Therefore, it is easier to implement and deploy differentiated services. The DiffServ framework needs signaling mechanism and IntServ framework can serve the purpose. The interoperability of DiffServ and IntServ is studied in [9].

The motivation for MPLS is to use a fixed length label to decide packet handling. The MPLS is a forwarding scheme, and a useful tool for traffic engineering [10, 11]. Each MPLS packet has a header, which is encapsulated between the link layer header and the network layer header [7]. A MPLS capable router, termed as Label Switched Router (LSR), examines only the label in forwarding the packet. The network protocol can be IP or others.

This is why it is called Multi-Protocol Label Switching. Compared to other tunneling mechanisms, the MPLS is unique because it can control the complete path of a packet without explicitly specifying the intermediate routers. The MPLS provides faster packet classifi cation and forwarding and effi cient tunneling mechanism.

We prefer to use the DiffServ framework in our research because this framework is scalable. Most of our solutions assume the presence of edge routers to conduct traffi c conditioning, monitoring, and flow control. The core routers only forward the packets.

1.2 Differentiated Services Architecture

In DiffServ architecture, all complexities are pushed to the boundary routers of a network domain to keep the core routers simple and scalable [5, 6]. The DiffServ model uses edge routers at the boundary of an administrative domain to shape, mark, and drop the traffi c if necessary. The operations are based on Service Level Agreements (SLAs) between adjacent domains [12, 13]. The traffi c enters into a DiffServ domain through an ingress routers and leaves a domain at an egress router. An ingress router is responsible for ensuring that the traffi c entering into the domain conforms to any SLA between it and the other domain to which the ingress node is connected. Egress routers may perform traffi c conditioning functions on traffi c forwarded to a directly connected peering domain, depending on the details of the SLA between the two domains. The framework to setup policy and controlling incoming traffi c in the domain is studied in [14].

Edge Routers. The edge routers conduct admission control to regulate the incoming flows. It may re-mark a traffi c stream, shape, or drop packets to alter the temporal characteristics of the stream and bring it into compliance with a traffi c profi le specifi ed by the network administrator. A Marker distributes the excess bandwidth using a predefi ned algorithm to improve the QoS [15, 16]. The marker puts a tag on each packet based on the target rate from the SLA and the current flow rate. An incoming packet is marked as *IN* profi le (low probability to drop) if the corresponding flow does not reach the target rate otherwise the packet is marked to a higher drop precedence. The shaping reduces the traffi c variation and makes it smooth. It provides an upper bound for the rate at which the flow traffi c is admitted into the network. Droppers drop some or all of the packets in a traffi c stream in order to bring the stream into compliance with the traffi c profi le. This process is know as *policing* the stream.

Core Routers. In the core of the network, Per Hop Behaviors (PHBs) achieve service differentiation using a differential drop algorithm. The current DiffServ model defines two forwarding mechanisms: Expedited Forwarding [17] and Assured Forwarding (AF) [18]. The AF PHB is studied with token bucket marker by Ibanez and Nichols [19] and suggested the factors to determine throughput of flows. The AF provides four classes (queues) of delivery for IP packets and three levels of drop precedence (DP0, DP1, and DP2) per class. Three drop precedences are proven to be useful [20, 21] to improve QoS for AF traffic in DiffServ networks. The DP0 has the lowest precedence to drop packet during congestion, and the DP2 has the highest precedence. The Differentiated Services Code Point (DSCP) [22], contained in the IP header DSFIELD/ToS, is set to mark the precedence. When congestion occurs, packets marked with higher precedence must be dropped first. The AF PHBs at core routers use an active queue management technique such as Random Early Detection (RED) [23]. The are several variations of RED [24–26] to make the queue management scheme stable and fair to all flows. Braden *et al.* recommend queue managements technique to avoid congestion in the Internet.

To provide service differentiation, Clark and Fang introduced RED for IN and OUT of profile (RIO) packets [27]. The RIO algorithm (Figure 1.1) distinguishes between two types of packets, IN and OUT of profile, using two RED instances. Each RED instance is configured with min_{th}, max_{th}, and P_{max}. Suppose the parameters for the IN profile packets are min_{in}, max_{in}, and P_{max_in}, and for the OUT of profile packets are min_{out}, max_{out}, and P_{max_out}. To drop OUT packets earlier than IN packets, min_{out} is chosen smaller than min_{in}. The router drops OUT packets more aggressively by setting P_{max_out} higher than P_{max_in}. To realize three drop precedences, three REDs can be used.

1.3 Components to Support QoS and Security

The QoS networks support real-time data, bulk data, and statistically multiplexed data, which make the traffic management in the network hard. The necessary traffic management components to support QoS are:

- **Admission control.** The admission control component takes into account resource reservation requests and the available capacity to determine whether to accept a new request with its QoS requirements. Dynamic admission control with statistical bounds on each SLA parameter is an ongoing research issue. Admission control

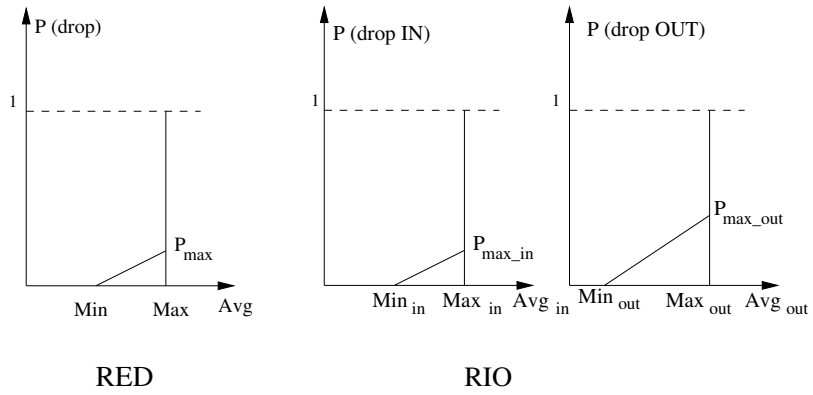


Fig. 1.1. Service differentiation using active queue management. RIO uses two instances of RED queues with different parameter settings.

needs to be done at the boundary of a network. It can be a part of an edge router or can be an individual entity in a network domain.

- **Monitoring.** The purpose of monitoring a network is to assess the network capacity for new users and observe the network for any unusual behavior. Misbehavior from fbw(s) may change the internal characteristics of the network. It is necessary to monitor a network domain for its proper operation. Existing monitoring schemes involve core routers to collect statistics. We need to devise an edge-to-edge, low overhead, and scalable monitoring scheme.

- **Policing/Shaping.** Users might send traffi c at a rate higher than the agreement. Policing is necessary to monitor these situations, and shaping makes the traffi c smooth and reduces its variations over time. Policing and shaping are done at the edge routers.

- **Congestion control.** Congestion control is required to avoid from happening anything bad inside a network domain. Some applications may not follow the standard protocol description and try to steal resources, thereby deteriorating the QoS of other applications. Mechanisms are needed to recover from congestion and control fbws accordingly.

- **Resource management.** QoS can be provided using over-provisioning of a network, which increases the cost incurred by the provider. Effi cient resource management is

a cost-effective solution for the provider, and it ensures that applications will get the specified QoS during the course of its execution.

- **Scheduling.** The scheduling component provides QoS by allocating resources depending on the service requirements. This requires mapping the user-defined QoS requirement to resource allocations for providing the service. An efficient and scalable scheduling mechanism is necessary at all routers to realize the QoS requirements into reality.

1.4 Proposed Research

In this research, we focus on components of an edge router to provide QoS and make a network domain safe to operate. The primary responsibility of an edge router is to meter, mark, and shape traffic. The edge router may control unresponsive flows, monitor the network for possible attack, and work as an bandwidth broker (BB) to sell bandwidth by dynamic provisioning. All components are necessary for a QoS-enable network domain. Some components can work as an individual entity. Scalable Edge-Based QoS for Intra-Domain Networks (SEQUIN) [28] is an ongoing project in Bell-Labs. The SEQUIN continuously measures the network and synthesizes this information to obtain actual edge-to-edge QoS. According to [28], these measurements provide a real-time view of network resource utilization. The edge-to-edge traffic metrics can be used for provisioning, service differentiation, SLA verification, and billing. Currently this project does not provide information about how to measure all edge-to-edge traffic metrics or the procedure of provisioning or SLA verification. We propose to design several components of scalable edge routers to secure a network domain. The router will monitor the network for service violation and bandwidth theft for security reason. The router improves the application level QoS and network resources utilization. The components of an edge router, the flow of traffic inside the router, and the flow of control packets are shown in Figure 1.2. Scalability poses a big challenge in the design of these components for the edge router. To achieve scalability, We propose not to put any overhead on the core routers to monitor the network and control flows. We ensure edge routers do not store per-flow information for all flows to provide QoS. The proposed components of an edge router ensure that:

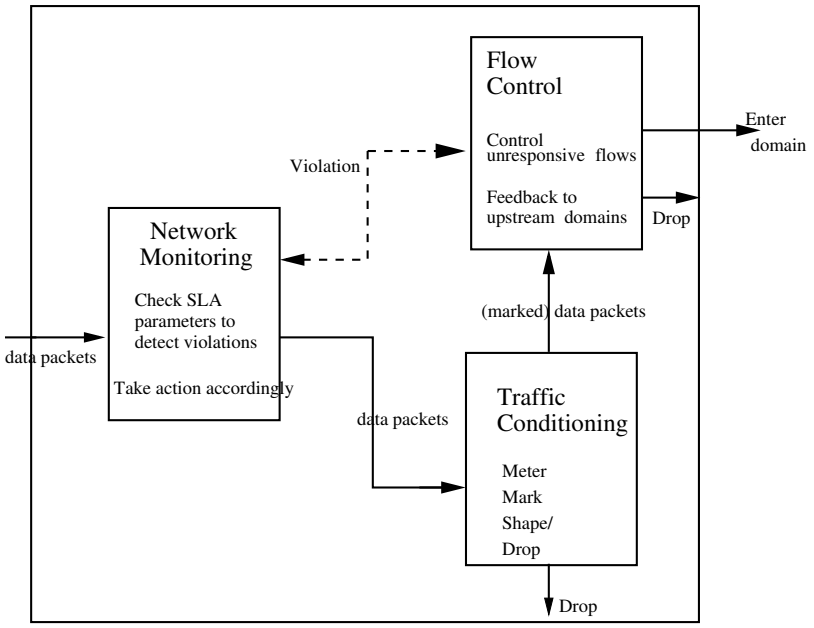


Fig. 1.2. Monitoring, conditioning, and flow control components inside an edge router to monitor and control a network domain.

- The provider protects the network from service violations, bandwidth theft, and denial of service (DoS) attacks. This is done by monitoring the flows that violate the SLA and cause other flows to suffer.

- The users applications receive specified throughput, delay, and loss requirements for each flow according to the service level agreement with the provider.

- Better utilization of resources by proper conditioning, controlling unresponsive flows, and sending feedback to the upstream routers to enforce policy and save wastage of resources by undelivered packets at the downstream routers.

We describe each component and the research questions associated with these components as follows:

SLA Monitoring. We need to monitor a network domain because some users may inject excessive traffic through several entry points even though at each entry point the traffic does not violate the service profile. This causes distributed denial of service attacks. Some users may inject traffic with a higher class than the appropriate one. This may cause other users to have low throughput, high delay, and high packet loss. We propose

to monitor the network to observe any incidence of high loss or delay based on the QoS parameters. The SLA monitoring component will perform service violation detection and bandwidth theft attacks. Monitoring network without the help of the core routers is a real challenge because measuring the loss ratio using edge-to-edge approach while keeping the communication overhead low is hard. We use the loss inference mechanism [29], network tomography [30, 31], and overlay networks to develop edge-to-edge network monitoring and SLA violation detection scheme.

Traffic Conditioner. The conditioner marks incoming traffic based on the current rate and the target rate of a flow. Marking provides a mechanism so that all flows can get their share. We study the behavior of transport protocols and use TCP characteristics to develop an adaptive and scalable conditioner. This adaptive design overcomes scalability problems arising from maintaining excessive per flow state. The edge router reduces the RTT-sensitivitiness on gaining bandwidth for TCP flows. The challenge related with traffic conditioner is to utilize flow characteristics to provide better QoS without using the per-flow information. This is necessary to achieve scalability.

Flow Control. Unresponsive flows do not react to congestion and continue sending packets with the same rate. The edge routers can be used to control the unresponsive flows and reduce the flow rate at the time of congestion. Congestion collapse can be mitigated using improved packet scheduling or active queue management [25, 32]. To solve the problem, we need a mechanism to ensure that the rate at which packets are entering into a network domain should be the same as the rate packets are leaving the domain. Dropping highest priority packets of each class exhibits that the network is congested [33]. We propose to use the DiffServ architecture and network tomography to address this issue. This congestion drop will be sent to the ingress routers to regulate unresponsive flows. The drops due to shaping at the ingress routers are propagated to egress routers of previous domain to regulate an unresponsive flow at the upstream path. We focus on exploring an alternative way to detect unresponsive flows without any help of core routers. We will use the flow aggregation characteristics with the control mechanism to cope with burstiness.

We provide analytic analysis of the proposed problems and conduct a series of experiments to show the behavior of the framework. Using simulation, we evaluate our edge router. Simulations are done to show that this router improves throughput of data extensive applications such as large FTP transfers, and achieves low packet delays and response times for Telnet and WWW traffic. We simulate DoS attacks and show that the proposed moni-

toring scheme can detect the attack with signifi cantly low communication and processing overhead.

1.5 Contribution of This Thesis

We design and implement the components of an edge router to monitor a network domain for any SLA violation and bandwidth theft attack. The network monitoring schemes involve only edge routers. The conditioner and flow control components alleviate the congestion and unfairness in the resources allocation problem. The edge routers share the congestion information with upstream routers to save resources wastage in the downstream domains. Designing a scalable edge router is a challenging research task because all of the components of an edge router should not use excessive per-flow information and can not involve core routers. This principle is followed in designing the components to achieve scalability. We solve the following research problems and evaluate the solutions using analytical models as well as a series of experiments.

- We defi ne and employ throughput, delay, packet loss, and security as QoS parameters for the design of an edge-to-edge SLA monitoring scheme to detect service violations and attacks. Our contribution of this work is *tomography-based* and *overlay-based* network monitoring methodologies for effi cient and scalable network monitoring. Both schemes use edge-to-edge measurements to infer the internal characteristics of a network domain. We provide probabilistic analysis about the performance of the monitoring algorithm. Our results show that we can monitor a network domain with $O(N)$ probes, where N is the number of edge routers. The monitoring mechanism can be used to detect denial of service (DoS) attacks at an early stage. A quantitative comparison among schemes to defeat DoS attacks is conducted, in which, we highlight the merits of each scheme and estimate the processing and communication overhead introduced by it. The comparison provides guidelines for selecting the appropriate scheme based on the requirements and how much overhead can be tolerated.

- The traffi c conditioners at the edge routers should intelligently mark and shape packets differentially based on the class parameters and network states. Our conditioner uses flow characteristics to provide better resource utilization and improve the application level QoS. We have designed an adaptive conditioner that considers congestion

window size, round trip time (RTT), retransmission time-out, and explicit congestion notification information to mark and shape a flow. Our results show that the conditioner improves throughput for data intensive applications and reduces the response time for delay sensitive applications. The conditioner improves the fairness among long and short-RTT flows.

- It is necessary to detect and regulate unresponsive flows that cause poor performance for adaptive flows, which retreats during congestion. We use the differentiated services framework and network tomography to detect the unresponsive flows. The flow detection mechanism is scalable and requires very low overhead. An adaptive congestion control framework is designed that follows TCP congestion control like algorithm and regulates the unresponsive flows to alleviate the congestion. The ingress (entry) edge routers propagate the congestion information to the egress (exit) routers of previous upstream network domain to reduce the resource wastage at the downstream network due to undelivered packets.

1.6 Organization of the Thesis

Chapter 2 discusses detecting service violation and bandwidth theft. This chapter introduces different monitoring techniques, and proposes edge-to-edge measurement-based distributed monitoring schemes. We provide the performance of each technique and the key differences among them.

Chapter 3 discusses different techniques to detect and prevent DoS attacks. Using an Internet-like collection of domains, techniques to defeat DoS attacks are investigated with their merits and demerits. Each technique is analyzed for its processing overhead and communication overhead. This comparison gives an insight and guidelines to choose a scheme based on the network size and the requirements.

Chapter 4 discusses traffic conditioner. The conditioner marks, shapes, and drops incoming traffic. However, an intelligent conditioning improves the resource utilization. In this chapter, an adaptive traffic conditioner is designed and implemented . The conditioner uses the flow characteristics to improve the QoS of a flow and ensures the fairness among different types of flows.

Chapter 5 discusses framework of controlling unresponsive flows. Two different flow control schemes are developed in this chapter. One technique takes the advantage of the

DiffServ framework with a small amount of feedback from the core routers. The other technique does not involve the core routers, and use network tomography to detect the unresponsive flows. An adaptive control mechanism is developed to control the misbehaving flows.

Chapter 6 concludes the thesis with a summary and future work.

2. NETWORK MONITORING TO DETECT SERVICE VIOLATIONS

Continuous monitoring of a network domain poses several challenges. First, routers of a network domain need to be polled periodically to collect statistics about monitoring parameters such as delay, loss, and throughput. Second, this huge amount of data has to be mined to obtain useful monitoring information. Polling increases the overhead for high speed core routers, and restricts the monitoring process from scaling to a large number of flows. To achieve scalability, polling and measurements that involve core routers should be avoided.

Monitoring schemes that observe the drop history at the core routers are referred to as core-assisted schemes in our work. The core-assisted approach is powerful because it can detect any attack and give precise information about which flows are misbehaving. The drop history at the core can help to detect flows that may launch a denial of service (DoS) attack in the downstream domain. Edge-to-edge measurement is, however, easier to deploy.

We use network tomography to develop *stripe-based* and *overlay-based* distributed monitoring schemes that use only edge-to-edge measurements, and scales to large network domains. The stripe-based monitoring scheme uses a series of back-to-back packets (referred to as a "stripe") to infer delay and loss of the internal links of a network domain. The link loss of each individual link can be calculated probing from any edge router to all other pair of edge routers. We extend stripe-based inference approaches to cope with different drop precedences in a QoS network. The overlay-based scheme further reduces the communication overhead. In this scheme, the edge routers form an overlay network with their neighboring edge routers. The overlay network is probed intelligently to identify the congested links with high losses in a domain. These links are used to identify the flows that are causing this high loss.

We compare the core-assisted scheme with the proposed edge-to-edge schemes. Simulation results indicate that the proposed monitoring schemes detect attacks and are useful for response and damage control in both QoS-enabled and best effort network domains.

2.1 Introduction

Monitoring of a *network domain* is necessary to ensure proper operation of the network by detecting possible service violations and attacks. Monitoring network activity is required to maintain confidence in the security and QoS of networks, from both the user (ensuring the service level paid for is indeed obtained) and provider (ensuring no unusual activity or attacks take place) perspectives. In this section, we describe existing and proposed monitoring techniques and tools.

2.1.1 Monitoring Techniques and Tools

A large variety of network monitoring tools can be found in [34]. Many tools use SNMP [35], RMON [36], or NetFlow [37], which are built-in functionality for most routers. Using these mechanisms, a centralized or decentralized model can be built to monitor a network. The centralized approach to monitor network latency, jitter, loss, throughput, or other QoS parameters suffers from scalability. One way to achieve scalability is to use a hierarchical architecture [38, 39]. Subramanyan *et al.* [39] design a SNMP-based distributed network monitoring system that organizes monitoring agents and managers in a hierarchical fashion. Both centralized or decentralized models obtain monitoring data by polling each router of a network domain, which limits the ability of a system to scale for a large number of flows. The alternative way of polling is to use an event reporting mechanism that sends useful information typically in a summarized format only when the status of a monitored element changes. A more flexible way of network monitoring is by using mobile agents [40] or programmable architecture [41]. However, periodic polling or deploying agents in high speed core routers put non-trivial overhead on them. Our goal is to design a low overhead and scalable monitoring scheme.

2.1.2 DiffServ Framework and QoS Attacks

We use the differentiated services (DiffServ) QoS framework as an underlying network, though our system is not specific to DiffServ. Packets entering into a DiffServ domain are classified and the DS field in the IP header is marked with three drop precedences (e.g., green, yellow and red) at the edge router. Typically, a user has a service level agreement with a provider that describes the expected service, user traffic profile, and charging models.

Differences in charging models of the service classes can attract attacks that inject marked packets to steal bandwidth and other network resources. Such attacks make use of known vulnerabilities in firewall filter rules to inject traffic or spoof the identity of valid users with high QoS levels. Since the DiffServ framework is based on aggregation of flows into service classes, valid user traffic may experience degraded QoS as a result of the injected traffic. Taken to an extreme, the attacks may result in denial of service. This creates a need for developing an effective defense mechanism that can automate the detection and reaction to attacks on the QoS-provisioned DiffServ network domain.

2.1.3 Proposed Monitoring Techniques

To detect attacks and service violations, we propose low overhead monitoring schemes that do not involve core routers for any kind of measurements. Our assumption is that if a network domain is properly provisioned and no user is misbehaving, the flows traversing through the domain should not experience high delay or high loss. An excessive traffic due to attacks changes the internal characteristics of a network domain. This change of internal characteristics is a key point to monitor a network domain. We employ agents on the edge routers of the DiffServ domain to efficiently measure packet delays, loss, and throughput. The SLA parameters such as delay, packet loss, and throughput are measured to ensure all users are getting their target share. The delay is an edge-to-edge latency measurement; packet loss is the ratio of total number of packets dropped from a flow to the total packets of the same flow entered into the domain; and throughput is the total bandwidth consumed by a flow inside a domain. A flow can be a micro flow with five tuples (two addresses, two ports, and protocol) or an aggregate one that is combined with several micro flows. Delay and loss are important parameters to monitor a network domain because these parameters mostly reflect the QoS of user applications. High delay and loss can be used as an indication of service violations. Although, jitter (delay variation) is another important SLA parameter, it is flow-specific and therefore, is not suitable to use in network monitoring.

Measurements are communicated to an SLA Monitor (SLAM). The SLAM analyzes measurements and automatically detects potential attacks and violations of negotiated SLAs, as well as flag the need to re-provision the network by increasing capacity or limiting users. Although measurement of path characteristics [42, 43] and network monitoring [44–46] have been extensively investigated, few studies of user SLA validation have been performed [47]. Inspired by recent results on network tomography [48–50], we infer internal

characteristics of a network domain using edge-to-edge probes, and design a distributed monitoring system to detect service violations and bandwidth theft in a network domain.

2.1.4 Our Contribution

In this chapter, we analyze core-assisted network monitoring scheme and develop efficient edge-to-edge measurement approaches for automatic detection of SLA violations, bandwidth theft, and denial of service attacks. We propose *stripe*-based and *overlay*-based monitoring schemes. In the stripe-based scheme, a series of probes are sent from one edge router to all other edge routers pairs. The congestion experience by successive packets in a stripe is assumed to be correlated. The packets have similar experience along the path on their journey to the destination. This correlation is used to infer the loss of each internal links using only edge-to-edge measurements. We extend stripe-based loss measurement approaches [29] to cope with different drop precedences in a QoS network. Throughput measurements are only performed when a delay or loss violation is reported.

We propose an overlay-based monitoring scheme that forms an overlay network using all edge routers on top of the physical network. The probing does not calculate loss ratio for each individual link, instead, the congested links due to high losses are identified using edge-to-edge loss measurements. Our solution consists of two methods: simple method and advanced method. In the simple method, all edge routers probe their neighbors in clockwise and counter-clockwise direction. This method requires only $O(n)$ probing, where n is the number of edge routers. Through extensive analysis, both analytical and experimental, we show that the simple method is very powerful to identify the congested links to a close approximation. If necessary, we use the advanced method to refine the solution of the simple method. The advanced method searches the topology tree intelligently for probes that can be used to identify the status of the undecided links from the simple method. When the network is less than 20% congested the advanced method requires $O(n)$ probes. If the congestion is high, it requires more probes, however, it does not exceed $O(n^2)$. The congested links are used as a basis to identify edge routers through which traffic are entering into and exiting from the domain. From exiting edge routers, we identify the flows that are violating any SLA agreement. If the SLA is violated for delay and loss, the network is probed to detect whether any user is stealing bandwidth. The service violations can indicate a possible attack on the same domain, or on a downstream domain.

Using simulation, we conduct a series of experiments to evaluate the proposed monitoring scheme. We conclude that the distributed monitoring scheme shows a promise for an efficient and scalable monitoring of a domain. This scheme can detect service violations, bandwidth theft attacks, and tell when many flows are aggregating towards a downstream domain for a possible DoS attack. The scheme requires low monitoring overhead, and detects service violations in both directions of any link in a network domain. We compare the core-assisted monitoring scheme with the proposed edge-to-edge schemes. The comparison can help network providers decide which technique best serves their needs.

2.2 Related Work

A number of studies have investigated differentiated services security, network monitoring, measurements of QoS parameters, loss inference, and SLA verification, as discussed next.

2.2.1 Network Security

A security analysis of Internet and of the DiffServ framework is provided in [51] and [52] respectively. The QoS attacks are classified into two kinds: attacking the *network provisioning process* and attacking the *data forwarding process*. Network provisioning involves configuration of DiffServ nodes by policy distribution points in the network, called Bandwidth Brokers (BBs). This is done through automatic signaling protocols such as RSVP [4] or SNMP [35, 53]. This process can be attacked by injecting bogus configuration messages, modifying the content of real configuration messages, delaying or dropping such messages. Networks can be secured against such attacks by employing encryption of the configuration messages of the signaling protocols. Attacks on the data forwarding process are of a more serious nature and can involve injecting traffic into the network with an intent to steal bandwidth or to cause QoS degradation by causing other customer flows to experience longer delays, higher loss rates, and lower throughput.

2.2.2 Network Monitoring

An Internet service provider needs to monitor its network domain to ensure the network is operating well. One obvious way of monitoring is to log packets at various points

throughout the network and then extract information to discover the path of any packet [54]. This scheme is useful to trace an attack long after the attack has been accomplished. The effectiveness of logging is limited by the huge storage requirements especially for high speed networks. Stone [55] suggested to create a virtual overlay network connecting all edge routers of a provider to reroute *interesting* flows through tunnels to central tracking routers. After examination, suspicious packets are dropped. This approach also requires a great amount of logging capacity.

Many proposals for network monitoring [44, 45] give designs to manage the network and ensure that the system is operating within desirable parameters. In efficient reactive monitoring [45], the authors discuss ways to monitor communication overhead in IP networks. Their idea is to combine global polling with local event driven reporting. Our core-assisted scheme depends on local event driven reporting to detect SLA violation and performs global polling only when it is absolutely necessary. Breitbart *et al.* [44] identify effective techniques to monitor bandwidth and latency in IP networks. The authors present *probing-based* techniques, where path latencies are measured by transmitting probes from a single point of control. The paper describes algorithms to compute an optimal set of probes to measure latency of paths in a network, whereas we focus on measuring parameters without the involvements of the core routers.

2.2.3 Measurements

A large body of research has focused on measuring delay, loss, and throughput in the Internet. Shared Passive Network Performance Discovery (SPAND) [56] is a tool that communicates with distant Internet hosts and reports to a performance server in the same domain. The clients query the server about the performance to a distant network site and obtain the history of average throughput of a TCP connection or the download time for a particular web page. The idea of measuring performance and sharing history to improve future measurement was proven useful. Savage *et al.* [57] propose *Detour* routers as edge devices in the Internet clouds that will tunnel traffic and improve the Internet performance. These edge routers exchange bandwidth, latency, drop rate among themselves. We also use intelligent routers at key access points that interchange information to improve the behavior of traffic management inside a network domain. Resilient Overlay Network (RON) [58] is an architecture to detect and recover from path outages and periods of degraded performance. The RON nodes monitor the quality of Internet paths among themselves and use

this information to route packets, optimizing application-specific routing metrics. RON uses three different routing metrics: latency, loss and throughput. The latency is an Exponential Weighted Moving Average (EWMA) with 10% weight for the most recent sample. The loss is an average of 100 probe samples. Measurement techniques in SPAND, Detour, and RON have been proven to be useful, which are utilized in our work.

2.2.4 Inference

Duffield *et al.* [59] propose trajectory sampling to infer traffic flows through a domain. In this process, each link samples packets based on a hash function computed over the content of the packets. Then, the trajectory of a packet is reconstructed using the same sample set of packets. This provides a neat way of monitoring a network domain, which does not depend on the network status information. However, all routers (edge and core) participate in sampling that might put large overhead on the high speed core routers. Our goal is to devise a low overhead scheme that does not involve the core routers for any measurement.

Duffield *et al.* [29] use packet "stripes" (back-to-back probe packets) to infer link loss and delay by computing the correlation of packet loss and delay within a stripe at the destinations. Using end-to-end unicast probing, the authors demonstrate how to infer loss characteristics of the links in the network interior. This work is an extension of loss inference for multicast traffic described in [48, 50]. To detect SLA violations, we need to compare monitored flow parameters to the SLA parameters. Therefore, we must monitor all flows of a user and calculate loss information of all links the user flows traverse. We develop an efficient and low overhead method to detect delay, loss and throughput on a per user basis.

2.2.5 SLA Verification

In [47], a histogram-based aggregation algorithm is used to detect SLA violations. The algorithm measures network characteristics on a hop-by-hop basis and uses them to compute end-to-end measurements and validate end-to-end SLA requirements. In a large networks, efficient collection of management data is a challenge. While exhaustive data collection yields a complete picture, there is an added overhead. The authors propose an *aggregation* and *refinement* based monitoring approach. The approach assumes that the routes used by SLA flows are known, citing VPN and MPLS [7] provisioning. Though

routes are known for double ended SLAs that specify both ingress and egress points in the network, they are unknown in cases where the scope of the service is not limited to a fixed egress point. As with RONs [58], we check violations using average values in a recent time frame. This reduces constraints on the network setup and need for knowledge of the flows traversing through each router.

2.3 Architecture for SLA Violation Detection

The DiffServ architecture [60] achieves scalability by pushing complexity to boundary devices, which process lower volumes of traffic and smaller numbers of flows. Ingress routers perform complex traffic conditioning that consists of traffic classification based on multiple fields in the packet header, traffic metering to ensure conformance to a profile, DSCP marking, and dropping, shaping or remarking of out-of-profile traffic. Core routers perform simple forwarding based on the DSCP. SLAs between the customer and provider networks are used to derive filter rules for traffic classification at the ingress routers. Therefore, ingress routers with appropriate configuration of filter rules should prevent non-conforming traffic from entering a DiffServ domain.

Though ingress routers serve as a good first line of defense, attackers can still succeed in injecting non-conforming traffic into a DiffServ domain in a variety of ways, e.g.:

1. Attackers can impersonate a legitimate customer by spoofing flow identity(IP addresses, protocol and port numbers). Network filtering [61] at routers in the customer network can detect such spoofing if the attacker and the impersonated customer are on different subnets, but the attacks proceed unnoticed otherwise.

2. Attackers can devise mechanisms to bypass the ingress routers by exploiting some well known vulnerabilities in the firewall filters. Thus, they can inject traffic with their own identity and a desired destination.

Such intelligent attacks escape detection at the ingress router and succeed in injecting traffic into the DiffServ domain. They require co-ordination between boundary routers or the support of core routers for detection. Changes that can be observed due to the attack traffic in the network include longer per-packet delays, higher average buffer occupancy, and higher packet drop rates. We use these characteristics, specifically delays, loss ratios, and bandwidth achieved by flows *after* aggregation within the domain to detect bandwidth theft attacks and violations.

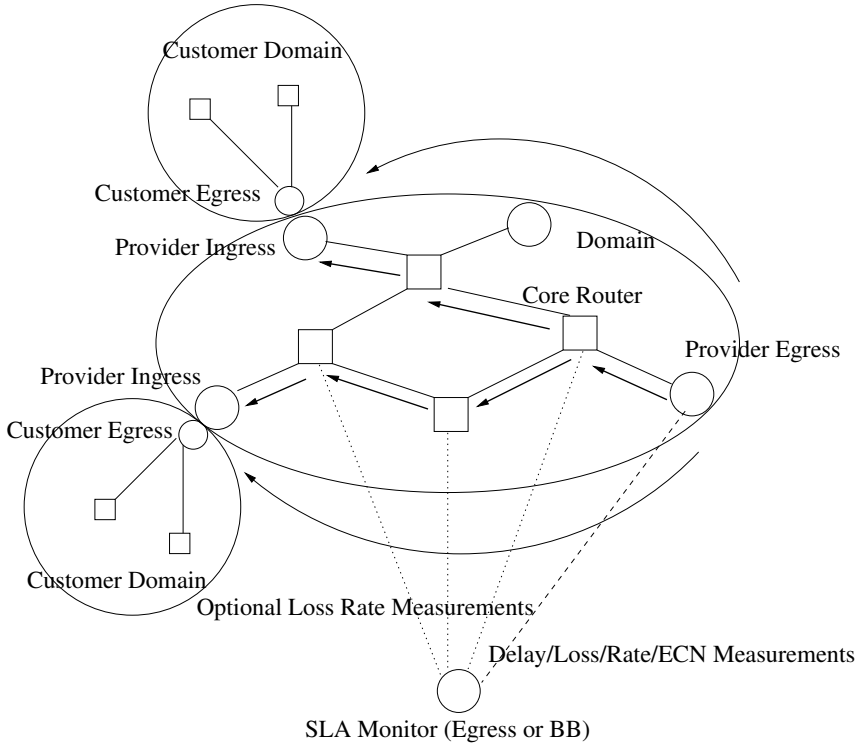


Fig. 2.1. A hybrid architecture for detecting SLA violations. This can be used for core-assisted as well as edge-to-edge schemes. The SLAM gets feedback from core routers in core-assisted scheme only. In both cases, SLAM sits on an edge router.

Figure 2.1 shows a hybrid architecture for detecting SLA violations. We use an SLAM to monitor the DiffServ domain. The SLAM gets feedback about delay, loss, and other parameters from edge and core routers depending on the monitoring scheme we use. The core routers send information to the SLAM only in core-assisted scheme. In this fi gure, the SLAM is shown as a separate entity. However, any edge router can take this responsibility as long as it has suffi cient resources and computing capabilities. A service provider can also use a dedicated host as a SLAM.

The SLAM maintains a table storing delay and loss information of misbehaving fbws only. The table is updated on receipt of new values from the egress and core routers. In addition, the SLAM maintains the SLA parameters for each customer for a certain domain. By comparing the delay and loss measurements against the specifi c customer SLA, the violations are identifi ed. In the core-assisted scheme, egress and core routers send delay and loss measurements respectively to the SLAM. Upon request, the ingress routers send

the number of packets entering a domain per flow to calculate loss ratio. The packet loss is computed as the ratio of the packet drop inside a domain to the total packets entering the domain. The loss ratio of a flow is a better metric than loss rate or number of drops per second. Another alternative is to measure delay, loss or throughput using only edge routers. The main difference between the edge-to-edge and the core-assisted approaches from an architectural point of view is that both use edge routers but core-assisted measurement use core routers as well.

2.4 Core-assisted Monitoring

The core-assisted monitoring estimates the SLA parameters with the involvement of the core routers in a network domain. These measurements are similar to the literature [42, 43] on measuring delay, loss, and throughput in the Internet. However, we follow different strategies to measure each parameter to make it more suitable for the monitoring process. In this section, we discuss the measurement of SLA parameters and how these measurements are used to monitor a network domain.

2.4.1 Delay Measurements

Delay bound guarantees made by a provider network to customer traffic flows are for the delays experienced by the flows between the ingress and egress edges of the providers domain. Delay measurements can be done using either real customer traffic or artificially injected traffic. The first is an*intrusive* approach and is difficult to implement because encoding timestamps into the data packets would require changing the packets at the ingress and rewriting the original content at the egress after appropriate measurements. The second approach is *non-intrusive*. Probe packets are injected with desired control information to enable an egress router to recognize such probes, perform measurements, and delete the probes from the traffic stream. We adopt the second approach in our design. For each packet traversing an ingress router, with a certain pre-configured probability p_{probe}, the ingress copies the packet IP header into a new probe packet. A timestamp $t_{ingress}$ is encoded into the payload, and an identifier field is marked with a new value in the probe packet. The egress router removes probes from the traffic stream, and computes delay for a packet of flow i as:

$$delay^i = t^i_{egress} - t^i_{ingress}. \tag{2.1}$$

The egress forwards the packet details and the measured delay information to the SLAM. The encoded timestamp must follow a well-known format such as the Coordinated Universal Time (UTC), or a standard protocol such as Network Time Protocol (NTP) to obtain the timestamp value at the edge routers. The clock synchronization problem can also be overcome by rerouting the same probe packet from egress to ingress router using strict source routing in the opposite direction. The ingress router then computes the approximate latency from the elapsed time between sending and receiving the probe. At time t, the SLAM classifies the packet as belonging to flow i of customer j, and updates the average packet delay of the customer traffic as an exponential weighted moving average (EWMA):

$$avg\_delay_j(t) = \alpha \times avg\_delay_j(t-1) + (1-\alpha) \times delay^j_i(t), \tag{2.2}$$

where α is a small fraction to emphasize the delay history more than the current sample. If this average packet delay exceeds the delay guarantee in the SLA, we conclude that an SLA violation has occurred. If the network is properly provisioned and all flows do not misbehave, delay for customer j should not exceed its delay guarantee.

Determining the probability with which we should inject probe packets is not an easy task. If there are M edge routers in a network domain, N^i flows (on the average) passing through an edge router i, and p^{ij}_{probe} is the probability that an edge router i and flow j will be selected to probe for latency, then $MN^ip^{ij}_{probe}$ is the average number of control/probe packets injected into the network domain. To keep the volume of these control messages low, we must select a low probability. However, if the probability is too low, the chance of undetected SLA violations is higher. Therefore, we choose a variable p_{probe} that changes dynamically over time at each edge router. The change in this probability is performed at all edge routers autonomously making sure the edges do not use the same random number generator sequence or seed.

2.4.2 Loss Measurements

Packet loss guarantees made by a provider network to a customer are for the packet losses experienced by its conforming traffic inside the provider domain. Since packet

losses in the domain are mainly due to packet drops at core routers, the buffer management schemes at the core router queues are used to detect such losses. This can detect the number of packets dropped more easily than the loss ratio, though, loss ratio is more meaningful in SLAs than the number of packet drops or the loss rate. For loss ratio, we need the number of packet drops, as well as the number of packets traversing through the core routers.

In the core-assisted loss measurement, packet drops are recorded for every flow over a time interval Δt at the core routers. The time interval is usually small enough to store count information in an integer variable without overflow. The measured values are reported to the SLAM. At time t, the SLAM then calculates an EWMA using

$$L^i(t) = \alpha \times L^i(t-1) + (1-\alpha) \times drop(t); 0.1 \leq \alpha \leq 0.2, \tag{2.3}$$

where $drop(t)$ is the total packet drop at core router i over time interval Δt. We typically give a low weight to the recent sample. The weighted average solves the wrap-around problem of total packets count during the life time of a flow.

To obtain the loss ratio, we must measure the number of packets dropped and the total number of flow packets traversing each core router. Measuring the total number of incoming packets at core routers and periodically reporting them to the SLAM is expensive. Incoming packet count information is obtained from the edge routers because they monitor all flows for profile checking, marking, shaping, and/or dropping. This ensures that cores do not need to transmit information to the SLAM unless there are sufficient packet drops to indicate attacks or violations. We synchronize the intervals by adopting a standard protocol such as NTP for time. The algorithm to compute the loss ratio, executed every Δt, is shown in Figure 2.2.

Proposition 2.4.1 *Algorithm CalcLossRatio (Figure 2.2) computes a close approximation of the loss ratio in a network domain.*

Proof This is because at time t, the SLAM receives drop D_i^j from core i. The SLAM computes total drop D^j for flow j over a time interval W seconds. If it takes Δt_1 seconds for drop information to reach to the SLAM from the core, then D^j reflects drop within $t - \Delta t_1$ time. The SLAM queries edge routers for incoming packet count of user j, B^j. In a differentiated services network, the ingress routers calculate the average incoming rate over a time interval for the purposes of marking and shaping. Edge routers easily reply to the

Algorithm: *CalcLossRatio()*

1. Core i reports to the SLAM whenever packet drop of flow j, D_i^j, exceeds the local threshold T_l^{ij}.

2. The SLAM computes the total drop during time interval Δt, $D^j = \sum_{i=1}^{N_c} D_j^i$, where N_c is number of core routers.

3. If total drops exceed a global threshold, $D^j > T^j$,

a. The SLAM sends a query to all edge routers requesting their current throughput, B^j for flow j

b. The SLAM calculates total incoming rate for flow j, $B^j = \sum_{i=1}^{N_e} B^{ij}$, where N_e is number of edges.

c. The SLAM computes the loss ratio $LR^j = \frac{D^j}{B^j}$. Note that both B^j and D^j are measured over the same time interval.

d. if $LR^j > LR_{SLA}^j$, an SLA violation is reported.

Fig. 2.2. Algorithm to calculate the loss ratio for the core-assisted monitoring scheme. The loss ratio is used to detect SLA violations.

SLAM query. Assume it takes Δt_2 seconds to send the query from the SLAM to the edges. The actual loss ratio is $LR^j = \frac{D^j}{B^j - \Delta B^j}$, and the approximate loss ratio is $LR_{approx}^j = \frac{D^j}{B^j}$, where ΔB^j is the number of packets arriving during Δt_2. If the SLAM and edge routers use the same time interval W to calculate drop and if $\Delta t \ll W$, then $\Delta B^j \ll B^j$, i.e., $LR^j \approx LR_{approx}^j$. ■

Selecting the local threshold value at each core router is difficult since the core router does not have any information about the user SLAs. The thresholds must be set on per-class basis. For each AF class k, the router needs a local loss threshold T_l^k. May *et al.* [62] compute the loss probability for an AF class k as follows:

$$LR^k = 1 - \sum_{n=0}^{B} \alpha^k(n)\pi(n), \tag{2.4}$$

where B is the buffer size of the queue, $\alpha^k(n)$ represents the probability that a class-k packet is accepted given that n packets are in the queue, and $\pi(n)$ is the stationary distribu-

tion of the buffer content. The local threshold at each core for class-k packets can thus be set to $T_l^k = (LR^k + \Delta)\lambda^j$, and $LR_{SLA}^k = (LR^k + \Delta)H$, where λ^j is the expected arrival rate of user j and H is the maximum number of hops a packet travels through the network domain.

The local threshold limits the state maintenance at the core to a subset of the total number of flows experiencing the highest loss ratio, since we are only interested in flows that result in the aggregate traffic experiencing high loss ratio. If a flow exhibits a high loss ratio, this does not mean that this particular flow is violating its profile. This drop may be caused by other misbehaving flows. Comparing aggregate throughput to the allowed rates of users can detect such attacks.

2.4.3 Throughput Measurements

The objective of throughput measurement is to ensure no user is consuming extra bandwidth (beyond its profile) after aggregation within a domain. This cannot be detected by an ingress router if the user sends at a lower rate than its profile through multiple ingress routers. The service provider allows the user to consume extra bandwidth if no other flow suffers degraded performance as a result, so throughput measurement is only performed if a violation has been reported.

The SLAM measures throughput by probing egress routers following a loss or delay violation report. The egress measures the rate at which user traffic is leaving a network domain. This is an average value over a time duration and it represents the per-domain throughput for a flow. When the SLAM measures throughput of flows at egress routers, it computes the aggregate throughput for user j as: $B^j = \sum_{i=1}^{N_e} B^{ij}$. If $B^j > SLA_{bw}^j$ then an SLA violation is reported.

2.4.4 Monitoring Algorithm

Let E be the set of all edge routers, both egress and ingress. One of these routers can act as a SLA Monitor (SLAM), or a separate entity can be used to act as SLAM. The algorithm proceeds as follows:

1. Each ingress router copies the header of user packets with probability p_{probe} to probe the network for delay measurement.

2. When an egress router receives these probes, the egress stamps its identity on the packet and resends them back to the source (ingress). The egress replies the delay probe to the ingress because by this way the monitor determines the receiver egress routers for the probing.

3. The monitor computes the average edge-to-edge delay and updates the average delay of the user using equation 2.2.

4. During congestion, the SLAM receives packet drop information from different core routers. Then, it polls the egress routers to obtain the packet count, which is required to calculate the loss ratio. The loss ratio is calculated using algorithm shown in Figure 2.2.

5. The SLAM probes the network for throughput approximation when the loss is higher than the pre-confi gured threshold.

6. For users with higher delay, loss, and bandwidth consumption, the monitor decides about possible SLA violation. The monitor knows the existing traffi c classes and the acceptable SLA parameters per class. If there is any loss at the core for the EF traffi c class and if the AF loss ratio exceeds a certain level, an SLA violation is flagged. DoS attack is checked by comparing the total bandwidth achieved of a user with it SLA_{bw}. For each violation, it takes proper action, such as throttling the particular user traffi c using flow control mechanism.

2.4.5 Experiments: Delay, Loss, and Throughput Measurements

Setup. The performance of our monitoring mechanism is evaluated using simulation. A series of experiments are conducted to investigate the delay, loss, and throughput approximation methods described in this section. We use a similar network topology that is used in [29] to evaluate our monitoring schemes. The topology is shown in Figure 2.3. Multiple domains are connected to all edges routers through which flows enter into the network domain. The flows are created from domains attached to $E1$, $E2$, $E3$ and destined to the domains connected to edge router $E6$ so that the link $C4 \rightarrow E6$ is congested. An attack is simulated on $C4 \rightarrow E6$ to show that the edge routers can detect service violations and attacks due to flow aggregation towards a downstream domain. Many other flows are created to ensure all links carry a signifi cant number of flows.

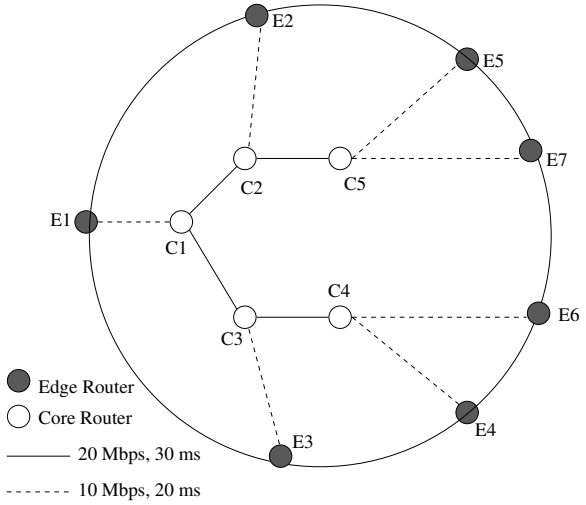
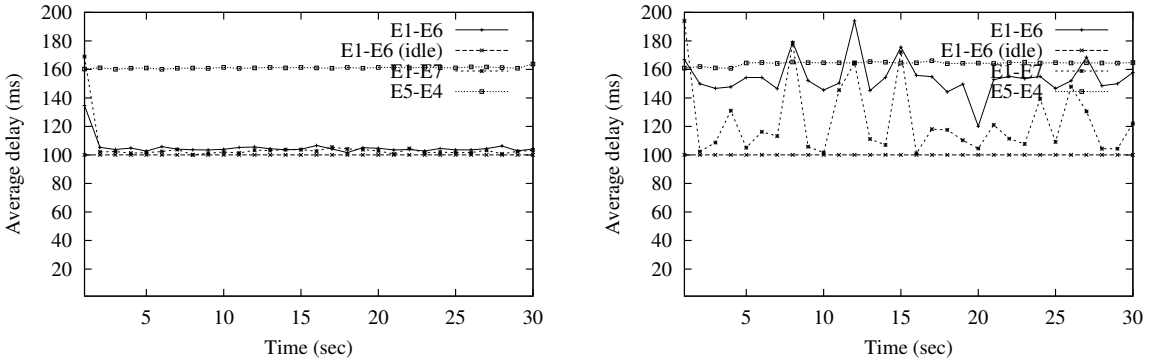


Fig. 2.3. Topology used to detect service violations. All edge routers are connected to multiple domains, and each domain has multiple hosts to act as senders or receivers.



(a) No Attack

(b) With high excess traffi c

Fig. 2.4. Edge-to-edge link delay when links are idle and during an attack. (a) Delay when the network is not under attack and most of the links are idle. (b) The delay goes high in presence of attacks.

Delay measurements. We measure delay when the network is properly provisioned or over-provisioned (and thus experiences little loss). An attack is simulated through links $C3 \rightarrow C4$ and $C4 \rightarrow E6$ to show how delay gets changed on these links during attacks. This scenario is illustrated in Figure 2.4. When idle, the end-to-end delay of $E1 \rightarrow E6$ link is 100 ms; $E1 \rightarrow E7$ delay is 100 ms; and $E5 \rightarrow E4$ delay is 160 ms. With the attack

traffic, the average delay of the $E1 \rightarrow E6$ link is increased to as high as 180 ms (Figure 2.4(b)). Since all the core router $\rightarrow$ core router links have a higher capacity than others, $C4 \rightarrow E6$ becomes the most congested link and increases the delay for all traffic traversing to $E6$. The delay of the $E5 \rightarrow E4$ link is not increased significantly because this path is not under attack.

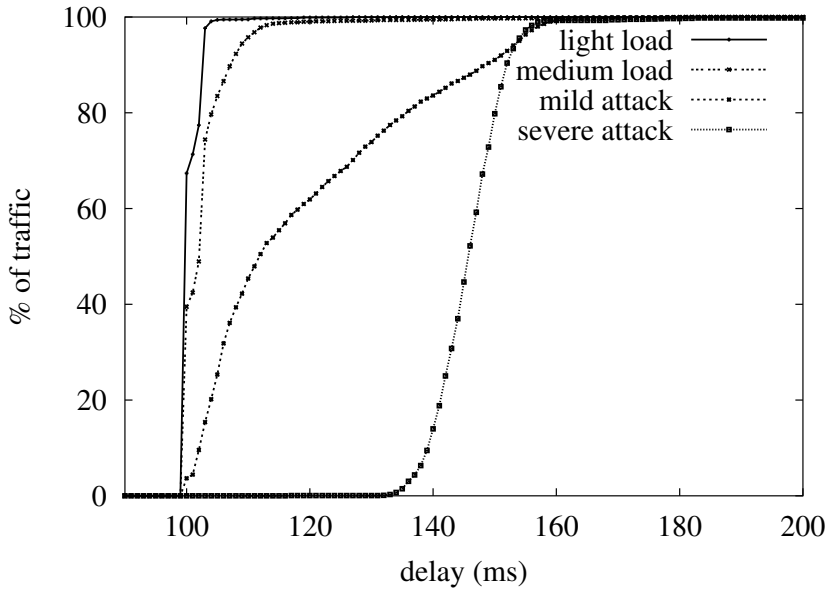
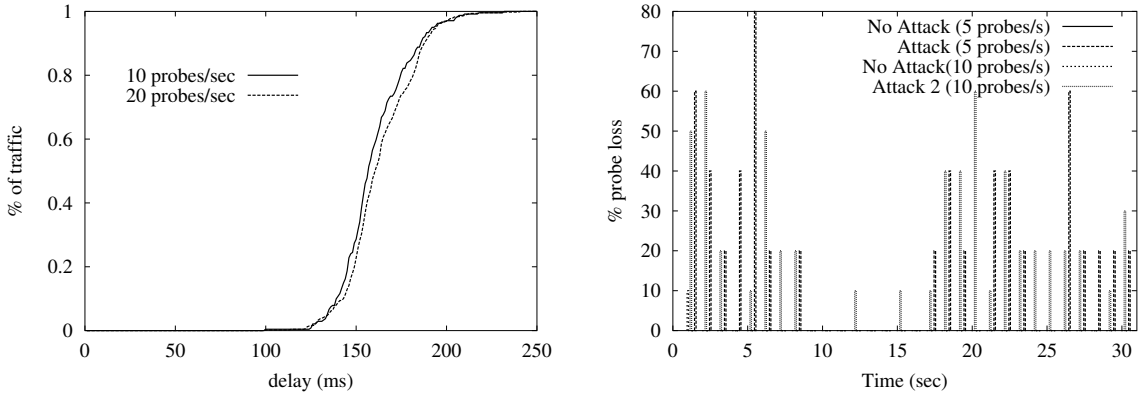


Fig. 2.5. Cumulative distribution function (CDF) of edge-to-edge delay for link $E1 \rightarrow E6$. The delay changes with network traffic load.

Figure 2.5 shows the cumulative distribution function (CDF) of edge-to-edge delay for the link $E1 \rightarrow E6$ under various traffic loads and in presence of attacks. When there is no attack, the end-to-end delay is close to the link transmission delay. If the network path $E1 \rightarrow E6$ is lightly loaded, for example with a 30% load, the delay does not go significantly higher than the link transmission delay. Even when the path is 60% loaded (medium load in Figure 2.5), the edge-to-edge delay of link $E1 \rightarrow E6$ is increased by less than 30%. Some instantaneous values of delay go as high as 50% of the link transmission delay but the average value does not fluctuate a lot. In both cases, the network is properly provisioned, i.e., the flows do not violate the SLAs. On the other hand, an excess traffic introduced by an attacker increases the edge-to-edge delay and most of the packets of attack traffic experience a delay 40-70% higher (Figure 2.5) than the link transmission delay.

Delay measurement is thus a good indication of the presence of excess traffic inside a network domain.



(a) Delay introduced by probes

(b) Probe loss ratio

Fig. 2.6. (a) Link delay is slightly changed when more probes are introduced. (b) Probing at a low rate suffers loss when excess traffic is introduced by an attacker. This can introduce incorrect experimental results.

Figure 2.6(a) shows that introducing more delay probes may increase the delay of actual traffic. The graph of delay probe loss ratio in Figure 2.6 (b) shows that sending only 5 probes per second can be dangerous because as high as 80% of the probes are lost in this experiment. We find that sending probes at a rate 10-15 per second is a good choice in this experiment.

Loss measurements. Figure 2.7 shows loss approximation using the core-assisted scheme. As the scheme uses an EWMA of the drop values and the number of incoming packets traversing edge routers, the initial approximated values deviate from the actual value. Thus, initial data should be discarded. After the initial periods, the approximated value is very close to the actual one. The approximated loss ratio is reasonably converges to the actual loss ratio within 5 seconds. According to the simulation setup, link $C4 \rightarrow E6$ sees a large number of losses, resulting in an increased loss ratio for the $E1 \rightarrow E6$ as shown in Figure 2.7.

Throughput measurements. Figure 2.8 shows the throughput approximation that computes the average bytes at the egress routers of different flows traversing the network domain. There are several aggregate flows going through the domain. We measure through-

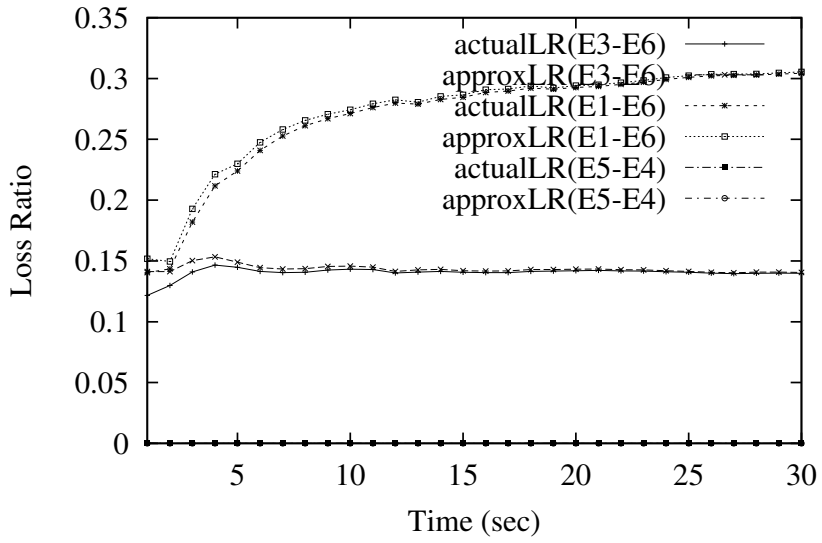


Fig. 2.7. Loss approximation with the core-assisted scheme. The approximated value converges to the actual loss in 5 sec.

put for flow $F1$, which follows the path $E3 \rightarrow E6$, flow $F2$ which follows path $E1 \rightarrow E6$, flow $F3$ which follows path $E2 \rightarrow E6$, and flow $F4$ which follows path $E5 \rightarrow E4$. Other aggregate flows follow paths $E1 \rightarrow E7$ and $E3 \rightarrow E7$. Figure 2.8 shows an initial fluctuation between actual and approximated throughput measurements due to the average calculation. After a few seconds, the values are close together. As the approximation is close with a small error, it is possible to sum up user throughput and compare it with the overall SLA profile to detect violations.

2.5 Stripe-based Monitoring

This scheme uses the same mechanism to measure delay and throughput as it is described in the core-assisted monitoring scheme. The main difference between the two lies in measuring loss. The stripe-based monitoring scheme probes a network domain to infer loss characteristics of each individual links. The loss inference makes the stripe-based monitoring scalable.

The real challenge of this work is to determine an algorithm that monitors the network in real time nature. We need to determine how often the stripes should be sent and to which receivers to monitor all links. As this mechanism involves only edge routers, the monitoring scheme will be scalable.

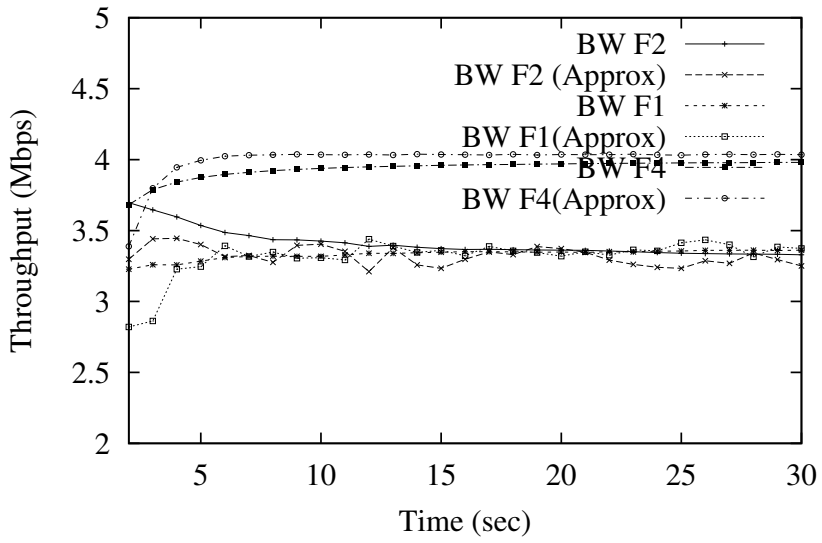


Fig. 2.8. Throughput approximation using core-assisted monitoring scheme. The approximated values converge to the actual values within 10 seconds.

2.5.1 Loss Inference

We show how to infer loss ratios for unicast traffi c as explained in [29]. This scheme sends a series of probe packets with no delay between the transmission of successive packets, or what is known as a "stripe." The scheme is designed as an end-to-end scheme, which can be adapted to the edge-to-edge scenario. We refer to this strategy as the *stripe-based* loss measurement scheme.

To simplify the discussion, consider a two-leaf binary tree spanning nodes 0, k, R_1, R_2, as shown in Figure 2.9. We have used the same fi gure as it is shown in [29] to explain their idea. The loss ratio of the link $k \rightarrow R_1$, for instance, can be estimated by sending stripes from the root 0 to the leaves R_1 and R_2. The fi rst packet of a 3-packet stripe is sent to R_1, while the last two are sent to R_2. If a packet reaches to any receiver, we can infer that the packet must have reached the branching point k. Further, if R_2 gets the last two packets of a stripe, it is likely that R_1 receives the fi rst packet of that stripe. The packet loss probability is calculated based on whether all packets sent to R_1 and R_2 reach their destination. Similarly, the loss ratio of the link $k \rightarrow R_2$ is inferred using a complementary stripe, in which the fi rst packet is sent to R_2 and the last two are sent to R_1. The loss ratio of the common path from $0 \rightarrow k$ from the transmission probability as shown below:

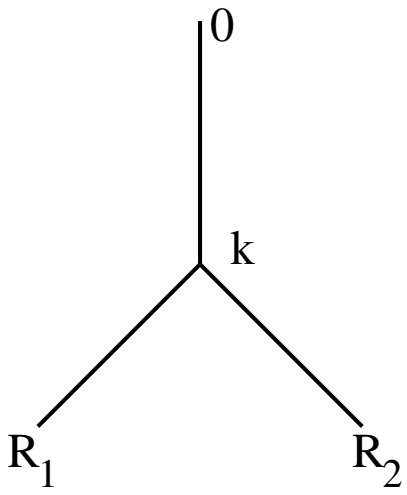


Fig. 2.9. Binary tree to infer loss of each link. The probes are sent from the sender 0 to both receivers R_1 and R_2.

$$A_k = \frac{Z_{R_1} Z_{R_2}}{Z_{R_1 \cup R_2}}, \tag{2.5}$$

where Z represents the empirical mean of a binary variable which takes 1 when all packets sent to R_1 reach their destination and 0 otherwise. The mean is taken over n identical stripes. By combining estimates of stripes down each such tree, the characteristics of the common path from $0 \rightarrow k$ is estimated.

This inference technique extends to general trees. Consider an arbitrary tree where for each node k, $R(k)$ denotes the subset of leaves descended from k. Let $Q(k)$ denote the set of ordered pairs of nodes in $R(k)$ descended from k. For each $(R1, R2) \in Q(k)$, a stripe should be sent from the root to the receivers R_1 and R_2.

2.5.2 Loss Inference for Active Queues

The unicast probing scheme is extended for routers with active queue management, e.g., 3-color RED [23]. This queue has different drop precedences. We have to adjust the loss inference to cope with the drop precedences. The new scheme will be used to monitor loss inside a QoS network domain.

Figure 2.10 shows the drop probabilities of a three drop precedence active queues. The red traffi c has higher probability to drop than yellow and green traffi c. The packets are

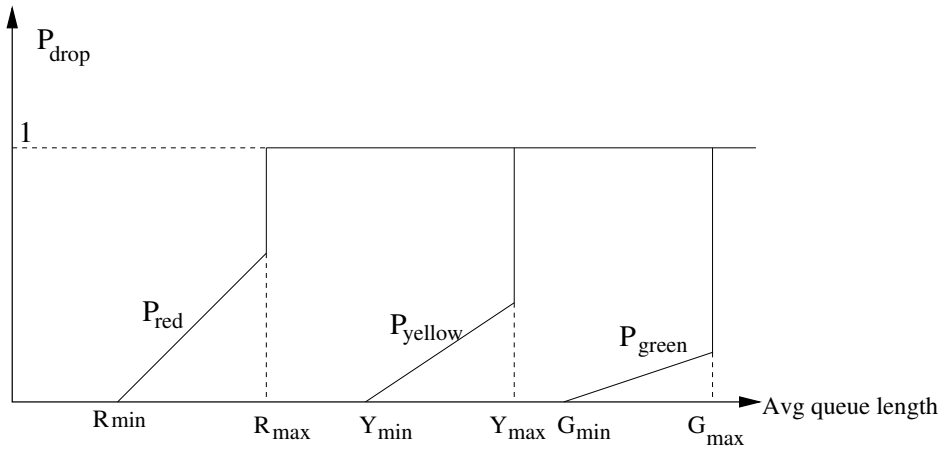


Fig. 2.10. RED Parameters for an active queue with three drop precedences.

dropped with these probabilities when the queue length falls in between minimum and maximum threshold. Below minimum threshold there is no drop, and after maximum threshold there is absolute drop. Let, $\alpha_G(n)$ be the probability that an incoming green packet will be accepted by the queue given that n packets are in the queue. $\alpha_Y(n)$ and $\alpha_R(n)$ have the same meaning for yellow and red traffi c respectively. We write equations ofαs shown below:

$$\begin{aligned} \alpha_G(n) &= 1, && if \quad n < G_{min} \\ \alpha_G(n) &= 0, && if \quad n > G_{max} \\ \alpha_G(n) &= 1 - P_{green}\frac{n - G_{min}}{G_{max} - G_{min}}, && \text{otherwise.} \end{aligned} \tag{2.6}$$

The equations are similar for yellow and red traffi c. These equations help to understand the behavior of active queues. Based on this information, we need to change the loss inference mechanism. The Assured Forwarding (AF) mechanism is realized using four queues where each queue has three drop precedences referred to as green, yellow, and red. The red traffi c is dropped with a probability P_{red} when the average queue size lies between two thresholds R_{min} and R_{max}. All incoming red packets are dropped when the average queue length is $\geq R_{max}$. Let $\mathcal{P}_{red}$ be the percentage of packet drops due to the behavior of active queue management for red packets, and let $\mathcal{P}_{yellow}$ and $\mathcal{P}_{green}$ be defi ned similarly. These percentages can be computed as:

$$\mathcal{P}_{red} = \frac{R_{max} - R_{min}}{R_{max}} \times P_{red} + \frac{G_{max} - R_{max}}{B} \times 100, \tag{2.7}$$

$$\mathcal{P}_{yellow} = \frac{Y_{max} - Y_{min}}{Y_{max}} \times P_{yellow} + \frac{G_{max} - Y_{max}}{B} \times 100, \tag{2.8}$$

$$\mathcal{P}_{green} = \frac{G_{max} - G_{min}}{G_{max}} \times P_{green}, \tag{2.9}$$

where B is the buffer (queue) size in the router.

Let, $\mathcal{P}'_{red} = 1 - \mathcal{P}_{red}$, be the percentage of *red* packets accepted by the active queue. We can defi ne percentages for yellow and green traffi c similarly using equations (2.8) and (2.9). Link loss can be inferred by subtracting of transmission probability of equation (2.5) from 1 i.e. $1 - A_k$. Therefore, if L_g, L_y, and L_r are the inferred losses of green, yellow and red traffi c, respectively, the overall loss of a class is expressed as shown in equation (2.10), where n_i is number of samples taken from i types of traffi c. To put equal weight to all traffi c, we can use $n = n_i, \forall i$. However, when loss of green traffi c is zero, we average yellow and red losses. When the loss of yellow traffi c is zero, we report only loss of red probes.

$$L_{class} = \frac{n_g \mathcal{P}'_{green} L_g + n_y \mathcal{P}'_{yellow} L_y + n_r \mathcal{P}'_{red} L_r}{n_g + n_y + n_r}. \tag{2.10}$$

Our goal is to detect SLA violations with minimal communication and implementation overhead. To achieve that, we propose to eliminate unnecessary probing when there is no traffi c or when there are no misbehaving fbws.

The essence of the proposed approach is to detect which egress and ingress routers are active at a certain time. This can be done using delay probes, which are anyway periodically transmitted. If many edge routers are idle or many links are under-utilized, the SLAM does not probe the whole network for loss information. This reduces the set of edge routers used as receivers in the stripe-based probing.

2.5.3 Monitoring Algorithm

Let E be the set of all edge routers, both egress and ingress. One of these routers can act as a SLA Monitor (SLAM), or a separate entity can be used to act as SLAM. The algorithm proceeds as follows:

1. Each ingress router copies the header of user packets with probability p_{probe} to probe the network for delay measurement.

2. The monitor computes the average edge-to-edge delay updates the average delay of the user using equation 2.2. If the delay exceeds a certain threshold, the monitor needs to probe the network for loss.

3. The SLAM maintains a set of edge routers E' to send stripes to in order to infer loss, where $E' \subseteq E$. The SLAM maintains a spanning tree of the network topology and the minimum set of edges S that needs to be probed to infer loss on active links. A set of edge routers, S^i, which we refer to as complementary edges, is associated with each edge i. At time t, the SLAM computes the set E' as:

$$E'_t = \bigcup_i S^i_t. \tag{2.11}$$

   Since $E' \subseteq E$, the communication overhead will be no more than that of the regular probing to all edge routers.

4. The SLAM probes the network for throughput approximation when the loss is higher than the pre-confi gured threshold.

5. For users with higher delay, loss, and bandwidth consumption, the monitor decides about possible SLA violation. The monitor knows the existing traffi c classes and the acceptable SLA parameters per class. If there is any loss at the core for the EF traffi c class and if the AF loss ratio exceeds a certain level, an SLA violation is flagged. The DoS attack is checked by comparing the total bandwidth achieved by a user with it SLA_{bw}. For each violation, it takes proper action, such as throttling the particular user traffi c using flow control mechanism.

Complementary Edges. In the stripe-based unicast probing methodology, the source needs to send certain packets of a stripe to one receiver and the rest of the packets of the stripe to a different receiver. Based on which link loss needs to be inferred, the order of these two receivers is different. We refer to each receiver as a complementary edge to the other receiver of an ordered pair of receivers. For a given node V, usually a leaf sibling of V can be a complementary node of V. We describe an algorithm (Figure 2.11) to fi nd complementary edges for each edge router of a given tree. These complementary edges will be used to infer link loss from the root to all links up to the closest common ancestor or least common ancestor (LCA) of both receivers, and from the LCA to both end receivers.

Union of the complementary edges of two edge routers will give all edge routers to use as receivers in the stripe-based methodology to infer loss of required links.

This algorithm needs to run initially when the network is setup and when it is reconfigured with additional routers/links. Then the result is stored in the SLAM. Since the algorithm is not run often, it does not impose excessive overhead.

Algorithm: *ComplementaryEdges (Tree T, Edge V)*

/* The tree is traversed backwards starting from V. */

$C' \leftarrow \emptyset$, $P = parent(V)$

while $P <>$ root

Add leaf X to C', where $lca(V, X) = P$

$P \leftarrow parent(P)$

return C'

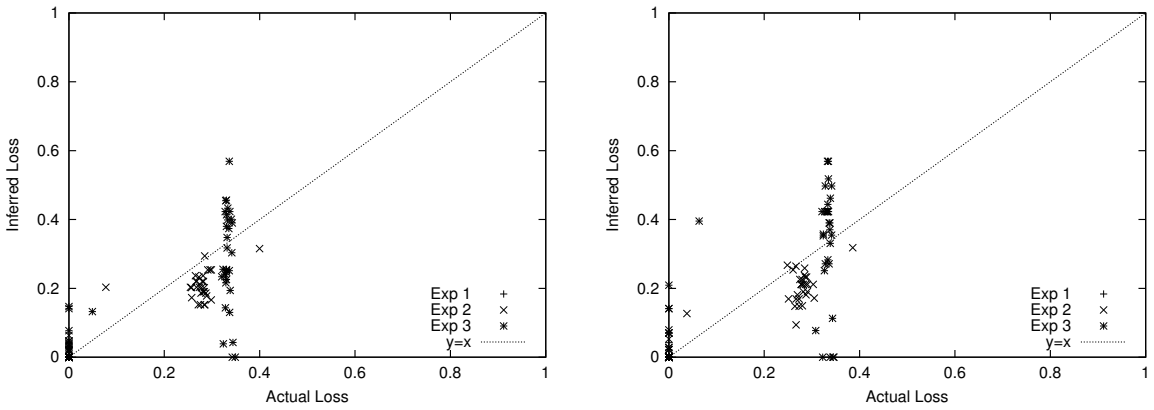
Fig. 2.11. Obtaining complementary edge router, which is a receiver in the stripe based monitoring.

2.5.4 Experiments and Results

Setup. The simulation setup is same as it is described in Section 2.4.5.

Loss inference. First, we depict the inferred loss using striped unicast probes. Figure 2.12 shows loss inference for the topology (Figure 2.3) for 3-packet stripes and 4-packet stripes. Exp 1 has fewer number of flows to cause packet drops inside the network domain. Exp 2 and Exp 3 have enough flows to cause huge packets drops in the network. The figure shows loss inference is close to the actual loss in most of the cases. In few cases, it over-estimates or under-estimates the loss. We can reduce this effect by increasing the time interval to measure probe loss. 4-packet stripe has little advantage over 3-packet stripe in our experiment.

Figure 2.13 shows the inferred loss of the link $C4 \rightarrow E6$ with different rates at which probes are sent. The objective of this experiment is to determine how often a stripe should be sent to infer loss accurately, i.e., the stripe transmission frequency. The figure shows that at least 20 stripes per second are required to infer a loss ratio close to the actual value.



(a) 3-packet Stripe

(b) 4-packet Stripe

Fig. 2.12. Inferring loss using unicast stripe-based probing. 4-packet stripes do not add signifi cantly over the 3-packet stripes.

The fi gure also demonstrates that a longer time is required for convergence in the striped-based scheme than in the core-assisted scheme. It takes at least 10 seconds to converge the inferred loss ratio reasonably to the actual loss ratio.

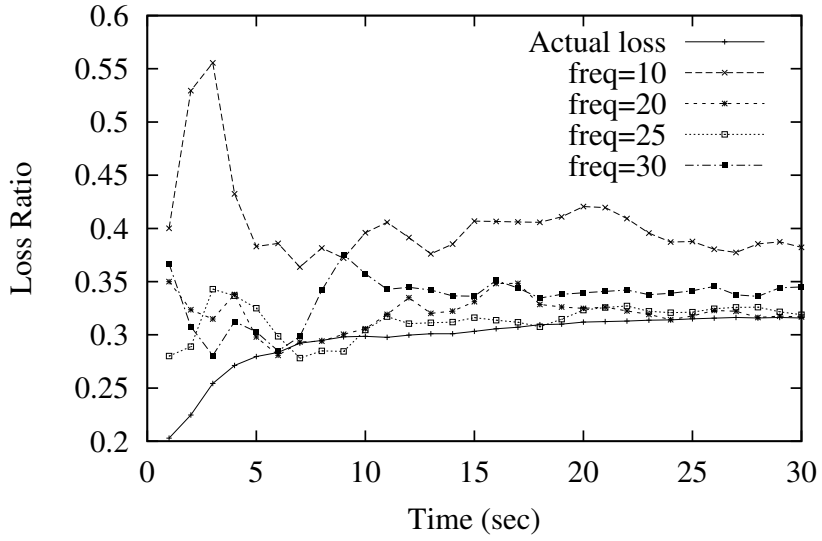


Fig. 2.13. Inferring loss of link $C4 \rightarrow E6$ using striped unicast probes. "freq" denotes the number of stripes transmitted per second.

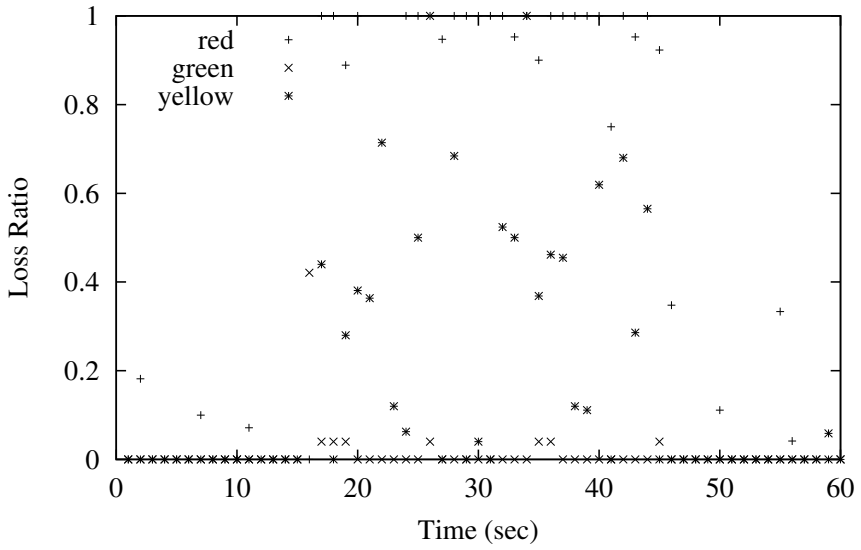


Fig. 2.14. Loss of different color probe packets in presence of high excess traffi c. Green probes see high loss when a severe attack starts. Yellow and Red probes experience high drops as expected

One important note is that stripe-based loss inference (as proposed in [29]) works well if core routers do not employ active queue management [23] or service differentiation. In assured forwarding, packets marked as red have a high drop probability while green packets have low drop probability. We send stripes of different colors to infer loss in this case. Figure 2.14 shows the loss of probes with different drop precedences.

2.5.5 Experiments: Detecting Attacks and SLA Violations

We demonstrate the detection of three scenarios: No attack, Attack1, Attack 2. "No attack" means no signifi cant traffi c in excess of capacity. This scenario has little loss inside the network domain. This is the normal case of proper network provisioning and enforcing traffi c conditioning at the edge routers. Attacks 1 and 2 inject more traffi c into the network domain from different ingress points. At each ingress point, the flows do not violate the profi les but overall they do. The intensity of the attack is increased during *t=15 seconds* to *t=45 seconds*. The attack causes packet drops of 15 to 30% in case of Attack 1 and more than 35% with Attack 2.

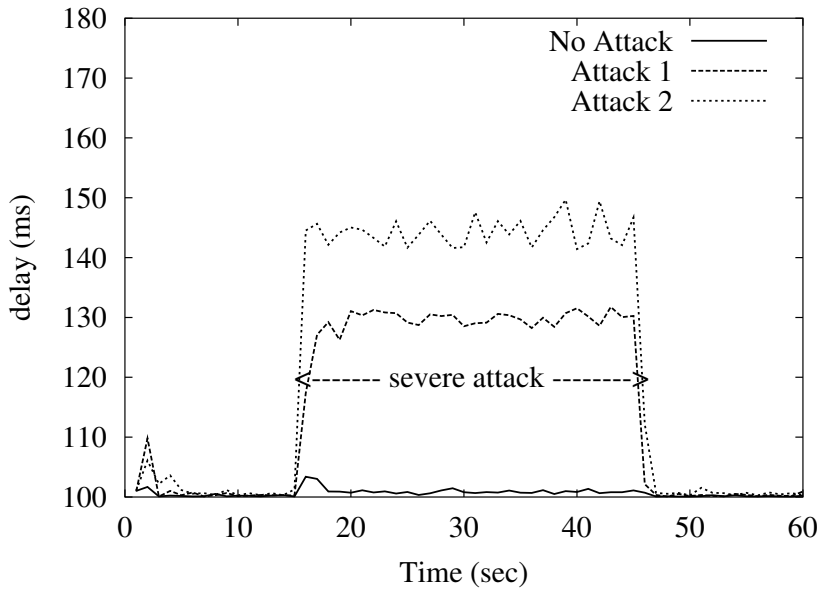


Fig. 2.15. Observed delay at the time of an attack. "Attack 1" results in packet loss in excess of 15-30%. "Attack 2" increases packet loss to more than 35%

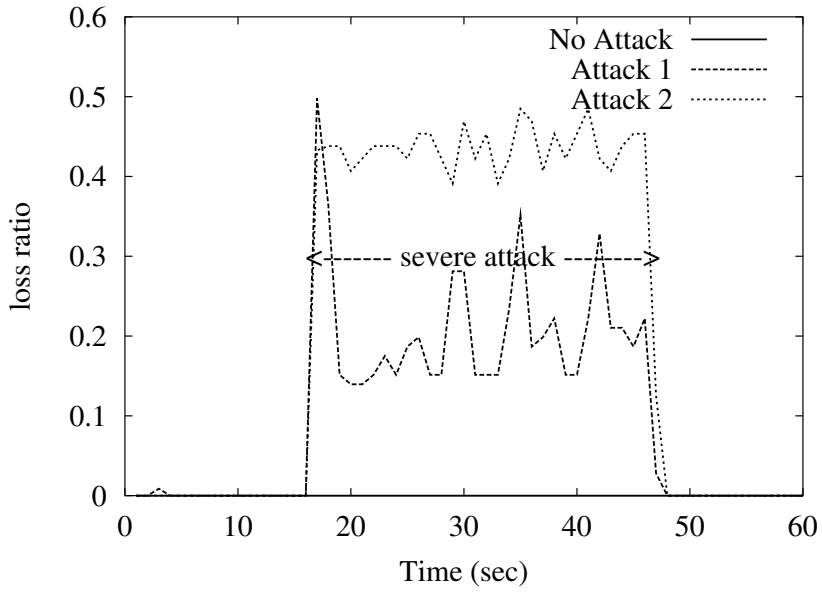


Fig. 2.16. Overall loss ratio during attack follows the same pattern as the delay.

Figure 2.15 shows that the increase of edge-to-edge link delay during the attack. The loss is inferred when a high delay is experienced inside the network domain. We use

equation (2.10) to compute overall loss for each class in a QoS network. This loss is shown in Figure 2.16. This loss follows the same pattern as the delay increment during the attack. This loss fluctuates with time. We measure the loss over short time duration so that it converges to the real loss in the network.

2.6 Overlay-based Monitoring

Overlay-based monitoring uses an interesting observation: service violation can be detected without exact loss value of each internal link. It only requires to check whether a link has loss higher than the specified threshold. Like [58, 63], we measure loss using average values in a recent time frame. The link with a high loss is referred to as a *congested link* (see Definition 2.6.1). The similar congestion measure is used in [64]. This congestion model is simple, and enables us to provide an in-depth analysis of the system. In future, we plan to use the model that considers loss correlation [65] among successive packets.

We devise a new approach to detect congested links by edge-to-edge measurements. These links are used to detect flows that pose threats to other flows by consuming extra resources.

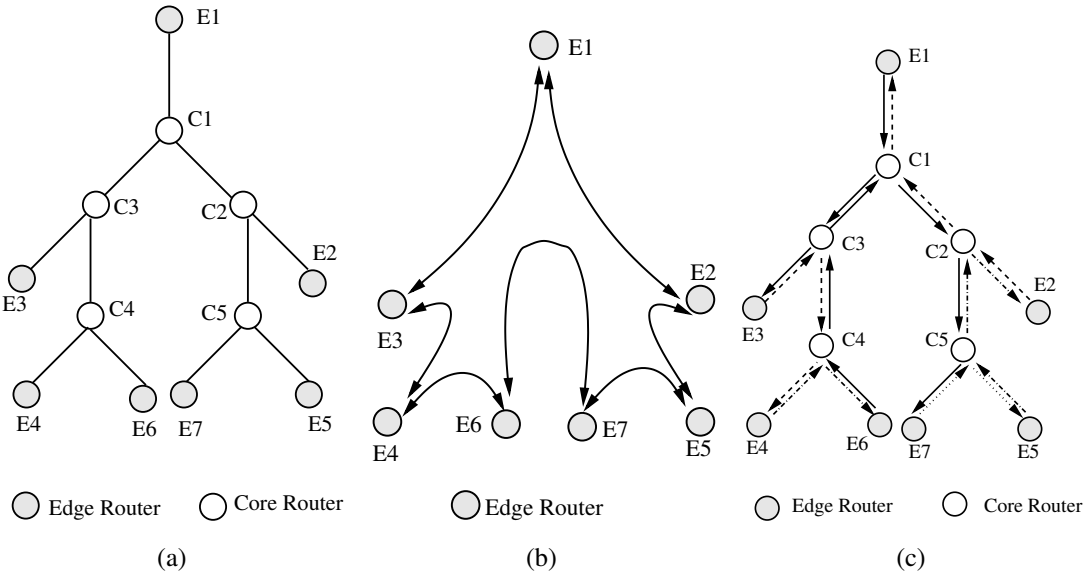


Fig. 2.17. (a) Tree topology transformed from a network domain. (b) All probing agents at the edge routers form a virtual network with both neighbors in an ordered sequence. (c) Direction of internal links for each probing.

2.6.1 Identifying Congested Links

To apply the overlay-based distributed probing, a network topology needs to be converted into a tree structure. This converting process is discussed later in this section. The tree contains core routers as internal nodes and edge routers as leaf nodes. Monitoring agents are deployed in the leaves to collect statistics from other edge routers to check SLA violations. The probing agents sit only at the edge routers and know their neighbors. The neighbors are determined by visiting the tree using depth first search algorithm starting from any edge router, and putting all edge routers in an ordered sequence. All probing agents form a virtual network on top of the physical network. The probes follow edge-to-edge path in the virtual network. A typical spanning tree of the topology, the corresponding overlay network, direction of all internal links for each probe are shown in Figure 2.17.

The following definitions and observations are used to describe the properties of the overlay network, and to identify congested links in the proposed simple and advanced method.

Definition 2.6.1 Congested link. *A link is congested if all loss measurement samples in a given time frame exceed a specified loss threshold.*

Definition 2.6.2 Overlay Network. *To connect all edge routers with their neighbors in a network domain, we build a virtual network and define as an overlay network.* We equivalently refer the tree topology or the virtual network to an overlay network.

Definition 2.6.3 Terminal core router. *A core router, which is connected to only one other core router in an overlay network is called a terminal core router.* In Figure 2.17, the core routers $C4$ and $C5$ are the terminal core routers.

Definition 2.6.4 Probe path. *A probe path $\mathcal{P}$ is a sequence of routers (either core or edge) $\langle E_1, C_1, C_2, \ldots, C_n, E_n \rangle$ where a router exists in the sequence only once. A probe packet originates at the edge router E_1, passes through the core routers C_1, C_2, $\ldots$, C_{n-1}, and C_n, in the given order, and terminates at the edge router E_n. We also represent the probe path $\mathcal{P}$ by the set of links, $\langle E_1 \rightarrow C_1, C_1 \rightarrow C_2, \ldots, C_n \rightarrow E_n \rangle$.*

Definition 2.6.5 Link direction. *A link $u \rightarrow v$, we say link from node u to v, is in inward direction (IN) with respect to node v. Similarly, the same link is in outward (OUT) direction with respect to node u.*

Lemma 2.6.1 *If a core router C is connected to two routers (core or edge) R_1 and R_2 only, the duplex path $R_1 \leftrightarrow C \leftrightarrow R_2$ can be replaced with the duplex link $R_1 \leftrightarrow R_2$, and both links are functionally equivalent in the overlay-based probing scheme.*

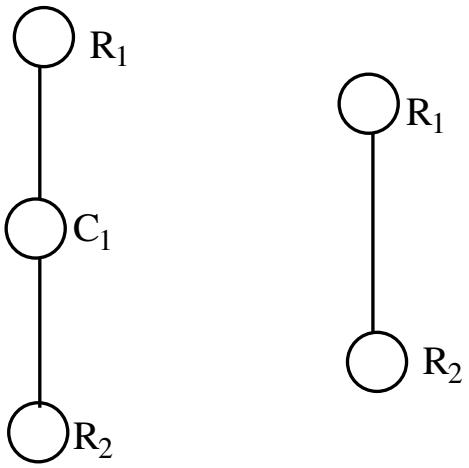


Fig. 2.18. Merging links that do not contribute during probing in the overlay-based monitoring.

Proof Let the core router c is connected to only two other routers R_1 and R_2. No probe path can be constructed that either includes the link $R_1 \rightarrow C$ and does not include $C \rightarrow R_2$ or vice versa; or includes $R_2 \rightarrow C$ and does not include $C \rightarrow R_1$ or vice versa. The traffic that passes through the link $R_1 \rightarrow C$ also passes through $C \rightarrow R_2$. The traffic that passes through the link $R_2 \rightarrow C$ also passes through $C \rightarrow R_1$. Therefore, for the purpose of probing, a logically equivalent overlay network can be constructed by replacing $R_1 \leftrightarrow C \leftrightarrow R_2$ with $R_1 \leftrightarrow R_2$. We say that the link $R_1 \rightarrow R_2$ is congested if and only if at least one of the links $R_1 \rightarrow C$ and $C \rightarrow R_2$ is congested, i.e. the bandwidth of $R_1 \rightarrow R_2$ is the minimum of the bandwidths of $R_1 \rightarrow C$ and $C \rightarrow R_2$. Similarly, the bandwidth of $R_2 \rightarrow R_1$ is the minimum of the bandwidths of $R_2 \rightarrow C$ and $C \rightarrow R_1$. ■

Lemma 2.6.2 *In an overlay network, every core router is connected to at least three other routers.*

Proof If a core router C is connected to two routers only, C together with its two connecting links can be replaced by a single link (Lemma 2.6.1). If C is a terminal core router and

C is not connected to any edge router, that is, C is connected to only one other router, C can never be included in a probe path and can be simply removed. Hence, all core routers are connected to at least three other routers. ■

Lemma 2.6.3 *An overlay network can be constructed in such a way that every terminal core router is connected to at least two edge routers.*

Proof Since a terminal core router C is connected to only one other core router (Definition 2.6.3), if C is not connected to at least two edge routers, C can be removed from the network (Lemma 2.6.1). ■

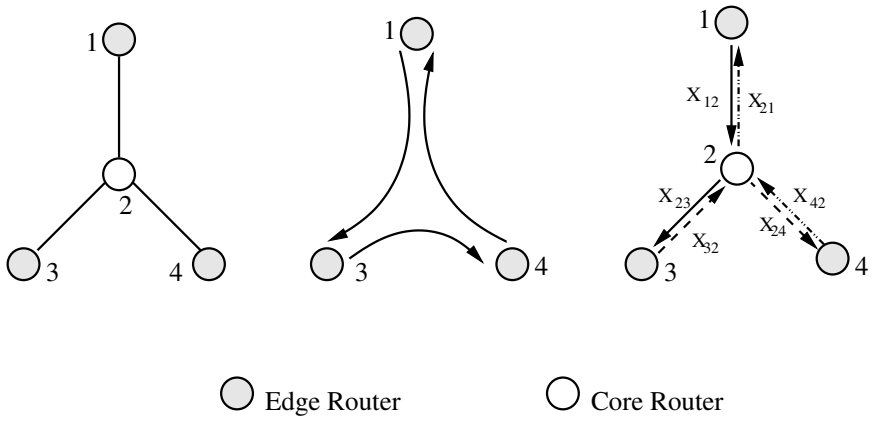


Fig. 2.19. (a) Spanning tree of a simple network topology. (b) Each edge router probes its neighbor edge router in counter-clockwise direction (c) Direction of internal links for each probing.

Simple Method

Our solution contains two methods: Simple method and Advanced method. We conduct total two rounds of probing in the Simple method. One in the counter-clockwise direction, and in the clock-wise direction starting from any edge router. The former one is referred to as *first round*of probing, and the latter one is referred to as *second round* of probing. In each round, probing is done in parallel.

We describe the loss monitoring scheme with a simple network topology. In this example, Figure 2.19b, edge router 1 probes the path $1 \rightarrow 3$, router 3 probes the path $3 \rightarrow 4$,

and 4 probes the path $4 \rightarrow 1$. Let $P_{i,j}$ be a boolean variable that represents the outcome of a probe between edge routers i to j. $P_{i,j}$ takes on value 1 if the measured loss exceeds the threshold in any link within the probe path, and 0 otherwise. Notice that $P_{i,j} = 0, \forall_{ij}, i = j$. We express the outcome of a probe in terms of combination of all link status. Let $X_{i,j}$ be a boolean variable to represent the congestion status of an internal link $i \rightarrow j$. We refer X to a *congestion variable*. From Figure 2.19c, we can write equations as follows:

$$X_{1,2} + X_{2,3} = P_{1,3} \quad X_{3,2} + X_{2,4} = P_{3,4} \quad X_{4,2} + X_{2,1} = P_{4,1}, \tag{2.12}$$

where (+) represents a boolean "OR" operation. We express status of internal links of any probe path of a network topology in terms of probe outcomes.

Note that loss in path $1 \rightarrow 3$ might not be same as loss in path $3 \rightarrow 1$. This path asymmetry phenomenon is shown in [66]. In general, $X_{i,j}$ is independent of $X_{j,i}, \forall_{ij}, i \neq j$.

The second round of probing, Figure 2.19(a), is done from $1 \rightarrow 4$, $4 \rightarrow 3$, and $3 \rightarrow 1$. We express the outcome of this round of probing in terms of internal links as follows:

$$X_{1,2} + X_{2,4} = P_{1,4} \quad X_{4,2} + X_{2,3} = P_{4,3} \quad X_{3,2} + X_{2,1} = P_{3,1}. \tag{2.13}$$

For an arbitrary topology,

$$X_{i,k} + \sum_{n=k}^{n=l-1} X_{n,n+1} + X_{l,j} = P_{i,j}. \tag{2.14}$$

The sets of equations (2.12 and 2.13) are used to detect congested link in the network. For example, if the outcome of the probing shows $P_{1,3} = 1$, $P_{1,4} = 1$, and rest are 0, we get the following:

$$X_{1,2} + X_{2,3} = 1 \qquad X_{1,2} + X_{2,4} = 1. \tag{2.15}$$

All other probes do not see congestion on its path, i.e., $X_{3,2} = X_{2,4} = X_{4,2} = X_{2,1} = X_{2,3} = 0$. Thus, the equation set (2.15) reduces to $X_{1,2} = 1$. Similarly, if any of the single link is congested, we can isolate the congested link. Suppose, two of the links, $X_{1,2}$ and $X_{2,3}$, are congested. The outcome of probing will be $P_{1,3} = 1$, $P_{1,4} = 1$, and $P_{4,3} = 1$, which makes $X_{3,2} = X_{2,4} = X_{4,2} = X_{2,1} = 0$. This leaves the solution as shown in equation(2.16). Thus, the overlay-based scheme can isolate links with high loss in this topology.

$$X_{1,2} + X_{2,3} = 1 \qquad X_{1,2} = 1 \qquad X_{2,3} = 1. \tag{2.16}$$

Analysis of Simple Method. The strength of simple method comes from the fact that congestion variables in one equation of any round of probing is distributed over several equations in the other round of probing. If n variables appear in one equation in the first round of probing, no two (out of the n) variables appear in the same equation in the second round of probing (Lemma 2.6.4) or vice versa. This property helps to solve the equation sets efficiently. Theorem 2.6.1 shows that if any single probe path is congested with arbitrary number of links, the simple method can identify all the congested links. In Theorem 2.6.2, we show that the simple method determines the status of a link with very high probability when the congestion is low.

Lemma 2.6.4 *If $\mathcal{P}$ and $\mathcal{P}'$ are any probe paths in the first and the second round of probing respectively, $|\mathcal{P} \cap \mathcal{P}'| \leq 1$.*

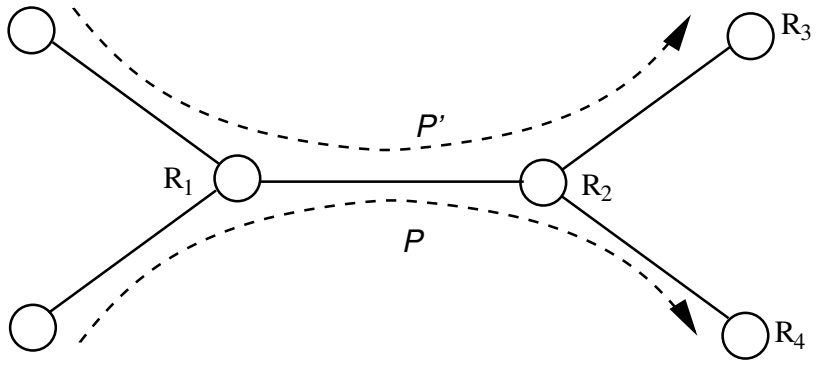


Fig. 2.20. Intersection of probe paths $\mathcal{P}$ and $\mathcal{P}'$. They meet at router R_1 and then split out at router R_2.

Proof Let the link $R_1 \rightarrow R_2$ (Figure 2.20) appears in path $\mathcal{P}$ in the first round of probing and path $\mathcal{P}'$ in the second round of probing. If R_2 is a core router, it is connected to at least two other routers, say R_3 and R_4 (Lemma 2.6.2). $\mathcal{P}$ passes through the link $R_2 \rightarrow R_4$ and $\mathcal{P}'$ passes through the link $R_2 \rightarrow R_3$. Since the tree does not have any cycle, $\mathcal{P}$ and $\mathcal{P}'$ never meet again. If R_2 is an edge router, both $\mathcal{P}$ and $\mathcal{P}'$ terminates at R_2. Therefore, $\mathcal{P}$ and $\mathcal{P}'$ can not have any common link in their paths after node R_2. Similarly, it can be shown that $\mathcal{P}$ and $\mathcal{P}'$ can not have common links before they meet at node R_1. That is $|\mathcal{P} \cap \mathcal{P}'| \leq 1$. ■

Lemma 2.6.5 *For any arbitrary overlay network, the average length of the probe paths in the Simple Method is* ≤ 4.

Proof In an overlay network, the number of links are $2(e + c - 1)$ considering both directions of a link. The edge routers are the leaves of the topology tree whereas the core routers are the internal nodes of the tree. The number of leaf nodes is greater than the number of internal nodes. Thus, the number of links is $\leq 2(e + e - 1) = 4e$. Number of the probe paths in first (or second) round of probing is$e$, and every link appears exactly once in each round. Hence, the average length of a path $\leq \frac{4e}{e} = 4$. ■

Theorem 2.6.1 *If only one probe path* $\mathcal{P}$ *is shown to be congested in the first round of probing, the simple method identifies each congestion link in*$\mathcal{P}$.

Proof Let, the congested probe path be $\mathcal{P} = \langle l_1, l_2, \ldots, l_k \rangle$ and X_i is the congestion variable for link $l_i, 1 \leq i \leq k$. X_i appears once in the equations for each round of probing. Let, X_m is in equation $X_m + f(S) = 1$ in the second round of probing, where S is a set of congestion variables excluding X_m that appear in the equation. The expression $f(S)$ does not contain any of the variables X_i for $1 \leq i \leq k, i \neq m$ (Lemma 2.6.4). From the first round of probing, we obtain $f(S) = 0$, because the outcome of all probe paths except $\mathcal{P}$ is zero in this round. Thus, we can determine X_m, which is 1, hence the status of the link l_m, for any $1 \leq m \leq k$. ■

Theorem 2.6.2 *Let* p *be the probability of a link being congested in any arbitrary overlay network. The simple method determines the status of any link of the topology with probability* $2(1-p)^4 - (1-p)^7 + 2p(1-p)^{12} - p(1-p)^{24}$.

Proof Let a particular link l appears in probe paths $\mathcal{P}_1$ and $\mathcal{P}_2$ in the first and second round of probing. The status of a link can be either non-congested or congested. We consider both cases separately and then combine the results.

When l *is non-congested.* The status of l can be determined if the rest of the links in either $\mathcal{P}_1$ or $\mathcal{P}_2$ are non congested. Let the length of probe paths $\mathcal{P}_1$ and $\mathcal{P}_2$ are i and k respectively. The probabilities that the other links in $\mathcal{P}_1$ and $\mathcal{P}_2$ are non-congested are $(1-p)^{i-1}$ and $(1-p)^{k-1}$ respectively. Since, only common link between paths $\mathcal{P}_1$ and $\mathcal{P}_2$ is l (Lemma 2.6.4), the following two events are independent: $Event_1$=all other links in $\mathcal{P}_1$ are non-congested and $Event_2$=all other links in $\mathcal{P}_2$ are non-congested. Thus, for a

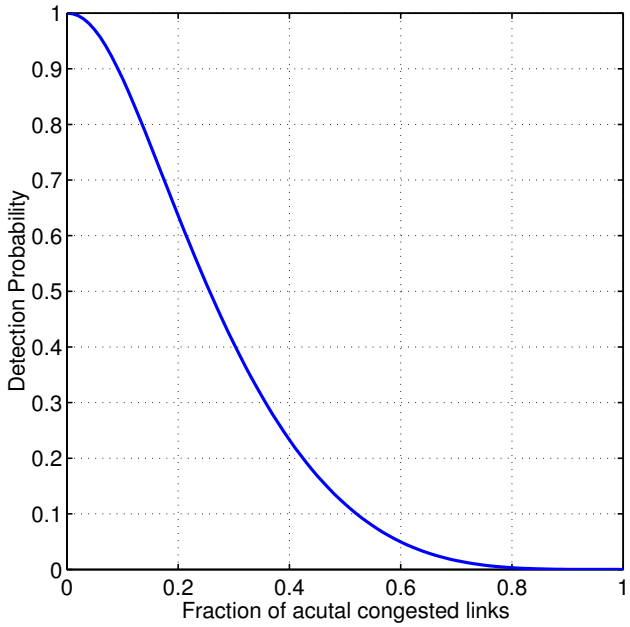


Fig. 2.21. Probability that the simple method determines the status of a link of any arbitrary topology. X-axis is the fraction of total links that are actually congested. The simple method performs extremely well when less than 20% links of a network are congested. If a network is more than 50% congested, the simple method can not contribute much.

non-congested link,

$$Pr\{\text{status of } l \text{ be determined}\} = (1-p)^{i-1} + (1-p)^{k-1} - (1-p)^{i-1}(1-p)^{k-1}$$
$$= (1-p)^{i-1} + (1-p)^{k-1} - (1-p)^{i+k-2}$$

Using the average length for the probe paths (Lemma 2.6.5), i.e., $i = k = 4$.
$Pr\{\text{Status of } l \text{ be determined}\} \approx 2(1-p)^3 - (1-p)^6$.

When l is congested. If l is a congested link, its status can be determined when all other links that appear on the probe path of l are non-congested and their status are determined. Let link l appears on a path in the first round of probing withl_1, l_2, and l_3 (considering the average path length is 4). The probability that l_1 (l_2 or l_3) is non-congested and determined is $(1-p)^4$. The probability to determine the status of these three links is $(1-p)^{12}$. This is true for the equations set in the second round, where l appears with variables other than l_1, l_2, and l_3. Thus, $Pr\{\text{Status of } l \text{ be determined}\} = 2(1-p)^{12} - (1-p)^{24}$. For any link l (congested or non-congested),

$$Pr\{\text{Status of } l \text{ be determined}\} = (1-p)[2(1-p)^3 - (1-p)^6] + p[2(1-p)^{12} - (1-p)^{24}]$$
$$= 2(1-p)^4 - (1-p)^7 + 2p(1-p)^{12} - p(1-p)^{24}. \quad \blacksquare$$

Figure 2.21 shows the probability to determine the status of a link when certain fraction of the links are actually congested. This figure shows that the simple method determines status of a link with probability close to 0.90 when 10% links of a network are congested. For 20% and 30% congestion, the probabilities are 0.64 and 0.40 respectively. This result is validated with the simulation result for two different topologies. The simple method does not help much when 50% or more links are congested. In that case, we use the advanced method to find probes that can decide the status of undecided links in the simple method.

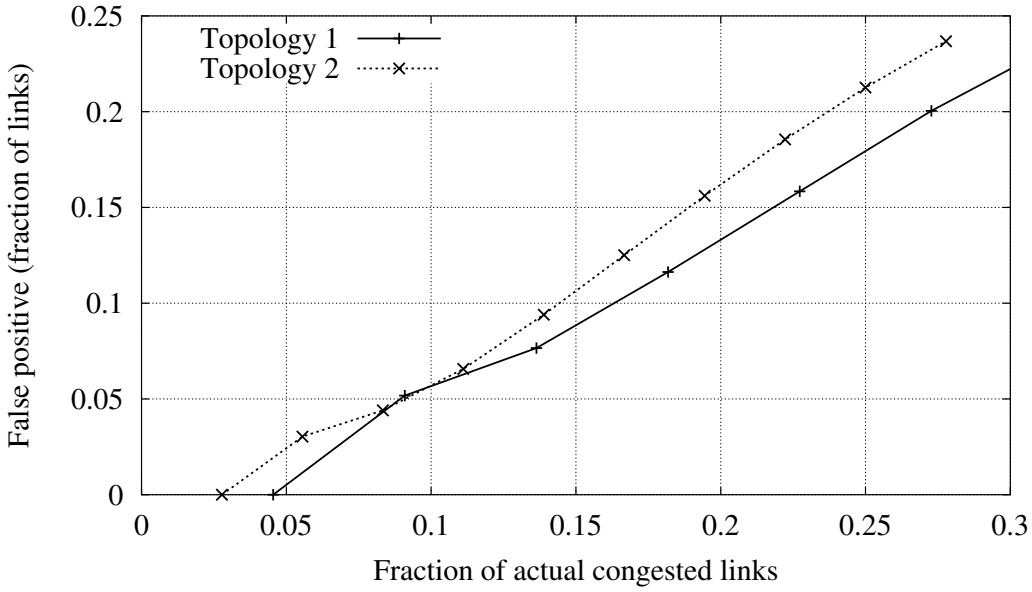


Fig. 2.22. The solution of the simple method can not decide about some links. If those links are considered as congested links, the solution of the simple method provides false positive by declaring some links as congested. The graph is shown for two topologies; Topology 1 shown in Figure 2.17 and Topology 2 shown in Figure 2.26(b). This figure does not compare the two topologies, instead, it shows the false positive as a percentage of total links with respect to percentage of links that are really congested. The solution does not have any false negative.

Having congestion on links that affect multiple probe paths might eventually lead to some boolean equations that do not have unique solutions. Thus, the solution of the simple method usually have some links undecided. If we report these undecided links as con-

gested, they will be referred to as false positive if some non-congested links will be reported as congested. The false positive is calculated as a ratio of undecided links labelled as congested to the total number links in the network. Figure 2.22 shows false positive for two topologies; Topology 1 shown in Figure 3.4(b) and Topology 2 shown in Figure 2.26(b). The false positive is a small percentage of all links of a domain. The number of links that are marked as false positive is very close to the number of actually congested links. The reason we get false positive is that some good (non-congested) links sit on the same probes of congested links, and the simple method does not have enough probes to isolate them. Notice that the solution does not have any false negative.

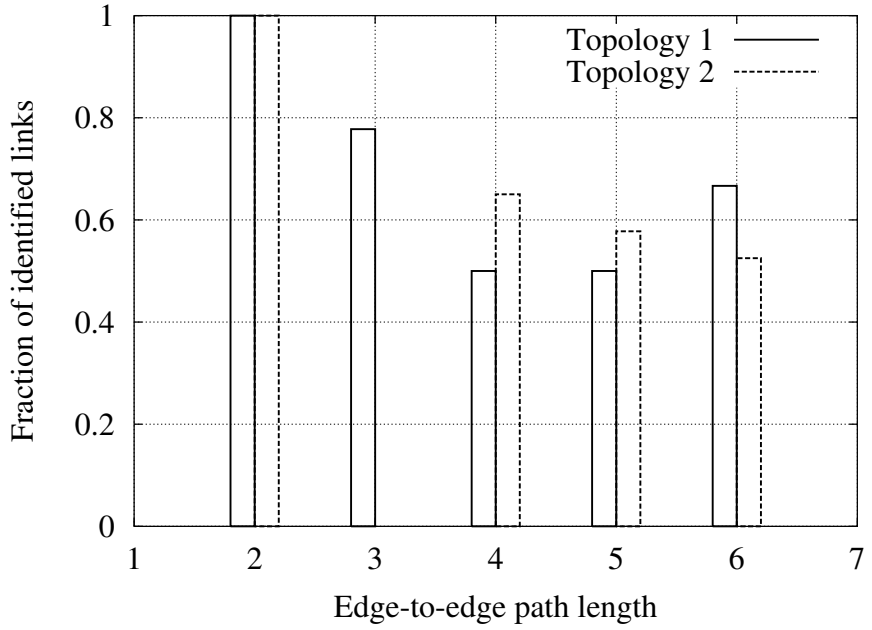


Fig. 2.23. Fraction of identifi ed links by the simple method for all edge-to-edge congested paths in the network. The X-axis shows all paths with a specifi c length. The solutions for edge-to-edge congestion paths do not have any false positive. Topology 2 does not have any path of length 3.

We further analyze the simple method when a network has congestion that spreads all the links from one edge router to any other edge routers. In real network, numerous flows come from different edge routers and make a series of links to be congested. In this case, the simple method performs very well. We observe that for an edge-to-edge congested path, the simple method performs well. It does not have any false positive. We plot this behavior for Topology 1 and Topology 2 with all possible edge-to-edge paths in Figure 2.23. On

the average, the simple method can isolate more than 50% of the congested links for the edge-to-edge congestion scenarios. Rest of the cases, the solutions have some equations with more than one variable. We can apply advanced method to be sure about the status of these links. The percentage of identified links is little high for the path length=6 in case of Topology 1, Figure 2.23. Because this path has more shared links comparing to other paths.

Advanced Method

The advanced method is used to identify the status of the undecided variables in the simple method. Therefore, the output of the simple method is used as the input of the advanced method. We traverse the topology tree to find probes that can help to decide about the values for each undecided variable.

The algorithm of the advanced method is shown in Figure 2.24. First, we conduct the simple method. Let the set of equations with undecided variables be $\mathbb{E}$. For each variable in equation set $\mathbb{E}$, we need to find two nodes that can be used to probe the network. Each probe requires one start node and one end node. The algorithm uses the same function to find start and end node of a probe. Link direction (Definition 2.6.5) plays an important role to find these probes. For example, in Figure 3.4, if link $C1 \rightarrow C3$ is congested, the start probe node can be $E2$, $E5$, or $E7$. On the other hand, if link $C3 \rightarrow C1$ is congested, the start probing node can be $E3$, $E4$, or $E5$.

For an undecided link $v_i \rightarrow v_j$, the function *FindNode* looks for leaves descended from node v_i and v_j. First, the algorithm searches for a node in IN direction on a subtree descended from v_i and then in OUT direction on a subtree descended from v_j. For any node v, the *DecidePath* explores all siblings of v to choose a path in a specified direction. The function avoids previously visited path and known congested path. It marks already visited path so that the same path will not be repeated in exploration of an alternate path.

If the network is congested in a way that no solution is possible, the *AdvancedMethod* can not add anything to the simple method. If there is a solution, the *AdvancedMethod* can obtain probes because this is an exhaustive search on the topology tree to find edge-to-edge paths that are not already congested.

Analysis of Advanced Method. The number of probes required in the advanced method depends on the number of congested links existing in a network. The advanced method starts with the undecided links. When the network is sparely congested or densely

```
Algorithm: AdvancedMethod()
begin
  Conduct the simple method. Outcome is an unsolved equation set 𝔼.
  for Each undecided variable X_ij of 𝔼 do
    node1 = FindNode(Tree T, v_i, IN) /*See Definition 2.6.5 for description of IN and OUT
    direction.*/
    node2 = FindNode(Tree T, v_j, OUT)
    if node1 ≠ NULL AND Node2 ≠ NULL then
      Probe(Node1, Node2). Update equation set 𝔼.
    end if
    Stop if no more probe exists
  end for
end

FindNode(Tree T, Node v_i, dir)
begin
  if v_i is leaf then
    return v_i
  end if
  v_k = DecidePath(v_i)
  if v_k = NULL then
    return NULL
  else
    node = FindEndNode(T, v_k, dir)
  end if
end

DecidePath(Node v_i, integer dir)
begin
  𝕍 ← siblings(v_i)
  for Each v of 𝕍 do
    if (dir=IN AND good(v → v_i)) OR (dir=OUT AND good(v_i → v)) then
      return v /*good(L) ⇔ L is neither congested nor visited.*/
    end if
  end for
  return NULL
end
```

Fig. 2.24. Advanced method to obtain probes that help to decide about the status of a congestion variable.

congested, the algorithm exit with fewer run and the number of trial for each congestion variable is low. To obtain how many trials we need to identify the status of each link, we

need the average length of a probe path d and on how many paths b a link lies on. For an arbitrary overlay network, we calculate the approximated value of d and b in Lemma 2.6.7 and Lemma 2.6.6 respectively. Using these two values we show that, Theorem 2.6.3, the advanced method identifi es the status of a link in$O(n)$ probing with a very high probability when the network is 20% congested or less.

Lemma 2.6.6 *For an arbitrary overlay network with e edge routers, on the average a link lies on $\frac{e(3e-2)}{8 \ln e}$ edge-to-edge paths.*

Proof To determine the average number of paths a link l lies on, we split the overlay network into two subtrees: T_1 and T_2. The link l lies on an edge-to-edge path whose one end belongs to T_1 and another end belongs to T_2. Let the number of edge routers in T_1 and T_2 be i and $e - i$ respectively. The total possible paths through l is $i(e - i)$. We observe that the probability that T_1 contains i edge routers is, $q_i \propto \frac{1}{i}$, (approximately, if the tree is not heavily skewed). i.e. $q_i = \frac{k}{i}$. The average number of paths the link l lies on, $b = \sum_{i=1}^{e/2} q_i.i.(e - i)$.

Now, $\sum_{i=1}^{e/2} q_i = \sum_{i=1}^{e/2} \frac{k}{i} = 1$,i.e. $k = \frac{1}{\ln \frac{e}{2}}$.

Therefor, $b = \sum_{i=1}^{e/2} k(e - i) = \frac{e(3e-2)}{8 \ln \frac{e}{2}} = \frac{e(3e-2)}{8 \ln e - 8 \ln 2} \approx \frac{e(3e-2)}{8 \ln e}$. ■

Lemma 2.6.7 *For an arbitrary overlay network with e edge routers, the average length of all edge-to-edge paths is $\frac{3e}{2 \ln e}$.*

Proof There are $e(e - 1)$ edge-to-edge paths exist for the advanced method. The number of links in a topology is $\approx 4e$ (see the proof of Lemma 2.6.5). The average length of a path $d = b \times \frac{4e}{e(e-1)}$, where $b = \frac{e(3e-2)}{8 \ln e}$ is the average number of paths a link lies on (Lemma 2.6.6).

Now, $d = \frac{e(3e-2)}{8 \ln e} \times \frac{4e}{e(e-1)} = \frac{3e-2}{2 \ln e} \times \frac{e}{e-1} \approx \frac{3e}{2 \ln e}$ (for large e). ■

Theorem 2.6.3 *Let p be the probability of a link being congested. The advanced method detects the status of a link with probability $1 - (1 - (1 - p)^d)^b$, where $d = \frac{3e}{2 \ln e}$ is the average path length and $b = \frac{e(3e-2)}{8 \ln e}$ is the average number of paths a link lies on.*

Proof The probability that a path of length d is non-congested $(1 - p)^d$. The probability of having all b paths congested is $(1 - (1 - p)^d)^b$. Thus, the probability that at least one non-congested path exists is $1 - (1 - (1 - p)^d)^b$. ■

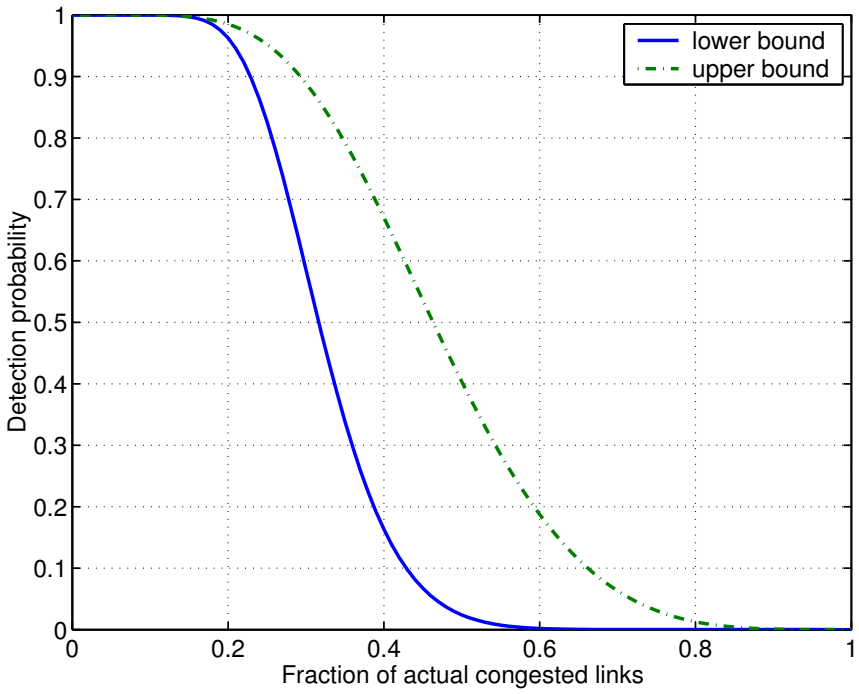


Fig. 2.25. Probability that the advanced method determines the status of a link of topology shown in Figure 2.17a. The X-axis is the probability that a link to be congested. The Y-axis is the probability that a good path (non-congested) exists for any link. The dotted graph shows the probability that a good path exists. The solid graph shows the probability that a good and decided path (from the first round) exists. These two curves provide lower and upper bound of the performance respectively.

The detection probability in the advanced method (Theorem 2.6.3) is plot in Figure 2.25 for Topology 1. This figure shows the probability that a good (non-congested) path exists for any link. The congestion status of the network is varied on the X-axis. Two graphs are shown: one shows the probability that a good path exists. It provides the upper bound because the solution can not be better than this limit. If no path exists, the advanced method can not do anything. The other graph shows the probability that a good as well as decided path exists. This provides the lower bound because it uses the decided links from the simple method and the solution can not be worst than this. The advanced method needs only *one probe* on the average to identify the status of the link when the network is less than 20% congested. In this case, the total required probes is $O(n)$. Some links might need more than one, which is not high because a good and decided path exists. If the network is 20-50%

congested, the advanced method might need multiple probes to decide the status of one unknown variable in $\mathbb{E}$. If the network is more than 50% congested, the advanced method can not find a good path easily because the path does not exist, and the advanced method terminates quickly. When the network is highly congested, we need to check almost all the flows any way. Thus, we can go to the detection phase instead of wasting time to rule out very few good links.

The performance of the advanced method is not significant when the network is heavily congested. It raises the question whether it is worth to use the advanced method when the network is highly congested. Instead, we can apply only the simple method, and go to the second phase of monitoring directly after that. Thus, we should go to the advanced method if the congestion is below a certain level. The congestion level is determined using the graph of the simple method shown Figure 2.21. For example, Figure 2.21 shows that the detection probability is 12% when the network is 50% congested. Therefore, if the simple method can identify the status of 12% links, we know that the network is 50% congested, and skip the advanced method. Thus, the algorithm to auto select simple and advanced method as follows: We conduct the simple method. The level of congestion is determined from Figure 2.21. If congestion level is less than a specified threshold (50%), only then we go to the advanced method. We proceed to the second phase (Section 2.7) of monitoring with this outcome.

2.6.2 Monitoring Algorithm

Let E be the set of all edge routers, both egress and ingress. One of these routers can act as a SLA Monitor (SLAM), or a separate entity can be used to act as SLAM. The algorithm proceeds as follows:

1. Each ingress router copies the header of user packets with probability p_{probe} to probe the network for delay measurement.

2. The monitor computes the average edge-to-edge delay updates the average delay of the user using equation 2.2.

3. The SLAM signals the appropriate edge routers (based on delay probes) to conduct probing for loss. This time each edge router probes its neighbor in clockwise and counterclockwise direction.

4. The SLAM analyze the outcome of the two rounds of probing. If necessary, it traverses the tree to look for probes for the advanced probing. In this case, the edge routers are informed who will be the receiver of this probing.

5. The edge routers feedback the outcome of the advanced probing to the SLAM. The SLAM continues the process until it gets a feasible solution.

6. The SLAM probes the network for throughput approximation when the loss is higher than the pre-confi gured threshold.

7. For users with higher delay, loss, and bandwidth consumption, the monitor decides about possible SLA violation. The monitor knows the existing traffi c classes and the acceptable SLA parameters per class. If there is any loss at the core for the EF traffi c class and if the AF loss ratio exceeds a certain level, an SLA violation is flagged. DoS attack is checked by comparing the total bandwidth achieved of a user with it SLA_{bw}. For each violation, it takes proper action, such as throttling the particular user traffi c using flow control mechanism.

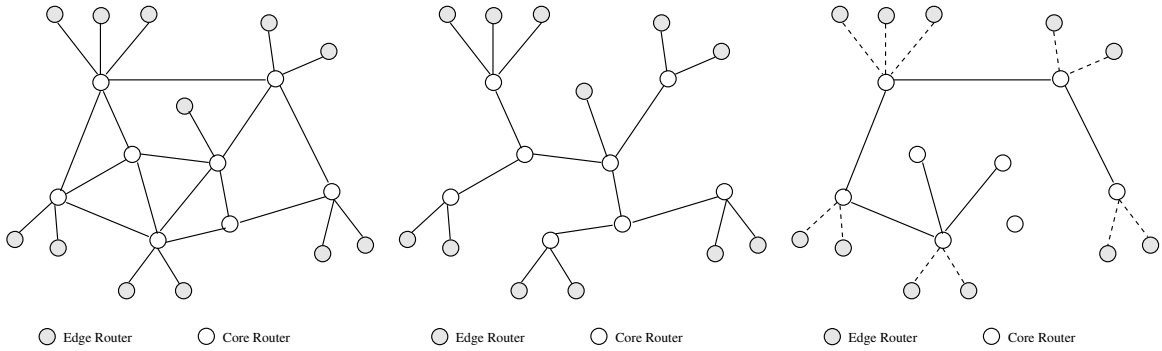
2.6.3 General Network Topology

Tree Conversion

The simple and the advanced methods are applicable to a network topology with a tree structure only. If there is any loop in the topology or multiple paths from one edge router to another edge router, we need to preprocess the topology before applying the algorithm. A related work for multicasting can be found in [49], which can be plugged in to our work. In this section, we describe a simple approach to solve this problem.

We split the topology into a spanning tree, and a set of subtrees. The monitoring algorithm is applied to all trees to identify the congested links. We might have multiple probe paths from one edge router to another. In this case, source routing is used for probe packets to follow the specifi ed route. Some of the subtrees may not be connected to edge routers, i.e., some parts of subtrees may consist of only core routers. To probe those links, non-congested links (determined by previous probing) are chosen to connect them to the edge routers. When all subtrees are done with probing, they are combined to obtain the fi nal outcome. As probing any path does not affect other paths, applying our scheme on

split tree will not affect each other. We obtain the union of all congested links from each topology as a final set of congested links for the whole topology.



(a) Original Topology (b) Converted tree topology (c) Rest of the topology

Fig. 2.26. Preprocessing of a general tree topology to apply edge-to-edge probing. The original topology is split into tree topologies. Then, the results are aggregated to get overall picture of a network.

In Figure 2.26, the general topology (Figure 2.26a) is split into two trees. The first one (Figure 2.26b) is a spanning tree of the general topology. The other one (Figure 2.26c) is a tree, where two core routers are not connected to any edge router. We need to add links to these core routers so that we can access this link from edge routers. When the spanning tree is done with probing, we select some good links to connect these core routers with the edge routers. At the end, all results can be combined together to reflect the overall status of the topology. The topology preprocessing is done infrequently only when a network is setup, and when any link or router is added.

We note that Figure 2.26a follows a similar pattern of the Sprint topology reported by Spring et al. [67]. For simplicity, we use this one instead of the actual backbone topology of Sprint. However, we can convert any arbitrary topology into tree structure to apply our monitoring algorithm.

Impact on Performance

The shape of the tree does not affect the monitoring mechanism. The probing scheme is applicable to any tree as long as it complies with the definitions we provide in this sec-

tion. However, the shape of the tree does have impact on the performance of the analytical analysis.

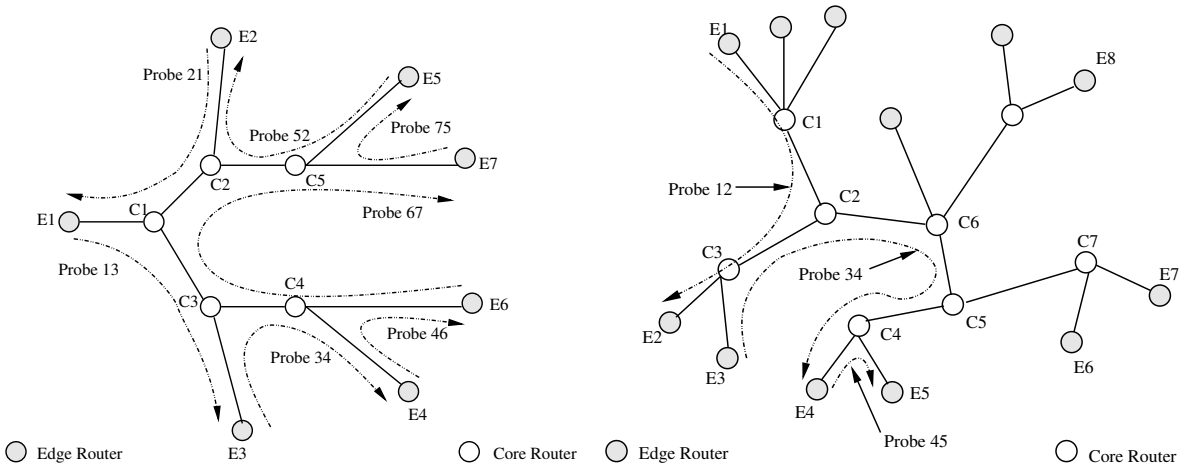
The analysis of the simple method uses Lemma 2.6.5 to calculate the average probe path length. It provides a upper bound of the length of probe paths. If number of edge routers is much higher than the number of core routers ($e \gg c$ in Lemma 2.6.5), the actual length will be lower than 4. The performance of the probing scheme will be better than the performance of the analytical analysis.

The analysis of the advanced method assumes the tree is not heavily skewed. To determine on how many paths a link lies on, Lemma 2.6.6 assumes that the probability with which a tree contains i edge routers after the tree is split into two subtrees is inversely proportional to the i. This is true when the tree is not heavily skewed. If it is, we can not use Lemma 2.6.6 and Lemma 2.6.7 to calculate d and b respectively for Theorem 2.6.3. If the d and b parameters can be calculated for any specialized tree, Theorem 2.6.3 will hold for the parameters. We note that the ISP topologies reported in [67] does not have this pattern.

2.6.4 Limitations of Overlay-based Monitoring

There are some limitations for the overlay-based distributed monitoring approach. For example, in Figure 2.19a, if both $X_{2,3}$ and $X_{2,4}$ are congested, we can not decide about $X_{1,2}$. Because we need at least one non-congested outgoing link from core router 2 to decide about the link $X_{1,2}$. The argument is the same for $X_{2,1}$ when both $X_{3,2}$ and $X_{4,2}$ are congested. If all links have the same bandwidth, we can report all three links as congested. Even if $X_{1,2}$ ($X_{2,1}$) has the combined capacity of the two outgoing (incoming) links, the argument is still valid. Each terminal core has at least two edge routers (Lemma 2.6.3). As long as any non-congested $core \rightarrow edge$ link exists, our method can provide partial solution. If not, the algorithm will report one non-congested link as congested, which is a close approximation of the actual result.

When all $edge \rightarrow core$ or $core \rightarrow edge$ links are congested in a network domain, the outcome of all probes will be congested. The final solution of the simple and advanced method is *all links are congested*. This solution is still useful because it is very likely that the whole network is congested. Thus, we can also go to the detection phase considering the whole network is congested. If some combinations of $edge \rightarrow core$ or $core \rightarrow edge$ are not congested, we can use them to provide a partial solution for the network.



(a) Topology 1 (b) Topology 2

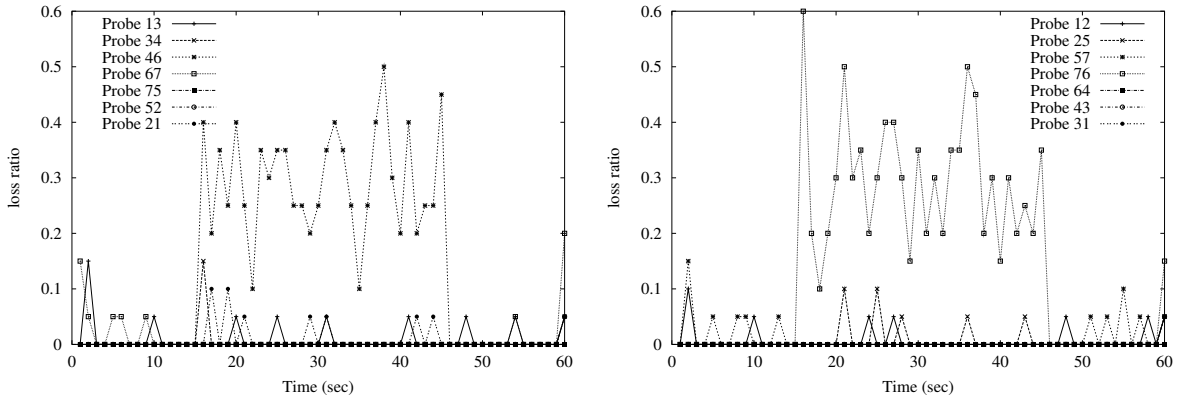
Fig. 2.27. Topology used to detect service violations using distributed probing. All edge routers are connected to one or multiple domains. All core to core router links are 20 Mbps with 30 ms delay and core to edge router links are 10 Mbps with 20 ms delay. The probes are named with the subscripts of the edge routers.

2.6.5 Experimental Results

Setup. The performance of the overlay-based monitoring scheme is evaluated using simulation. Attacks are simulated to show that the edge routers can detect service violations and attacks due to flow aggregation towards a downstream domain.

We use a network topology shown in Figure 2.27a, which is similar to the one used in [29] to evaluate stripe-based loss ratio approximations. Figure 2.27b is a more complex topology, which is used to show what happens when multiple attacks occur simultaneously, and one changes the behavior of the others. Multiple domains (not shown in the Figure 2.27) are connected to the edge routers for both topologies to create flows along all links in the domain. In Topology 1, flows coming through $E1$, $E2$, $E3$ are destined to edge router $E6$ to make the link $C4 \rightarrow E6$ congested. Many other flows are created to ensure that all links carry a significant number of flows. In Topology 2, multiple links are made congested.

Inferring congested links. We demonstrate how the distributed probing to detect congested links in a network domain. Some of the hosts that are connected to domains at-



(a) Counterclockwise probing.

(b) Clockwise probing.

Fig. 2.28. Probe outcome both for counterclockwise and clockwise direction. Probe 46 in (a) and Probe 57 in (b) have high losses, which means that link $C4 \rightarrow E6$ is congested.

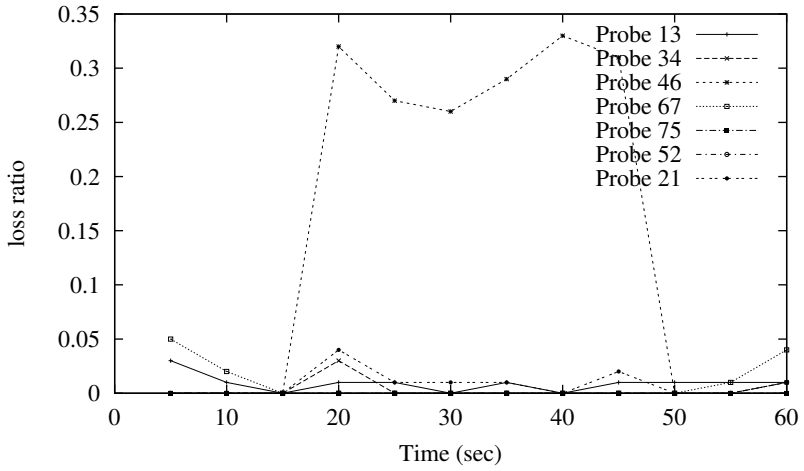


Fig. 2.29. Probe outcome using 5-second averages for the same experiments shown in Figure 2.28a.

tached with the edge routers violate SLAs. They inject more traffi c through multiple ingress routers to conduct an attack on the link $C4 \rightarrow E6$. The intensity of the attack is increased during the interval from *t=15* seconds to *t=45* seconds. The attack causes around 35% of packet drops except an initial jump at 15 sec.

To identify the congested links, the edge routers probe to their neighbors. Figure 2.28 shows that *Probe 46* in counterclockwise direction and *probe 57* in clockwise direction experience high losses. Other probes do not face high losses, that is, most of the internal links are not congested. It is important to note that *Probe 46* experiences high loss but *Probe 64*, which is in the opposite direction to *Probe 46*, faces very small amount of packet loss. This verifies that link loss in both directions of a link can be very different, based on the traffic load on each direction. We detect that link $C4 \rightarrow E6$ is the only congested link in the domain.

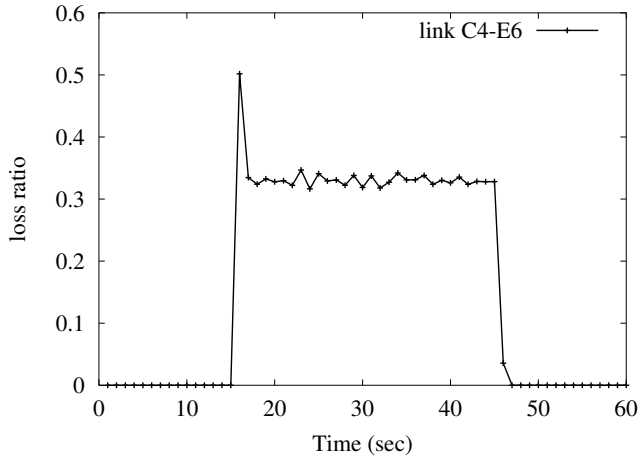


Fig. 2.30. Actual loss in link $C4 \rightarrow E6$. Other links have low losses. This verifies that our monitoring scheme detects the congestion properly.

All points in the Figure 2.28 are calculated by taking averages of samples over one second time period. If we take the average over a longer time period, we can avoid the high fluctuations of loss. Figure 2.29 shows the same graph as it is shown in Figure 2.28 with a 5-second average of each sample point. We observe that taking averages over a longer time period helps more in reducing the fluctuations than in increasing the number of probes per second. The actual loss for this congested link is very high, which is shown in Figure 2.30. It verifies that the distributed probing is able to detect links with high losses.

2.6.6 Local vs. Global Congestion

We address the question what happens if the congestion status is changed during the probing. To show an example, we use the Topology 2 (Figure 2.27b). This topology is more complex, and we simulate congestion in such a way that congestion in one area might affect the congestion of another area. Two attacks are simulated in this case. The first attack (Attack 1) is due to excessive flows coming from different edge routers to make the link $C4 \rightarrow E5$ congested. All of the probes in the first round are good except "Probe 45". This attack continues up to time $T = 50$ second (Figure 2.31). At this time, we have another attack (Attack 2), which is more severe than Attack 1. This attack causes several links on "Probe 34" path congested. It is interesting to note that Attack 2 actually causes Attack 1 to be disappeared. Because most of the traffic that causes Attack 1 on the link $C4 \rightarrow E5$ are now dropped earlier in their path due to Attack 2 (Figure 2.31).

This experiment shows that a local congestion might disappear due to a global and severe congestion. The main objective of our work is to pin point a congestion. However, if the congestion is changed while an experiment is being conducted, it catches the latest congestion. The simple method can complete two rounds of probing within $10-20$ seconds. If both rounds of probing are done in parallel, it takes only 10 seconds. If a congestion does not last for 20 seconds, we believe that no action is necessary to alleviate it.

2.7 Detecting Violations and Attacks

Violation detection is the second phase of our monitoring process. When delay, loss, and bandwidth consumption exceed the pre-defined thresholds, the monitor decides whether the network experiences a possible SLA violation. The monitor knows the existing traffic classes and the acceptable SLA parameters per class. For each service class, we obtain bounds on each SLA parameter that will be used as a threshold. A high delay is an indication of abnormal behavior inside a network domain. If there is any loss for the guaranteed traffic class, and if the loss ratios for other traffic classes exceed certain levels, an SLA violation is flagged. This loss can be caused by some flows consuming bandwidths above their SLA_{bw}. Bandwidth theft is checked by comparing the total bandwidth obtained by a user against the user's SLA_{bw}. The misbehaving flows are controlled at the ingress routers.

To detect DoS attacks, set of links L with high loss are identified. For each congested link, $l(v_i, v_j) \in L$, the tree is divided into two subtrees: one formed by leaves descendant

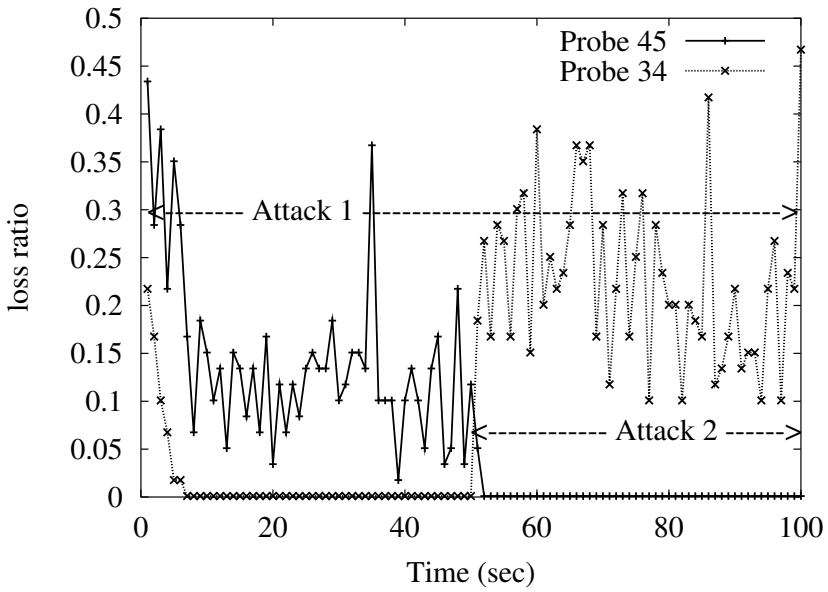


Fig. 2.31. Attack 1 causes link $C4 \rightarrow E5$ congested. However, Attack 2 comes from different edge routers to E4, which causes the traffic of Attack 1 to drop early. As a result Probe 45 is not congested after 50 sec.

from v_i and the other from the leaves descendant from v_j. The former subtree has egress routers as leaves through which high aggregate bandwidth flows are leaving. If many exiting flows have the same destination IP prefix, either this is a DoS attack or they are going to a popular site [68]. Decision is taken by consulting the destination entity. Jung *et al.* analyze the characteristics of flash crowd and DoS attacks in [69], which reveals several distinguishable features between these two. For example, the client distribution of a flash crowd event follows popular distribution among ISPs and networks, however, this is not true for a DoS attack. The other distinguishable features are per client request rate, overlap of clusters a site sees before and after the event, and popularity distribution of the file accessed by the clients. Using these characteristics, the monitor can decide whether it is a DoS attack or a flash crowd. In case of an attack, we control it by triggering filters at the ingress routers, which are leaves of the subtree descendant from v_i and feeding flows to the congested link. For each violation, the monitor takes action such as throttling a particular user's traffic using a flow control mechanism.

A scenario of detecting and controlling DoS attack is now illustrated using Figure 2.27a. Suppose, the victim's domain $\mathcal{D}$ is connected to the edge router $E6$. The monitor observes

that links $C3 \rightarrow C4$ and link $C4 \rightarrow E6$ are congested for a specified time duration Δt sec. From both congested links, we obtain the egress router $E6$ through which most of these flows are leaving. The destination IP prefix matching at $E6$ reveals that an excess amount of traffic is heading towards the domain $\mathcal{D}$. To control the attack, the monitor needs to identify the ingress routers through which the suspected flows are entering into the domain. The procedure to identify these ingress routers is discussed next.

2.8 Flow Aggregation and Filtering

An important question is how to identify ingress routers through which the flows are entering into the domain. To identify the flow aggregation, we use delay probes, and assign an ID to each router. An ingress router puts its ID on the delay probe packet. The egress router knows from which ingress routers the packets are coming. For example, in Figure 2.27a, say egress router $E6$ is receiving flows from $E1$, $E2$, $E3$, and $E5$. These flows aggregate during their trip to $E6$, and makes the link $C4 \rightarrow E6$ congested. We traverse the path backwards from the egress router to the ingress routers through the congested link to obtain the entry points of the flows that are causing attacks. In this example, all edge routers can feed the congested links and they all will be candidates for activating filters. Knowing the ingress routers and congested links, we figure out the ingress routers for the flows that are causing the attacks.

2.9 Experiments: Detecting DoS Attacks

A major advantage of using the SLA monitor is that it is able to detect denial of service (DoS) and Distributed DoS (DDoS) attacks in a network domain. The egress routers measure the outgoing rate of each flow. Using these rates, the monitor computes the total bandwidth consumption by any particular user. This bandwidth obtained by an user is compared to the SLA_{bw} of that user. If any flow gets very high bandwidth than it should, a DDoS attack is flagged. A DoS attack in a downstream domain can be detected by identifying the congested links, and the egress routers connected to the congested links. Using destination IP address prefix matching [68], we check whether many flows are aggregating towards a specific network or host. Consulting with the destination object, we control these flows at the ingress routers, if necessary.

We demonstrate the detection of *no attack* and *DoS attack*. "No attack" means no significant traffic in excess of the capacity. This scenario has little loss inside the network domain. This is the normal case of proper network provisioning and enforcing traffic conditioning at the edge routers. A DoS attack injects excessive traffic into the network domain from different ingress points. At each ingress point, the flows do not violate the profiles but overall they do. The intensity of the attack is increased during *t=15* seconds to *t=45* seconds. This attack causes packet drops of more than 35%. Figure 2.32 shows that the edge-to-edge delay is increased more than 100% in presence of the attack. The outcome of one round of loss probing is shown in Figure 2.33. The overlay-based schemes detect high losses in links $E2 \rightarrow C2$, $C1 \rightarrow C3$, $C3 \rightarrow C4$, and $C4 \rightarrow E6$. The link $C4 \rightarrow E6$ has a high loss for a short period of time. Since, some TCP flows adjusted their rates, and it causes the link to be non-congested one again. The egress router for the exiting flows is $E6$, and ingress routers through which flows enter into the domain are $E1$, $E2$, $E3$, $E4$, and $E5$. No traffic came from $E7$.

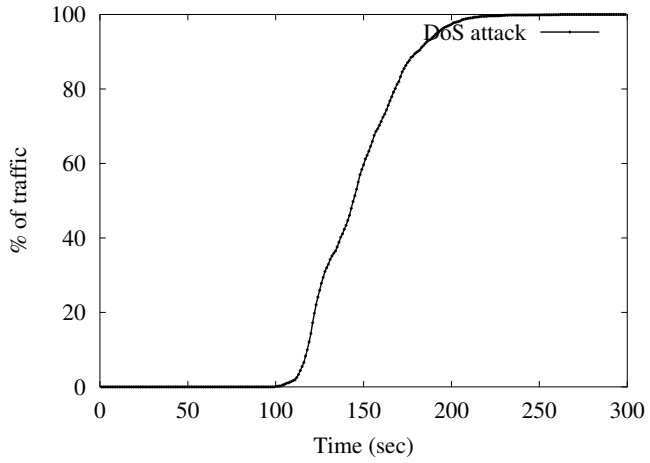


Fig. 2.32. Cumulative distribution function of edge-to-edge delay for link $E1 \rightarrow E6$. High delay indicates presence of severe attack in the domain.

2.10 Advantages of Overlay-based Monitoring

The advantages of using *overlay*-based monitoring are as follows:

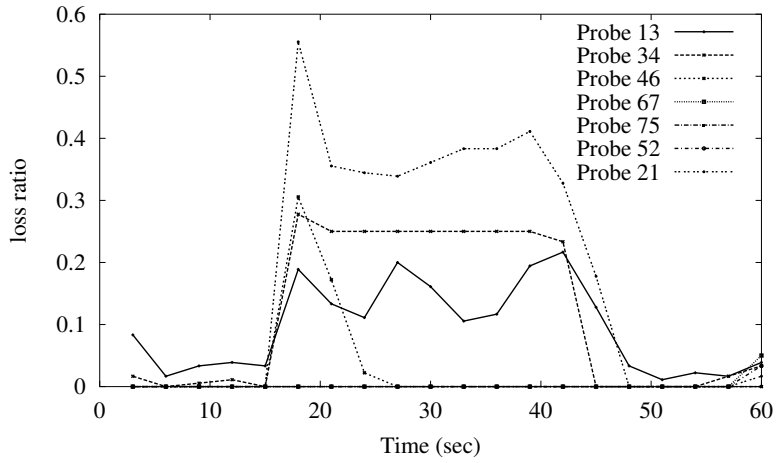


Fig. 2.33. Congestion on multiple probe paths due to severe attack. It indicates multiple links are having high losses.

1. The simple method of the overlay-based probing requires $O(n)$ probes to identify links with a high loss whereas the stripe-based scheme requires $O(n^2)$, where n is the number of edge routers in the domain. When the congestion is low, the advanced method requires $O(n)$ probes with a probability at least 0.97. If the congestion is high, more probes are required, however, it does not exceed $O(n)$.

2. The overlay scheme is able to detect violations in both directions for any link in a domain, whereas the stripe-based method can detect any violation only if the flow direction of the misbehaving traffic is the same as the probing direction from the root. To achieve the same result, the stripe-based method needs to probe the whole tree from all edge routers requiring $O(n^3)$ probes.

3. The overlay-based scheme can use TCP-based loss measurements (e.g. *Sting* [66]) to detect losses in both directions in one probe cycle.

4. In the stripe based scheme, two leaves/receivers are probed at a time. It takes a long time to complete probing the whole tree. If all leaves are probed simultaneously, in our example, $E1 \rightarrow C1$ link will face huge amount of traffic at that time. On the other hand, the overlay scheme can do parallel probing quite naturally.

2.11 Comparative Evaluation

Based on our experiments, we present a quantitative measure of performance to compare the core-assisted, stripe-based and overlay-based approaches. We use the topology shown in Figure 2.27a. We compare accuracy, convergence time, deployment overhead, and flexibility.

The communication and computation overhead calculation for all monitoring schemes are discussed in Chapter 3. In this comparative study, we show the communication overhead among them. Moore *et al.* [70] show that 50% of the attacks last for 10 minutes, 30% last for 30 minutes, 17% last for 60 minutes, 2% last for 5 hours and 1% last for 10 hours or more. Based on this study, we use a similar scenario to compute the communication overhead for the core-assisted approach.

Accuracy is computed using the deviations of approximating the loss ratio from the actual loss ratio value. We calculate accuracy based on our experimental results, with $f = 20$ as the probing frequency for the edge-to-edge approach. The convergence time is related to accuracy, i.e., time to sample the result to make it close to the actual value.

Deployment overhead considers which components of the network must be modified. The edge-to-edge approach needs to modify only edge routers, while the core-assisted approach requires change to both edge and core routers. The edge-to-edge scheme is thus considered more flexible since it is easier to deploy. However, the core-assisted approach gives more insight of the network domain. The packet drop history can be used to detect flow aggregation with IP prefix matching to detect DoS attacks.

Figure 2.34 depicts a quantitative comparison of the three approaches. Note that we use a high percentage of misbehaving flows as in [70]. For a large domain with millions of flows per second, the core-assisted approach will exhibit high communication overhead. The overlay has the lowest communication overhead, whereas the core-assisted has the highest. The core and overlay have better accuracy and less convergence time. The core-assisted in not flexible because it has high deployment overhead.

2.12 Conclusion

We have investigated methods to detect service level agreement violations in QoS networks. These methods are useful for network re-dimensioning, as well as for detection of denial of service and bandwidth theft attacks. The core-assisted loss measurement method

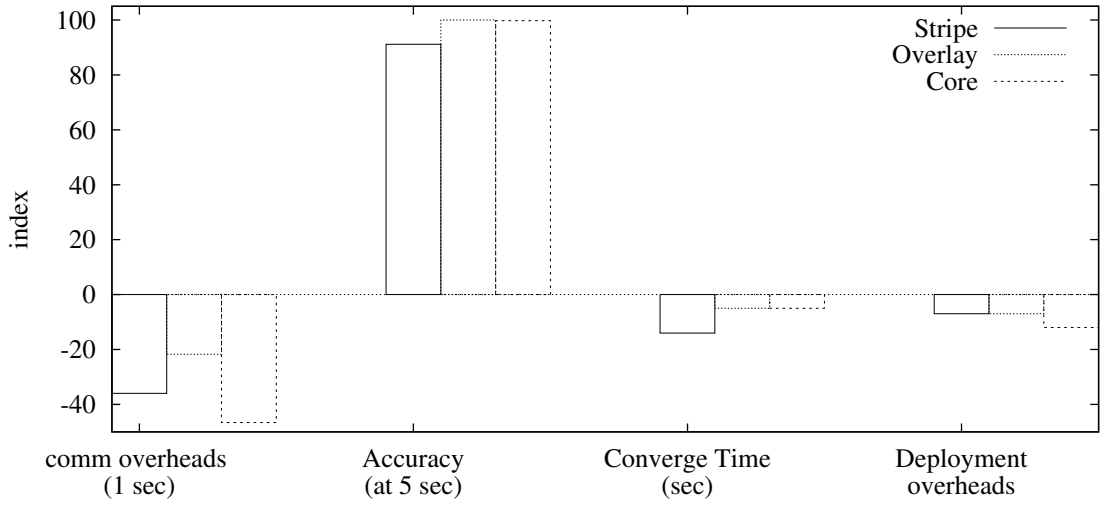


Fig. 2.34. Core-assisted, stripe-based, and edge-to-edge approaches: A quantitative study. Negative values are used for data for which low index represents better performance. For example, high overhead is not a desirable parameter. The core-assisted monitoring has the highest communication overhead (highest index) among the three schemes.

is powerful but difficult to deploy. An alternative edge-to-edge stripe-based loss inference scheme for different drop precedences was thus proposed. In the edge-to-edge probing approach, a low network probing rate has been shown to give incorrect results due to the loss of probes in case of excess traffic caused by an attack. A large number of probes, however, increases actual traffic delay and loss. We have shown that using probes with different drop precedences is necessary to infer loss in a QoS network. Our proposed stripe-based monitoring technique can aid in detecting attacks such as malicious traffic remarking or injection, without excessive overhead.

We have developed an overlay-based distributed network monitoring scheme to keep a domain safe from service violations and bandwidth theft attacks. We do not measure actual loss of all internal links, instead, we identify all congested links with high losses using network tomography and overlay networks. Our analytic analysis (verified by simulation) shows that even if 20% links of a network are congested, the status of each link can be identified with probability $\geq .98$. If the network is 40% congested, this probability is still high ($.65$). However, if the network is more than 60% congested, this method can not achieve anything significant since almost every edge-to-edge path has one or more congested links.

This new tomography scheme requires only $O(n)$ probes when less than 20% links are congested, where n is the number of edge routers. For an OC3 link, the probe traffic to identify the congested links is 0.002% of link capacity. The distributed monitoring requires $O(n^2)$ in worst case in contrast to $O(n^3)$ probes required by the stripe-based monitoring to detect attacks in both directions of all links. The distributed monitoring conducts probing in parallel enabling the system to perform real time monitoring. The simulation results indicate that the proposed scheme detects service violations, bandwidth theft attacks, and DoS attacks caused by flow aggregation towards a victim network domain.

3. DOS ATTACKS: DETECTION AND PREVENTION

Denial of Service (DoS) attacks are a serious threat for the Internet. DoS attacks can consume memory, CPU, and network resources and damage or shut down the operation of the resource under attack (victim). The quality of service (QoS) enabled networks, which offer different levels of service, are vulnerable to QoS attacks as well as DoS attacks. The aim of a QoS attack is to steal network resources, e.g., bandwidth, or to degrade the service perceived by users. We present a classifi cation and a brief explanation of the approaches used to deal with the DoS and QoS attacks. In chapter 2, we have shown that monitoring can be used as an early detection of DoS attacks. In this chapter, a quantitative comparison among all schemes is conducted, in which, we highlight the merits of each scheme and estimate the overhead (both processing and communication) introduced by it. For the completeness of the discussion about the techniques to detect and prevent DoS attacks, we provide a brief summary of different monitoring schemes in this chapter. The comparison provides guidelines for selecting the appropriate scheme, or a combination of schemes, based on the requirements and how much overhead can be tolerated.

3.1 Introduction

The San Diego Supercomputer Center reported 12,805 denial of service (DoS) attacks over a three-week period in February 2001 [70]. This is just one of the numerous incidents in which DoS attacks are causing serious security threats to many systems connected to the Internet. The DoS attacks can be severe if they last for a prolonged period of time preventing legitimate users from accessing some or all of computing resources. Imagine an executive of a fi nancial institution deprived of access to the stock market updates for several hours or even several minutes. In [70], the authors showed that whereas 50% of the attacks lasted less than ten minutes, unfortunately, 2% of them lasted greater than fi ve hours and 1% lasted more than ten hours. There were dozens of attacks that spanned multiple days. Wide spectrum of motivation behind these DoS attacks exists. They range from political

conflicts and economical benefits for competitors to just curiosity of some computer geeks. Furthermore, cyber terrorism may not be excluded in the future.

In addition to DoS attacks, the quality of service (QoS) enabled networks are vulnerable to another type of attacks, namely, the QoS attacks. This attack is introduced and described in Chapter 2. A QoS-enabled network, such as a differentiated services network [5], offers different classes of service for different costs. Differences in the charging rates may entice some users to steal bandwidth or other network resources. An attacker in this environment as a user who tries to get more resources, i.e., a better service class, than what he has signed (paid) for. QoS attacks inject traffic into the network with the intent to steal bandwidth or to cause QoS degradation for other flows. Since the differentiated services framework is based on aggregation of flows into service classes, legitimate customer traffic may experience degraded QoS as a result of the illegally injected traffic. Taken to an extreme, that excess traffic may result in a denial of service attack. This creates a need for developing an effective defense mechanism that automates the detection and reaction to attacks on the QoS-enabled networks.

In this chapter, we first elaborate on the denial of service attacks and their potential threat on the system. We then classify the solutions proposed in the literature into two main categories: *detection* and *prevention* approaches. Several techniques are briefly discussed, focusing mainly on the salient features and highlighting the potential as well as the shortcomings of each mechanism. In addition, we show network monitoring techniques to detect service violations and to infer DoS attacks. We believe that network monitoring has the potential to detect DoS attacks in early stages before they severely harm the victim. Our conjecture is that a DoS attack injects a huge amount of traffic into the network, which may alter the internal characteristics (e.g., delay and loss ratio) of the network. Monitoring watches for these changes and identifies the congested links, which helps in locating the attacker and alerting the victim. Finally, we conduct a comparative evaluation study among the approaches presented. The aim of the study is to compare the behavior of the approaches under different situations of the underlying network. We draw insightful comments from the comparison that guide the selection of one or more defending approaches suitable for a given environment.

3.2 Approaches to Defeat DoS Attacks

In the literature, there are several approaches to deal with denial of service (DoS) attacks. In this section, we provide an approximate taxonomy (Figure 3.1) of these approaches. In addition, we briefly describe the main features of each approach and highlight the strengths and weaknesses of it.

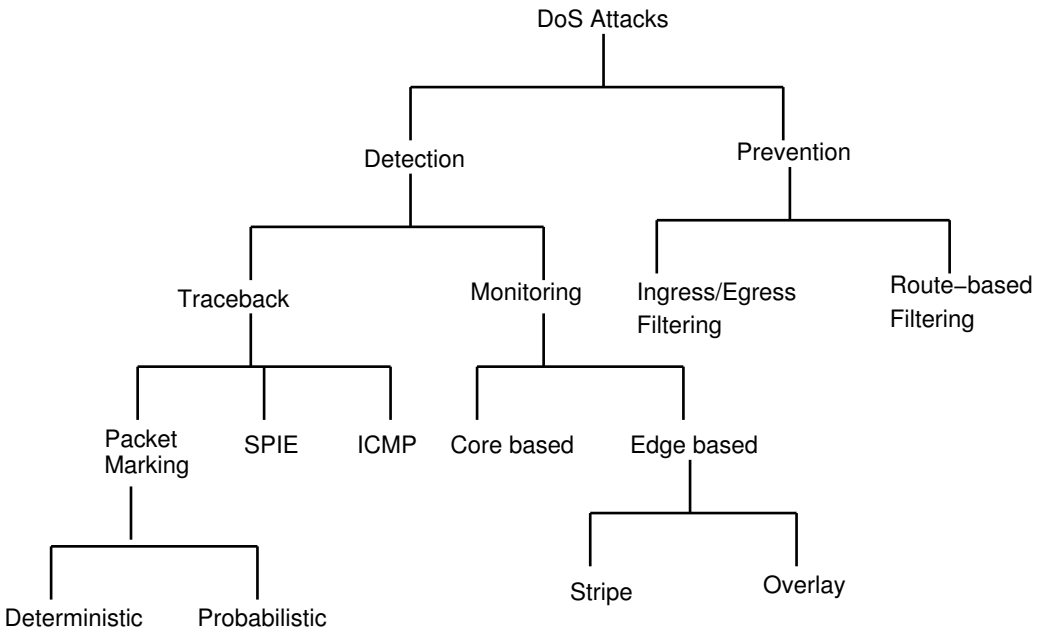


Fig. 3.1. Classifi cation of approaches to detect and prevent DoS attacks.

We divide the approaches for dealing with DoS attacks into two main categories: *detection* and *prevention* approaches. The detection approaches capitalize on the fact that appropriately punishing wrong doers (attackers) will deter them from re-attacking again, and will scare others to do similar acts. The detection process has two phases: detecting the attack and identifying the attacker. To identify an attacker, several *traceback* methods can be used, as explained later in this section. The obvious way to detect an attack is just waiting till the system performance decreases sharply or even the whole system collapses. We propose a more effective method for detecting attacks before they severely harm the system. We propose to use monitoring for *early* detection of DoS attacks. The details are given in Section 3.3. The prevention approaches, on the other hand, try to thwart attacks before they harm the system. Filtering is the main strategy used in the prevention approaches.

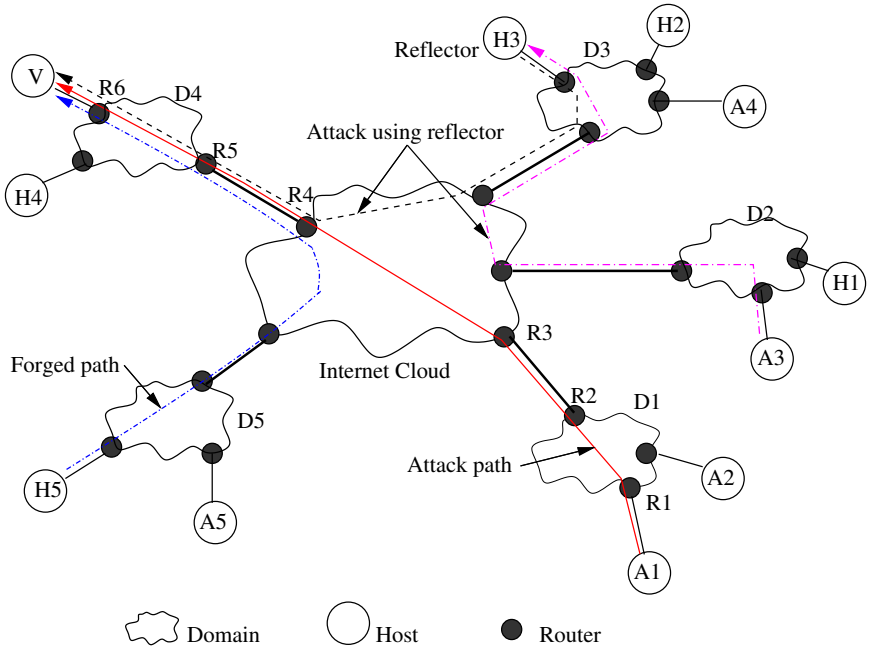


Fig. 3.2. Different scenarios for DoS attacks. Attacker $A1$ launches an attack on the victim V. $A1$ spoofs IP address of host $H5$ from domain $D5$. Another attacker $A3$ uses host $H3$ as a reflector to attack V.

To clarify the presentation, we use the hypothetical network topology shown in Figure 3.2 to demonstrate several scenarios for DoS attacks and how the different approaches react to them. The figure shows several hosts (denoted by Hs) connected to four domains. Throughout the chapter, we use "domain" to refer to an Autonomous Systems (AS) domain, which is a network administered by a single entity. $D1, D2, D3$, and $D4$, which are interconnected through the Internet cloud. In the figure, A_i represents an attacker i while V represents a victim.

3.2.1 DoS Attacks

The aim of a DoS attack is to consume the resources of a victim or the resources on the way to communicate with a victim. By wasting the victim's resources, the attacker disallows it from serving legitimate customers. A victim can be a host, server, router, or any computing entity connected to the network. Inevitable human errors during software development, configuration, and installation open several unseen doors for these type of attacks.

Several DoS attacks are known and documented in the literature [51, 70–72]. Flooding a victim with an overwhelming amount of traffic is the most common. This unusual traffic clogs the communication links and thwarts all connections among the legitimate users, which may result in shutting down an entire site or a branch of the network. This happened in February of 2000 for the popular web sites Yahoo, E*trade, Ebay, and CNN for several hours [71].

TCP SYN flooding is an instance of the flooding attacks [73]. Under this attack, the victim is a host and usually runs a Web server. A regular client opens a connection with the server by sending a TCP SYN segment. The server allocates buffer for the expected connection and replies with a TCP ACK segment. The connection remains half-open (backlogged) till the client acknowledges the ACK of the server and moves the connection to the established state. If the client does not send the ACK, the buffer will be deallocated after an expiration of a timer. The server can only have a specific number of half-open connections after which all requests will be refused. The attacker sends a TCP SYN segment pretending a desire to establish a connection and making the server reserves buffer for it. The attacker does not complete the connection. Instead, it issues more TCP SYNs, which lead the server to waste its memory and reach its limit for the backlogged connections. Sending such SYN requests with a high rate keeps the server unable to satisfy connection requests from legitimate users. Schuba *et al.* [73] developed a tool to alleviate the SYN flooding attack. The tool watches for SYN segments coming from spoofed IP addresses and sends TCP RST segments to the server. The RST segments terminate the half-open connections and free their associated buffers.

Other types of flooding attacks include TCP ACK and RST flooding, ICMP and UDP echo-request flooding, and DNS request flooding [51, 70]. This list is by no means exhaustive.

A DoS attack can be more severe when an attacker uses multiple hosts over the Internet to storm a victim. To achieve this, the attacker compromises many hosts and deploys attacking agents on them. The attacker signals all agents to simultaneously launch an attack on a victim. Barros [74] shows that DDoS attack can reach a high level of sophistication by using *reflectors*. A reflector is like a mirror that reflects light. In the Internet, many hosts such as Web servers, DNS servers, and routers can be used as reflectors because they always reply to (or reflect) specific type of packets. Web servers reply to SYN requests, DNS servers reply to queries, and routers send ICMP packets (time exceeded or host un-

reachable) in response to particular IP packets. The attackers can abuse these reflectors to launch DDoS attacks. For example, an attacking agent sends a SYN request to a reflector specifying the victim's IP address as the source address of the agent. The reflector will send a SYN ACK to the victim. There are millions of reflectors in the Internet and the attacker can use these reflectors to flood the victim's network by sending a large amount of packets. Paxson [75] analyzes several Internet protocols and applications and concludes that DNS servers, Gnutella servers, and TCP-based servers are potential reflectors.

3.2.2 Detection Approaches

The detection approaches rely on finding the malicious party who launched a DoS attack and consequently hold him liable for the damage he has caused. However, pinning the real attacker down is not a straightforward task. One reason is that the attacker spoofs the source IP address of the attacking packets. Another reason is that the Internet is *stateless*, which means, whenever a packet passes through a router, the router does not store any information (or traces) about that packet. Therefore, mechanisms such as ICMP traceback and packet marking are devised to figure out the real attacker. In this subsection, we describe several techniques to identify the attacker after the attack took place. We defer the issue of early detection of an attack till Section 3.3.

ICMP Traceback

Bellovin [76] proposes the idea of ICMP traceback messages, where every router samples the forwarded packets with a very low probability (e.g., 1 out of 20,000) and sends an ICMP Traceback message to the destination. An ICMP Traceback message contains the previous and next hop addresses of the router, timestamp, portion of the traced packet, and authentication information. In Figure 3.2, while packets are traversing the network path from the attacker $A1$ to the victim V, the intermediate routers ($R1, R2, R3, R4, R5,$ and $R6$) sample some of these packets and send ICMP Traceback messages to the destination V. With enough messages, the victim can trace the network path $A1 \rightarrow V$. The pitfall of this approach is that the attacker can send many false ICMP Traceback messages to confuse the victim.

To address Distributed DoS (DDoS) attacks by reflectors, Barros [74] proposes a modification to the ICMP Traceback messages. In his refinement, routers sometimes send ICMP

Traceback messages to the *source*. In Figure 3.2, $A3$ launches a DDoS attack by sending TCP SYN segments to the reflector $H3$ specifying V as the source address. $H3$, in turn, sends SYN ACK segments to the victim V. According to the modification, routers on the path $A3 \rightarrow H3$ will send ICMP messages to the source, i.e., to V. This *reverse trace* enables the victim to identify the attacking agent from these trace packets. The *reverse trace* mechanism depends only on the number of attacking agents, and not on the number of reflectors [75]. This achieves scalability because the number of available reflectors is much higher than the number of attacking agents on the Internet.

Snoeren *et al.* [77] propose an attractive hashed-based system that can trace the origin of a single IP packet delivered by a network in the recent past. The system is called source path isolation engine (SPIE). The SPIE uses an efficient method to store information about packets traversing through a particular router. The method uses n bits of the hashed value of the packet to set an index of a 2^n-bit *digest table*. When a victim detects an attack, a query is sent to SPIE, which queries routers for packet digests of the relevant time periods. Topology information is then used to construct the attack graph from which the source of the attack is determined.

Packet Marking

Instead of having routers send separate messages for the sampled packets, Burch and Cheswick [78] propose to inscribe some path information into the header of the packets themselves. This marking can be deterministic or probabilistic. In the deterministic marking, every router marks all packets. The obvious drawback of the deterministic packet marking is that the packet header grows as the number of hops increases on the path. Moreover, significant overhead will be imposed on routers to mark every packet.

The probabilistic packet marking (PPM) encodes the path information into a small fraction of the packets. The assumption is that during a flooding attack, a huge amount of traffic travels towards the victim. Therefore, there is a great chance that many of these packets will be marked at routers throughout their journey from the source to the victim. It is likely that the marked packets will give enough information to trace the network path from the victim to the source of the attack.

Savage *et al.* [72] describe efficient mechanisms to encode the path information into packets. This information contains the XOR (exclusive OR) of two IP addresses and a distance metric. The two IP addresses are for the start and the end routers of the link.

The distance metric represents the number of hops between the attacker and the victim. To illustrate the idea, consider the attacker $A1$ and the victim V in Figure 3.2. Assume there is only one hop between routers $R3$ and $R4$. If Router $R1$ marks a packet, it will encode the XOR of $R1$ and $R2$ addresses into the packet and sets the distance metric to zero, that is, it will encode the tuple $< R1 \oplus R2, 0 >$. Other routers on the path just increase the distance metric of this packet, if they don't decide to mark it again. When this packet reaches the victim, it provides the tuple $< R1 \oplus R2, 5 >$. Similarly, some packets may get marked at routers $R2$, $R3$, $R4$, $R5$, and $R6$ and they will provide the tuples $< R2 \oplus R3, 4 >$, $< R3 \oplus R4, 3 >$, $< R4 \oplus R5, 2 >$, $< R5 \oplus R6, 1 >$, $< R6, 0 >$, respectively, when they reach the victim. The victim can retrieve all routers on the path by XORing the collected messages sorted by distance. (Recall that $Rx \oplus Ry \oplus Rx = Ry$.) This approach can reconstruct most network paths with 95% certainty if there are about 2,000 marked packets available and even the longest path can be resolved with 4,000 packets [72]. For DoS attacks, this amount of packets is clearly obtainable because the attacker needs to flood the network to cause a DoS attack. (Moore *et al.* [70] report that some severe DoS attacks had a rate of thousands of packets per second.) The authors describe ways to reduce the required space and suggest to use the identification field (currently used for IP fragmentation) of IP header to store the encoding of the path information. They also propose solutions to handle the co-existence of marking and fragmentation of IP packets [72].

The main limitation of the PPM approaches stems from the fact that, nothing prevents the attacker from marking packets. If a packet marked by the attacker does not get re-marked by any intermediate router, it will confuse the victim and make it harder to trace the real attacker. Park and Lee [79] show that for single-source DoS attacks, PPM can identify a small set of sources as potential candidates for a DoS attack. For DDoS attacks, however, the attacker can increase the uncertainty in localizing the attacker. Therefore, PPM is vulnerable to distributed DoS attacks [79].

3.2.3 Prevention Approaches

Preventive approaches try to stop a DoS attack by identifying the attack packets and discarding them before reaching the victim. We summarize several packet filtering techniques that achieve this goal.

Ingress Filtering

Incoming packets to a network domain can be filtered by ingress routers. These filters verify the identity of packets entering into the domain, like an immigration security system at the airport. Ingress filtering, proposed by Farguson and Senie [61], is a restrictive mechanism that drops traffic with IP address that does not match a domain prefix connected to the ingress router. As an example, in Figure 3.2, the attacker $A1$ resides in domain $D1$ with the network prefix a.b.c.0/24. The attacker wants to launch a DoS attack to the victim V that is connected to domain $D4$. If the attacker spoofs the IP address of host $H5$ in domain $D5$, which has the network prefix x.y.z.0/24, an input traffic filter on the ingress link of $R1$ will thwart this spoofing. $R1$ only allows traffic originating from source addresses within the a.b.c.0/24 prefix. Thus, the filter prohibits an attacker from using *spoofed* source addresses from outside of the prefix range. Similarly, filtering foils DDoS attacks that employ reflectors. In Figure 3.2, ingress filter of $D2$ will discard packets destined to the reflector $H3$ and specifying $V's$ address in the source address field. Thus, these packets will not be able to reach the reflector.

Ingress filtering can drastically reduce the DoS attack by IP spoofing if *all* domains use it. It is hard, though, to deploy ingress filters in *all* Internet domains. If there are some unchecked points, it is possible to launch DoS attacks from that points. Unlike ingress filters, egress filters [80] reside at the exit points of a network domain and checks whether the source address of exiting packets belong to this domain. Aside from the placement issue, both ingress and egress filters have similar behavior.

Route-based Filtering

Park and Lee [81] propose route-based distributed packet filtering, which rely on route information to filter out spoofed IP packets. For instance, suppose that $A1$ belongs to domain $D1$ and is attempting a DoS attack on V that belongs to domain $D4$. If $A1$ uses the spoofed address $H5$ that belongs to domain $D5$, the filter at domain $D1$ would recognize that a packet originated from domain $D5$ and destined to V should not travel through domain $D1$. Then, the filter at $D1$ will discard the packet. Route-based filters do not use/store individual host addresses for filtering, rather, they use the topology information of Autonomous Systems (ASes). The authors of [81] show that with partial deployment of route-based filters, about 20% in the Internet AS topologies, it is possible to achieve a good

fi ltering effect that prevents spoofed IP fbws reaching other ASes. These fi lters need to build route information by consulting BGP routers of different ASes. Since routes on the Internet change with time [82], it is a challenge for route-based fi lters to be updated in real time.

Finally, all fi lters proposed in the literature so far fall short to detect IP address spoofi ng from the domain in which the attacker resides. For example, in Figure 3.2, if $A1$ uses some unused IP addresses of domain $D1$, the fi lters will not be able to stop such forged packets to reach the victim V.

3.3 Monitoring to Detect Service Violations and DoS Attacks

In this section, we discuss how network monitoring techniques can be used to detect service violations and to infer DoS attacks. This discussion is necessary for the comparative evaluation presented in Section 3.4. Each monitoring scheme is discussed in details in Chapter 2.

We believe that network monitoring has the potential to detect DoS attacks in early stages before they severely harm the victim. Our conjecture is that a DoS attack injects a huge amount of traffi c into the network, which may alter the internal characteristics (e.g., delay and loss ratio) of the network. Monitoring watches for these changes and our proposed techniques can identify the congested links and the points that are feeding them. We describe the monitoring schemes in the context of a QoS-enabled network, which provides different classes of service for different costs. The schemes are also applicable to best effort (BE) networks to infer DoS attacks, but not to detect service violations because there is no notion of service differentiation in BE networks.

To monitor a domain, we measure three parameters: delay, packet loss ratio, and throughput. We refer to these parameters collectively as the service level agreement (SLA) parameters, since they indicate whether a user is achieving the QoS requirements contracted with the network provider. In our discussion, delay is the end-to-end latency; packet loss ratio is defi ned as the ratio of number of dropped packets from a fbw to the total number of packets of the same fbw entered the domain; and throughput is the total bandwidth consumed by a fbw inside the domain. A fbw can be a micro fbw with fi ve tuples (addresses, ports, and protocol) or an aggregate one that is combined with several micro fbws. Delay and loss ratio are good indicators for the current status of the domain. This is because, if the domain is properly provisioned and no user is misbehaving, the fbws traversing through the

domain should not experience high delay or loss ratio inside that domain. It is worth mentioning that delay jitter, i.e., delay variation, is another important SLA parameter. However, it is flow-specific and therefore, is not suitable to use in network monitoring.

The SLA parameters can be estimated with the involvement of internal (core) routers in a network domain or can be inferred without their help. We describe both *core-assisted* monitoring and *edge-based* (without involvement of core routers) monitoring in the following subsections.

3.3.1 Core-assisted Monitoring

A core-assisted monitoring scheme for QoS-enabled network is studied in Chapter 2. In this scheme, the delay is measured by having the ingress routers randomly copy the header of some of the incoming packets. The copying depends on a pre-configured probability parameter. The ingress router forms a probe packet with the same header as the data traffic, which means that the probe packet will likely follow the same path as the data packet. The egress router recognizes these probe packets and computes the delay.

This monitoring scheme measures the loss ratio by collecting packet drop counts from core routers. It then contacts the ingress routers to get the total number of packets for each flow. The loss ratio is computed from these two numbers. To measure the throughput, the scheme polls the egress routers. The egress routers can provide this information because they already maintain this information for each flow. This scheme imposes excessive overhead on the core routers, therefore, it is not scalable. Other monitoring schemes that involve both core and edge routers are proposed in the literature, see for example [44, 45, 47].

3.3.2 Edge-based Monitoring

We describe two edge-based monitoring schemes: *stripe-based* and *overlay-based*. Both schemes measure delay and throughput using the same techniques as the previous core-assisted scheme. They differ, however, in measuring the packet loss ratio.

Stripe-based Monitoring. The stripe-based scheme infers loss ratio inside a domain without relying on core routers. We show how to infer loss ratios for unicast traffic as explained in [29] and refer the reader to [50] for the multicast traffic case. The scheme sends a series of probe packets, called a stripe, with no delay between them. Usually, a stripe consists of three packets. To simplify the discussion, consider a two-leaf binary tree spanning nodes

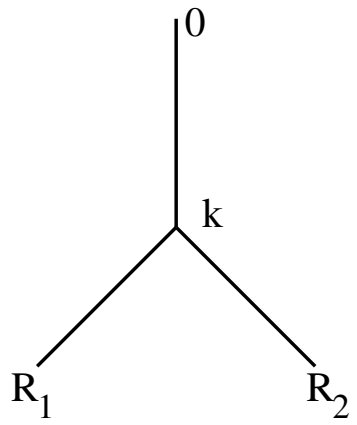
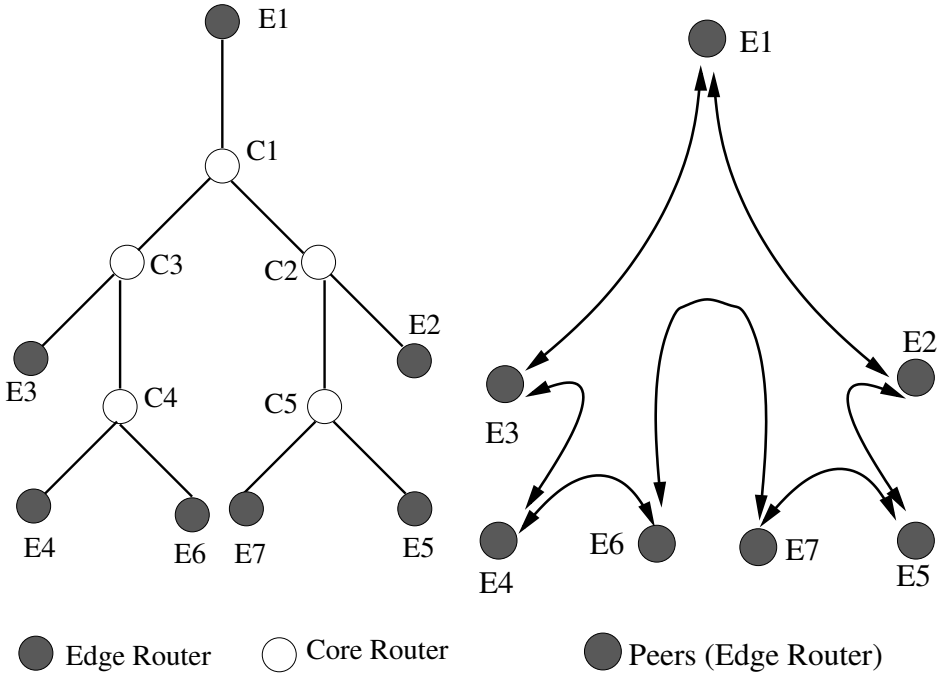


Fig. 3.3. Inferring loss ratio from the source 0 to receivers R_1 and R_2.

0, k, R_1, R_2, as shown in Figure 3.3. The loss ratio of the link $k \rightarrow R_1$, for instance, can be estimated by sending stripes from the root 0 to the leaves R_1 and R_2. The first packet of a 3-packet stripe is sent to R_1, while the last two are sent to R_2. If a packet reaches to any receiver, we can infer that the packet must have reached the branching point k. Further, if R_2 gets the last two packets of a stripe, it is likely that R_1 receives the first packet of that stripe. The packet loss probability is calculated based on whether all packets sent to R_1 and R_2 reach their destination. Similarly, the loss ratio of the link $k \rightarrow R_2$ is inferred using a complementary stripe, in which the first packet is sent toR_2 and the last two are sent to R_1. The loss ratio of the common path from $0 \rightarrow k$ can be estimated by combining the results of the previous two steps. For general trees, this inference technique sends stripes from the root to all ordered pairs of the leaves of the tree. Finally, this technique is extended in for routers with active queue management in a QoS domain.

Overlay-based Monitoring. The overlay-based monitoring approach is proposed in Chapter 2 to further reduce the monitoring overhead. In this mechanism, the edge routers of a domain form an overlay network on top of the physical network. Figure 3.4(a) shows the spanning tree of the domain's topology. The edge routers form an overlay network among themselves, as shown in Figure 3.4(b). This overlay is used to build tunnel for probe packets on specified paths. The internal links for each end-to-end path in the overlay network are shown in Figure 3.4(c). In this monitoring approach, an SLA monitor sits at any edge router. The monitor probes the network regularly for unusual delay patterns. The



(a) The spanning tree of the domain's topology.

(b) An overlay constructed over all edge routers.

Fig. 3.4. Overlay-based distributed network monitoring. (a) Tree-like topology (b) The overlay network formed by the edge routers.

delay and throughput measurements are the same as described in *stripe-based* scheme. The two schemes differ in measuring loss. Since service violation can be detected without exact loss values, we need only to determine whether a link has higher loss than the specified threshold or not. The link with high loss is referred to as a congested link. The goal of this monitoring is to detect all congested links.

When delay goes high, the SLA monitor triggers agents at different edge routers to probe for loss. Each edge router probes its neighbors. Let X_ρ be a boolean random variable that represents the output of probe ρ. X_ρ takes on value 1 if the measured loss exceeds the threshold in any link throughout the probe path, and takes on 0 otherwise. For example, if the outcome of $E1 \rightarrow E3$ probing path is 1, it means either $E1 \rightarrow C1$, $C1 \rightarrow C3$, $C3 \rightarrow E3$, or a combination of them is congested. If the outcome is 0, then definitely all internal links are *not* congested. In this way, we write equations to express all internal links in terms of the probe outcomes. Solving these equations and identifying the congested links are detailed in Chapter 2.

The overlay-based monitoring scheme requires less number of total probes, $O(n)$, compared to the stripe-based scheme, which requires $O(n^2)$, where n is the number of edge routers in the domain. This scheme is able to detect violation in both directions of any link in the domain, whereas the stripe-based can detect a violation only if the flow direction of the misbehaving traffic is the same as the probing direction from the root. To achieve same ability, the stripe-based needs to probe the whole tree from several points, which increases the monitoring overhead substantially.

3.3.3 Violation and DoS Detection

In both the *stripe-based* and *overlay-based* monitoring schemes, when delay, loss, and bandwidth consumption exceed the pre-defined thresholds, the monitor decides on possible SLA violation. The monitor knows the existing traffic classes and the acceptable SLA parameters per class. High delay is an indication of abnormal behavior inside the domain. If there is any loss for the guaranteed traffic class and if the loss ratios of other traffic classes exceed certain levels, an SLA violation is flagged. This loss can be caused by some flows consuming bandwidth beyond their SLA. Bandwidth theft is checked by comparing the total bandwidth achieved by a user against the user's SLA for bandwidth. The misbehaving flows are controlled at the ingress routers.

To detect DoS attacks, set of links L with high loss are identified. For each congested link, $l(v_i, v_j) \in L$, the tree is divided into two subtrees: one formed by leaves descendant from v_i and the other from the leaves descendant from v_j. The first subtree has egress routers as leaves through which high aggregate bandwidth flows are leaving. If many exiting flows have the same destination IP prefix, we can infer that either this is a DoS attack or the traffic is a going to a popular site [68]. Decision can be taken with consulting the destination entity. If it is an attack, we can stop it by triggering filters at the ingress routers that are leaves of the *other* subtree.

We illustrate a scenario of detecting and controlling DoS attack using Figure 3.4. Suppose, the victim's domain $\mathcal{D}$ is connected to the edge router $E6$. The monitor observes that links $C3 \rightarrow C4$ and link $C4 \rightarrow E6$ are congested for a time duration Δt sec. From both congested links, we obtain the egress router $E6$ through which most of these flows are leaving. The destination IP prefix matching at $E6$ reveals that an excess amount of traffic is heading towards the domain $\mathcal{D}$ connected to $E6$. To control the attack, the monitor needs to figure out through which ingress routers the suspected flows are entering into the domain. The monitor activates filters at these ingress routers to regulate the flows that are destined to $\mathcal{D}$.

The advantage of the monitoring-based attack detection is that the neighbor domains of the victim can detect the attack early by observing the violation of SLA parameters. By consulting with the potential victim, these domains can regulate the intensity of the attack and even an early detection can thwart the attack. For each violation, the monitor takes actions such as throttling a particular user's traffic using a flow control mechanism.

3.4 Comparative Evaluation

In this section, we conduct a quantitative analysis of the overhead imposed by different schemes to detect and prevent DoS attacks. The objective of this comparison is to show the characteristics of each scheme and how they behave when different configuration parameters of a domain are changed. We do not emphasize on numeric overhead value of any specific scheme, rather, we draw a relative comparison among them. The comparison provides guidelines for selecting the appropriate scheme, or a combination of schemes, based on the requirements and how much overhead can be tolerated. The schemes we compare here are: Ingress Filtering (*Ingf*), route-based packet filtering (*Route*), traceback with

Table 3.1
Symbols used in the comparison and their values. The parameters defi ne a high speed network domain, where thousands of flows are passing through it.

Symbol	**Description**	**Values used in comparison**
$\mathcal{P}_{sch}$	Processing overhead for scheme sch	–
$\mathcal{C}_{sch}$	Communication overhead for scheme sch	–
M	Number of edge routers	[10 – 20]
N	Number of core routers	12
F	Number of flows entering through each edge router	100,000
P	Number of packets per flow	10
p	Probability to mark a packet	[0 – 0.20]
θ	Percentage of misbehaving flows	[0 – 20%]
h	Path length inside a domain or hop count	4, 6
s	Length of a stripe	3
f_s	Frequency of stripe per unit time in stripe-based monitoring	20
f_d	Frequency of probes per unit time in overlay-based monitoring	30
α_1	Processing overhead for fi ltering	–
α_2	Processing overhead for marking	–
α_3	Processing overhead for monitoring	–

probabilistic packet marking (*PPM*), core-assisted network monitoring (*Core*), stripe-based monitoring (*Stripe*), and overlay-based monitoring (*Overlay*).

3.4.1 Setup

For each scheme, we calculate two different overheads: processing and communication. The processing overhead is due to extra processing required at all routers of a domain per unit time. The communication overhead is due to extra packets injected into a domain. The communication overhead is computed as the number of extra *bytes* (not packets) injected per unit time. For processing overhead, the extra processing at routers may contain: more

address lookups, changing some header fields, checksum re-computation, and any CPU processing needed by the scheme. For example, filters need to check the source IP address to verify whether a packet is coming from a valid source. This requires one extra address lookup (to check the source IP address) for each packet. The monitoring schemes inject probe packets into the network. Each router inside a domain requires processing such as address lookup, TTL field decrement, checksum computation for each probe packet. For simplicity, we charge the filtering schemeα_1 processing units, the marking scheme α_2 processing units, and the monitoring schemes α_3 processing units for each packet processed. We express the processing overhead in terms of α_1, α_2, and α_3 (processing units), and the communication overhead in terms of the total kilobytes (KB) injected in the domain.

We consider a domain $\mathcal{D}$ with M edge routers and N core routers. We assume there are F flows traversing through each edge router and each flow has P packets on average. We define θ as the percentage of misbehaving flows that may cause DoS attacks. We denote $\mathcal{C}_{sch}$ as the communication overhead and $\mathcal{P}_{sch}$ as the processing overhead respectively for scheme sch. Table 3.1 lists the variables used in the comparison and their values.

3.4.2 Overhead Calculation

Filtering and marking techniques do not incur any communication overhead. The monitoring schemes have both processing and communication overhead.

Ingress filtering. The processing overhead of ingress filtering depends on the number of packets entering a domain. It requires one processing unit to check the source IP address of every packet. For our domain $\mathcal{D}$, the total entering packets is $M \times F \times P$. Thus, the total processing overhead of ingress filtering is given by:

$$\mathcal{P}_{Ingf} = M \times F \times P \times \alpha_1. \tag{3.1}$$

Route-based filtering. We need to deploy ingress filters in every domain in the Internet to effectively stop all possible attacks. The route-based filtering scheme, on the other hand, does not require every single domain to have a filter. Park*et al.* show that placing this filter at approximately 20% of all autonomous systems can prevent DoS to a great extent [81]. For a domain that deploys a router-based filter, the overhead is the same as the ingress

filter. Globally speaking, the overhead of route-based filtering is one fifth of the overhead of ingress filtering on the average. In our comparison, we use

$$\mathcal{P}_{Route} = 0.2 \times \mathcal{P}_{Ingf}. \tag{3.2}$$

Probabilistic packet marking (PPM). *PPM* does not incur any communication overhead but adds extra α_2 processing units for every packet that gets marked at an intermediary router. *PPM* might need sophisticated operation such as taking hash of certain IP fields. The traceback with *PPM* marks packets with a probability p at each router on the path to the victim. If a packet passes through h hops, on the average, in the network domain $\mathcal{D}$, the processing overhead is computed as:

$$\mathcal{P}_{PPM} = M \times F \times P \times p \times h \times \alpha_2. \tag{3.3}$$

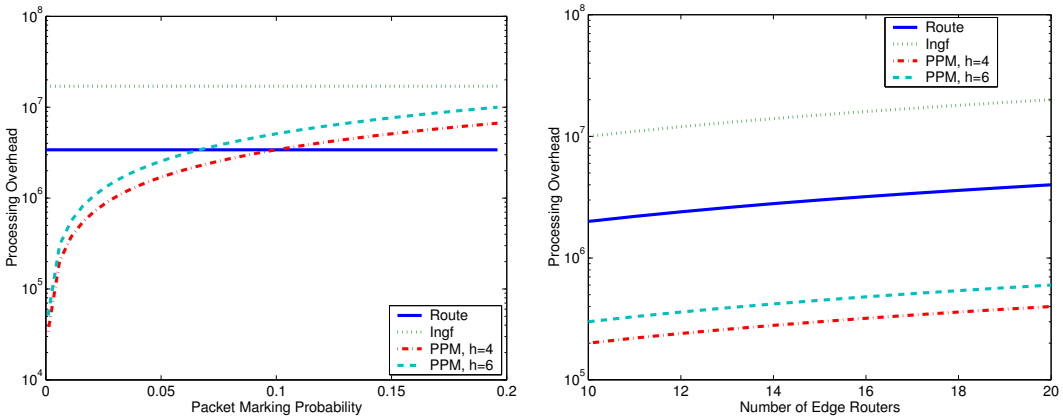
Core-assisted monitoring. The monitoring schemes inject probe traffic into the network and add processing overheads as well. The total number of injected probes and the size of each probe packet are used to calculate the communication overheads in terms of bytes. The *Core* scheme depends on the number of packets that core routers send to the monitor to report drop history. The drop history at each core router depends on the flows traversing the network domain and the percentage of these flows that are violating their SLAs at a particular time. For the domain $\mathcal{D}$, if d bytes are required to record the drop information of each flow, then each core needs to send $C = max(1, \frac{F \times \theta \times d}{packet\_size})$ control packets to the monitor. The $packet\_size$ is the size of a control packet, which depends on the MTU of the network. To obtain *loss ratio,* the monitor queries all edges for packet count information of the misbehaving flows. Every edge replies to this query. The total number of packets exchanged among all edge routers and the monitor is $(2M + N) \times C$ packets. Therefore, the communication overhead is given by:

$$\mathcal{C}_{Core} = (2M + N) \times max(1, \frac{F \times \theta \times d}{packet\_size}) \times packet\_size, \tag{3.4}$$

and the processing overhead is given by:

$$\mathcal{P}_{Core} = (2M + N) \times max(1, \frac{F \times \theta \times d}{packet\_size}) \times h \times \alpha_3, \tag{3.5}$$

where $packet\_size$ is a configurable parameter.



(a) Effect of varying the marking probability on the processing overhead.

(b) Effect of varying the number of edge routers on the processing overhead.

Fig. 3.5. The processing overhead per unit time for filters and probabilistic packet marking (PPM) schemes. Marking scheme has less processing overhead than filtering scheme if the marking probability is not too high (e.g.,$p \leq 0.07$).

Stripe-based monitoring. In the stripe-based monitoring scheme, a stripe of s packets is sent from the monitor to every egress router pairs. For the network domain $\mathcal{D}$, the total number of probing packets is $s \times (M-1) \times (M-2) \times f_s$, where f_s is the frequency by which we need to send stripes per unit time. The communication overhead and the processing overhead are shown in equation (3.6) and equation (3.7) respectively.

$$\mathcal{C}_{Stripe} = s \times (M-1) \times (M-2) \times f_s \times packet\_size, \tag{3.6}$$

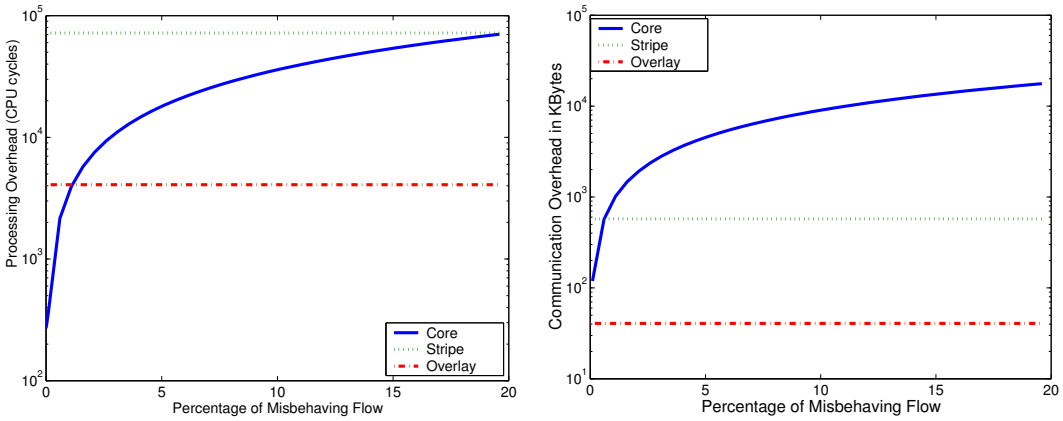
$$\mathcal{P}_{Stripe} = s \times (M-1) \times (M-2) \times f_s \times h \times \alpha_3. \tag{3.7}$$

Overlay-based monitoring. For this monitoring, each edge router probes its left and right neighbors. If it requires f_d probes per unit time, the communication overhead is:

$$\mathcal{C}_{Overlay} = 2 \times M \times f_d \times packet\_size. \tag{3.8}$$

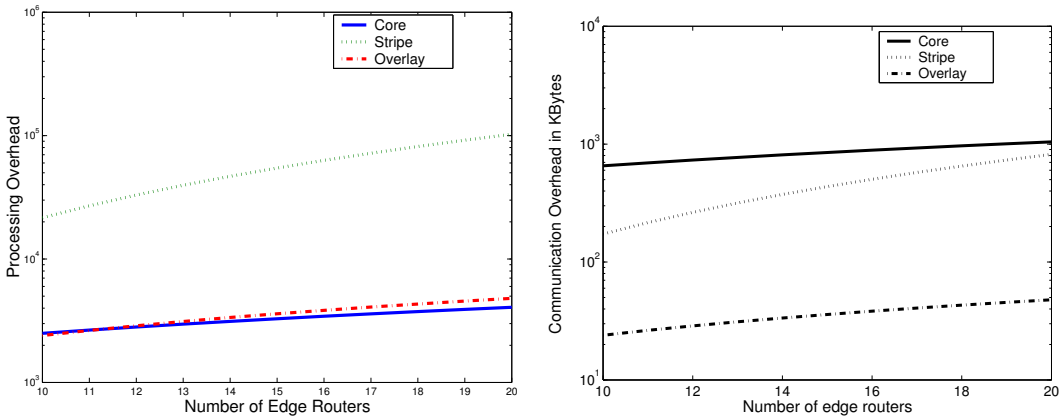
On the average, each probe packet traverses h hops and thus the processing overhead can be calculated as:

$$\mathcal{P}_{Overlay} = 2 \times M \times f_d \times h \times \alpha_2. \tag{3.9}$$



(a) Processing overhead. (b) Communication overhead.

Fig. 3.6. The processing and communication overhead for the monitoring schemes when the percentage of misbehaving flows is increased. The *Core* scheme has less communication overhead than *Stripe* scheme for $\theta < 20\%$. Both *Stripe* and *Overlay* schemes have less communication overhead than *Core* unless θ is very low.



(a) Processing overhead. (b) Communication overhead.

Fig. 3.7. The processing and communication overhead for the monitoring schemes when the number of edge routers in a domain is increased. The *Core* scheme has less processing overhead than both edge-based schemes when the number of edge routers in the domain is increased. Edge-based schemes always impose less communication overhead than the *Core* scheme. The *Core* might perform better than *Stripe* for a large domain (e.g., $M > 20$) depending on the value of θ.

3.4.3 Results and Analysis

To visualize the differences among all schemes, we plot the processing and communication overhead for one of the domain shown in Figure 3.2. Usually, DoS attacks are directed towards a particular host or a set of hosts connected to a relatively small size domain. In the example, Figure 3.2, the DoS attack is directed towards domain $D4$ and the attack traffic is coming from various other domains. For our comparison, we use the parameters' values shown in Table 3.1 for domain D. We use *second* as unit time in all comparisons.

Figure 3.5 (a) shows the processing overhead in terms of α_1 for ingress filtering, route-based filtering, and PPM when packet marking probability is varied along the X-axis. The route-based filtering requires less processing than marking scheme for $p \geq 0.07$ because this filtering scheme does not need to be deployed at all routers of all domains. Savage*et al.* use marking probability $p = 0.04$ in their traceback analysis [72]. Using this probability, the marking mechanism has less overhead than others. We use two different path lengths in the plot; one is $h = 4$ and another is $h = 6$. The path length does not increase the overhead substantially because the path length does not go up very high for a small-size domain. Figure 3.5(b) shows that when number of edge routers are increased in a domain the processing over is increased for all schemes.

Figure 3.6 shows both processing and communication overhead for different monitoring schemes. The processing overhead is low for *Core* scheme than the *Stripe* scheme for $\theta \leq 20\%$. This is because the control packet size of *Core* can be set equal to the maximum transmission unit of the network to minimize total number of packets sent, whereas the probe packet size of the *Stripe* is 20 bytes with 20 bytes of IP header. However, if the attack intensity is high, i.e., the value of θ is high, the overhead of *Core* exceeds the overhead of both edge-based schemes. In this example, probes injected by *Stripe* scheme consumes 600K bytes of bandwidth per sec, which is distributed over all links of the domain. If all links are OC3 type, on average each link experiences probe traffic less than 0.015% of the link capacity. The *Overlay* scheme consumes ten times less than the *Stripe* one in this setup.

In Figure 3.7, we vary the domain size changing the number of edge routers while keeping the number of core routers fixed to $N = 12$. The percentage of misbehaving traffic θ is fixed and equals 1%. Figure 3.7 (a) shows that *Core* can result in less computation overhead than edge-based schemes when the number of edge routers increases. Even though the overhead of *Core* scheme depends on both core and edge routers, this scheme reduces

Table 3.2
Comparison among different schemes to detect and prevent DoS attacks. Some techniques can detect service violation additionally.

Property	*PPM*	*Ingress Filtering*	*Route-based Filtering*	*Core-assisted Monitoring*	*Stripe-based Monitoring*	*Overlay Monitoring*
Overhead depends on	attack volume	number of incoming packets	number of incoming packets	number of flows violating SLAs	routers, topology, attack traffic	routers, topology, attack traffic
Implementation overhead	all routers	all ingress edge routers	all routers of selective domains	all edge and core routers	all edge routers	all edge routers
Clock synchronization	—	—	—	at edge and core routers	at edge routers	at edge routers
Response	reactive	proactive	proactive	reactive	reactive	reactive
SLA violation detection	no	no	no	yes	yes	yes
Detect attacks initiated using	any IP	spoofed IP from other domains	spoofed IP from other domains	any IP	any IP	any IP

processing overhead by aggregating flows when it reports to the monitor. When number of edge routers increases, overhead for both *Core* and *Overlay* schemes increase linearly. The overhead for *Stripe* increases substantially with the increase of edge routers. Depending on θ, Figure 3.7 (b) shows that the communication overhead for *Stripe* may exceed the communication overhead of *Core* when $M > 20$.

3.4.4 Summary

We summarize the important features of all schemes in Table 3.2. Ingress filtering and core-assisted monitoring schemes have high implementation overhead because the former needs to deploy filters at all ingress routers in the Internet and the latter needs support from all edge and core routers in a domain. But filtering and monitoring can provide better safety compared to the traceback which only can identify an attacker after the attack has occurred. All monitoring schemes need clock synchronization to measure SLA parameters, which is an extra overhead. But, they can detect service violations and DoS attacks as well. Filters are proactive in nature and all other schemes are reactive. Filters can detect attacks by spoofed packets whereas the rest of the schemes can detect an attack even if the attacker does not use spoofed IP addresses from other domains.

3.5 Conclusions

We have investigated several methods to detect service level agreement violations and DoS attacks. We showed that there is no single method that fits all possible scenarios. Specifically, in ICMP traceback and probabilistic packet marking mechanisms, the attacker may be able to confuse the victim by sending false ICMP traceback packets and by randomly marking attacking packets. Ingress filters need global deployment to be effective, whereas route-based filters strive against the dynamic change of the routing information.

We have showed that network monitoring techniques can be used to detect service violations by measuring the SLA parameters and comparing them against the contracted values between the user and the network provider. Monitoring techniques have the potential to detect DoS attacks in early stages before they severely harm the victim. Our argument is based on the fact that a DoS attack injects a huge amount of traffic into the network, which may alter the internal characteristics (e.g., delay and loss ratio) of the network. The monitoring techniques watch for these changes and identify the congested links, which helps in locating the attacker and alerting the victim.

The presented comparative study showed several issues. First, it showed that while marking imposes less overhead than filtering, it is only a forensic method. Filtering, on the other hand, is a preventive method, which tries to stop attacks before they harm the system. Second, the core-assisted monitoring scheme has a high deployment cost because it needs to update all edge as well as core routers. However, the core-assisted scheme has less processing overhead than the stripe-based scheme because it aggregates flow information when it reports to the monitor. Third, the stripe-based monitoring scheme has lower communication overhead than the core-assisted scheme for relatively small size domains. For large domains, however, core-assisted may impose less communication overhead depending on the attack intensity. Fourth, the overlay scheme outperforms the other monitoring schemes in terms of deployment cost and overhead in many of the cases.

4. TRAFFIC CONDITIONER FOR SLA ENFORCEMENT

Traffi c conditioner is an important component of the edge routers to mark, shape, and regulate traffi c. We discuss how flow characteristics can be used to improve the conditioner to provide better quality of service (QoS). The characteristics are combined to develop an adaptive and scalable traffi c conditioner. The conditioner is adaptive because the marking algorithm changes based upon the current number of flows traversing through an edge router. If there are a small number of flows, the conditioner maintains and uses state information to intelligently protect critical TCP packets. On the other hand, if there are many flows going through the edge router, the conditioner only uses flow characteristics as indicated in the TCP packet headers to mark without requiring per flow state. The conditioner uses Round Trip Time (RTT) as well as the Retransmission Time-out (RTO) to mitigate TCP bias to short RTT connections.

4.1 Introduction

Designing an edge router that intelligently conditions Assured Forwarding (AF) traffi c has been an active research area. Several studies show that application performance is poor if traffi c conditioning at network edges does not consider transport protocol reaction to drop at the end systems, and dropping behavior at the core routers. Several proposals adjust the marking, dropping, or shaping scheme of a traffi c conditioner based upon TCP connection state. Most of these proposals, however, do not scale well to large numbers of flows. The literature is discussed in related work section of this chapter. In addition, the proposals only consider bulk data applications, and do not examine delay-sensitive traffi c and WWW traffi c.

We study the behavior of transport protocols, and use TCP characteristics to develop an adaptive traffi c conditioner that protects critical TCP packets from drop in order to avoid TCP timeouts. Each conditioner feature is studied individually, and then they are studied in combination. Our conditioner behaves differently based on the number of flows traversing

it. This adaptive design overcomes scalability problems arising from maintaining excessive per flow state.

We investigate how to mitigate the Round Trip Time (RTT) bias on a TCP flow to improve the QoS of a long RTT flow. The throughput of a TCP connection is inversely proportional to the Round Trip Time (RTT) of the connection . Traffic conditioners that mitigate this unfairness by being RTT-aware were first proposed in [83]. These conditioners avoid RTT bias of TCP connections through marking packets with high drop priority inversely proportional to the square of their RTTs according to the steady state TCP behavior. Such conditioners work well when the number of flows is small. We show that for a large number of flows, *short RTT* flows often so far time out because only long RTT flows are protected by the conditioner after satisfying the target rate. Excess bandwidth is mostly given to long RTT flows. To remedy this unfairness introduced by an RTT-aware conditioner, we propose two strategies. The first strategy is to combine the RTT-aware conditioner with techniques that protect a TCP flow when its congestion window is small. The second method is to re-design the RTT-aware conditioner to consider time-outs as well as RTTs to approximate throughput. Both strategies are analyzed for data intensive applications and delay sensitive applications with realistic traffic models.

We note that our method for incorporating RTT-awareness into the conditioner does not grant all available resources to long-RTT connections while short-RTT connections starve. The RTT-awareness only mitigates unfairness in distributing excess bandwidth. When a network is under-provisioned, the RTT-aware conditioner does not consider RTTs.

The performance of the conditioner is analyzed both for data intensive applications and delay sensitive applications in multiple-domain and variable delay configurations.

4.2 Basics of a Conditioner

The conditioner may re-mark a traffic stream or discard or shape packets to alter the temporal characteristics of the stream and bring it into compliance with a traffic profile specified by the network administrator [5]. Figure 4.1 shows different components of a traffic conditioner. We describe each component below:

Meter and Marking Strategy. The meter measures and sorts the classified packets into precedence levels. The marking, shaping, or dropping decisions are based on the measurement result. If C denotes the capacity of bottleneck link of a network system, F

denotes set of flows going through an edge router and R_i denotes the reservation rate of flow i then the excess bandwidth E at time t is expressed as shown:

$$E = C - \sum_{i \in F} R_i. \tag{4.1}$$

A marker distributes the excess bandwidth using a predefined algorithm. This marking can be deterministic or probabilistic depending on the policy taken by the provider. A probabilistic marker gets the current flow rate, *measuredRate*, of a user from the meter and puts a tag on each packet based on the *targetRate* from the service level agreement and the current flow rate. An incoming packet is marked as *IN* profile (low probability to drop) if the corresponding flow does not reach the target rate otherwise the packet is marked with probability $1 - p$, where p is shown in equation (4.2), to a higher drop precedence.

$$p = \frac{measuredRate - targetRate}{measuredRate}. \tag{4.2}$$

Shaping/Dropping Traffic. Shaping reduces the traffic variation and makes it smooth. It provides an upper bound for the rate at which the flow traffic is admitted into the network. A shaper usually has a finite-size buffer. Packets may be discarded if there is not sufficient space to hold the delayed packets. Droppers drop some or all of the packets in a traffic stream in order to bring the stream into compliance with the traffic profile. This process is known as *policing* the stream.

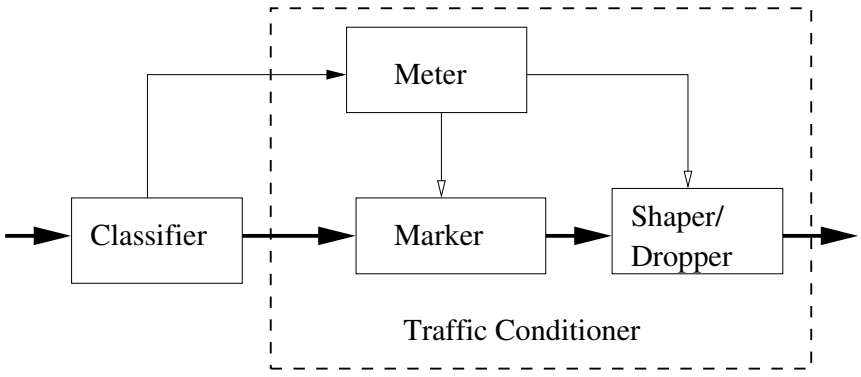


Fig. 4.1. Components of a traffic conditioner to meter, mark, shape, and/or drop incoming packets.

4.3 Related Work

Ibanez and Nichols [19] used a token bucket marker for Assured Service and showed that target rates and TCP/UDP interaction are key factors in determining throughput of flows. The TCP response to packet loss is the main problem. Seddigh, Nandy and Pieda [84] showed that the above mentioned factors are also critical to the distribution of excess bandwidth in an over-provisioned network. Lin, Zheng and Hou [85] proposed an enhanced TSW profiler, but their solution requires state information to be maintained at core routers, which does not scale well.

The Single Rate Three Color Marker (srTCM) [86] and Two Rate Three Color Marker (trTCM) [87] are two basic markers applicable to three drop precedences. srTCM meters an IP packet stream and marks its packets either green, yellow, or red using one Committed Information Rate (CIR) and two associated burst sizes, a Committed Burst Size (CBS) and an Excess Burst Size (EBS). The trTCM marks based on two rates, PIR and CIR. These markers can be used as a component of a traffic conditioner. Bonaventure and Cnodder [88] propose rate adaptive shaper in combination with srTCM and trTCM to improve the performance of TCP by reducing the burstiness of the traffic. With TCP traffic, this reduction of the burstiness is accompanied by a reduction of the number of marked packets and by an improved TCP throughput. Fang, Seddigh and Nandy [89] proposed the Time Sliding Window Three Color Marker (TSW3CM), which we use as a standard conditioner.

Yeom and Reddy [90] pass the marking information to the sender, so that a sender can slow down its sending rate in the case of congestion. This requires modifying the host TCP implementation. They also use three drop precedences IN, OUT-IN and OUT-OUT to provide better QoS. Storing and searching per flow information at the border router for a large number of flows may, however, not scale well.

Feroz *et al.* [91] propose a TCP-Friendly marker. As TCP applications are influenced by bursty packet loss behavior, they use TCP characteristics to design their marker. The main concept is to "protect small-window flows from packet losses" by marking their traffic IN. Detailed analysis on a good window size threshold (below which a flow is marked as IN) for various situations is not provided in [91]. We investigate different thresholds to identify a small window and analyze how they affect the throughput of flows with different RTTs. We also incorporate the idea of protecting small window flows into one of our RTT-aware traffic conditioner proposals.

Nandy *et al.* extend the TSW marker to design RTT-aware traffic conditioners [83]. The basic idea of this conditioner is to adjust the packet drop rate in relation to the RTT. Hence, the acquired bandwidth for the aggregate becomes less sensitive to RTT. Their conditioner is based on the steady state TCP behavior as reported by Matthis *et al.* [92], i.e., bandwidth is inversely proportional to RTT. Their model does not consider time-outs. However, we observe time-out events when a large number of flows is multiplexed onto a bottleneck.

Andrikopoulos *et al.* [93] emphasizes on fair sharing of bandwidth between TCP and UDP flow in their fair traffic conditioner. They mainly focus on queue management. The fairness capability based on the use of FRED (Flow RED). They propose to use Fair MRED in the border of a DiffServ network. This work is an extension of simple fair marker proposed by Kim [94].

Adaptive packet marker proposed by Feng *et al.* [95] uses a Packet Marking Engine (PME), which can be a passive observer under normal conditions, but becomes an active marker at the time of congestion. The marking rate is adjusted by the throughput. This engine can be source transparent or source integrated. The host TCP reacts to the marked/unmarked packet drop differently using two congestion windows: one for best effort traffic and another for priority traffic. The source integrated technique is hard to deploy.

4.4 Proposed Traffic Conditioner

In this section, we discuss techniques to incorporate in a conditioner to improve performance of applications running on top of TCP. Some of these techniques are (loosely or closely) based on ideas proposed in the literature as cited below, but the techniques not requiring per flow state, the combination of techniques, and the adaptivity of the conditioner to the number of flows have not been previously proposed. Like [89], we use the TSW tagger [27], a rate estimator, and the TSW3CM marker as a standard conditioner [83].

4.4.1 Marking Techniques

We examine the following TCP-adaptive features to protect critical TCP flow by marking them as low probability to drop:

SYN. The first few packets of a TCP flow should not be dropped to allow the TCP congestion window to grow. At the edge router, the first few packets can be identified by their sequence numbers. As the initial TCP sequence number is not known to the conditioner,

the conditioner needs to store it. To avoid storing per flow information at the edge, we propose to give low drop priority only to SYN packets as indicated in the TCP header

Small Window (SW). We protect flows with small window from packet losses by marking them with DP0. This strategy was introduced in [91]. TCP increases the congestion window exponentially until it reaches the slow start threshold, *ssthresh*. The congestion window reduces to 1 or half of the *ssthresh* for timeouts or packet loss respectively. The congestion control algorithms of different flavors of TCP are discussed in [96–98]. We give low drop priority to flows with small congestion window sizes. The calculation of TCP window size needs sequence number of data and acknowledgment (ACK) packets. This technique requires per flow state at the edge router. We use SW when we have state information and use SYN otherwise.

Congestion Window Reduction (CWR). An ECN-Capable TCP may reduce its congestion window due to a timeout, triple duplicate ACKs, or in response to explicit congestion notification (ECN) [99]. TCP sets the CWR flag in the TCP header of the first data packet sent after the window reduction. The CWR bit should not be set on retransmitted packets [100] for greater robustness and against denial of service attacks. Instead, when the TCP data sender is ready to set the CWR bit after reducing the congestion window, it should set the CWR bit on the first new data packet that it subsequently transmits. We propose to give low drop priority for a packet if the CWR bit is set. This avoids consecutive *ssthresh* reductions that lead to poor performance with TCP Reno [101].

Target Rate (TR). The target rate is an important factor in marking. Nandy *et al.* [83] mark DP1 and DP2 only when target rates have been achieved, and marking is inversely proportional to the square of the flow requested rates if proportional sharing of excess bandwidth is required. Another strategy is to mark packets based on the difference between target rate and exponentially averaged input rate of the aggregate in order to improve fairness. We use the first strategy.

Burst. The marker avoids marking high drop priority in bursts to work well with TCP Reno. The shaper avoids burstiness to avoid consecutive packet drops and poor performance. This strategy was introduced in [91].

4.4.2 Avoiding RTT-bias

The RTT-aware traffic conditioner proposed in [83] avoids the TCP short RTT bias through marking packets with high drop priority inversely proportional to the square of

their RTTs. This is based upon the steady state TCP behavior modeled in [92]. Equation (4.3) shows that, in this model, bandwidth is inversely proportional to RTT where MSS is the maximum segment size and p is the packet loss probability:

$$BW \propto \frac{MSS}{RTT\sqrt{p}}. \tag{4.3}$$

The RTT-aware marking algorithm proposed in [83] works well when the number of flows is small because (4.3) accurately represents the fast retransmit and recovery behavior when p is small. We have observed that for a large number of flows, short RTT flows time out because only long RTT flows are protected by the conditioner after satisfying the target rates. Excess bandwidth is mostly given to long RTT flows.

To remedy this situation, we can use one of two strategies. First, we can combine the RTT-aware conditioner with a technique that protects the TCP packets after time-outs. Feroz *et al.* propose the small window protection technique [91], which marks TCP packets with lowest drop priority when the congestion window of TCP is small. TCP grows the congestion window exponentially until it reaches the slow start threshold, *ssthresh*. The congestion window reduces to 1 or half of the *ssthresh* for time-outs or packet loss, respectively. Giving low drop priority to flows with small congestion window sizes helps these flows to achieve high throughput.

We have already discuss and analyze the performance of Small Window (SW) based conditioning earlier in this chapter. With SW, a packet is marked as $DP0$ when the congestion window size of a particular flow is $< k$. We show that SW protects the short RTT flows when an RTT-aware conditioner is used. This combination eliminates the unfairness of the basic RTT-aware conditioner for a large number of flows. The RTT-aware marking algorithm with SW is referred to as RTT-SW in this work.

The second approach to eliminate unfairness is to use the throughput approximation by Padhye *et al.* [102], which considers time-outs. Equation (4.4) shows this approximation, where b is the number of packets acknowledged by a received ACK, and To is the time-out length:

$$BW \approx \frac{1}{RTT\sqrt{\frac{2bp}{3}} + To \times min(1, 3\sqrt{\frac{3bp}{8}})p(1+32p^2)}. \tag{4.4}$$

If we take $b = \frac{3}{2}$ (one delayed ACK for two packets for every three incoming packets), approximate $min(1, 3\sqrt{\frac{3bp}{8}})$ to 1 (so that BW will be less than or equal to the right side

of the (4.5), and discard the higher order term of p, i.e., $32p^3$ (if $p \ll 1$, $p^3 \approx 0$), we can simplify (4.4) to:

$$BW \approx \frac{1}{RTT \times \sqrt{p} + To \times p}. \tag{4.5}$$

Designing an RTT-aware traffi c conditioner using (4.5) is more accurate than using (4.3). Consider two fbws with achieved bandwidths BW_1 and BW_2. The objective is to obtain:

$$BW_1 = BW_2. \tag{4.6}$$

(4.5) and (4.6) give:

$$RTT_1 \times \sqrt{p_1} + To_1 \times p_1 = RTT_2 \times \sqrt{p_2} + To_2 \times p_2. \tag{4.7}$$

Let $\rho = \frac{p_2}{p_1}$. Equation (4.7) can then be written as:

$$RTT_1 \times \sqrt{p_1} + To_1 \times p_1 = RTT_2 \times \sqrt{\rho \times p_1} + To_2 \times \rho \times p_1. \tag{4.8}$$

Solving for ρ, this means that we should have:

$$\rho \propto \left(\frac{RTT_1}{RTT_2}\right)^2. \tag{4.9}$$

And:

$$\rho \propto \frac{To_1}{To_2}. \tag{4.10}$$

Equations (4.9) and (4.10) show that the packet drop ratio between two fbws depends on the square of ratio of RTT of the two fbws and the ratio of their time-outs. We combine the two equations to obtain the following heuristic:

$$\rho^2 = \left(\frac{RTT_1}{RTT_2}\right)^2 \times \frac{To_1}{To_2}. \tag{4.11}$$

We follow the same steps as in [83] to derive the marking probabilities. If measured rate is beyond the target rate of a fbw, it marks the packet as $DP1$ or $DP2$ with probability $\frac{measuredRate - targetRate}{measuredRate}$. The ratio of $DP1$ and $DP2$ marked packets is directly related to the packet drop probabilities at the core. This means that packet drop at the core is proportional to the out-of-profi le marked packets. Thus, (4.11) is used to marked packets as $DP1$ and $DP2$. The resulting algorithm, which we refer to as the RTT-RTO algorithm, is shown in Figure 4.2. Note that for the fbw with minimum RTT and RTO, the packets are marked

Algorithm: *RTT-RTO aware Conditioner()*
If measuredRate $<=$ targetRate
  mark packets as DP0
Else
  mark packets as DP0 with probability (1-$\mathbf{p}^2$)
  If packet is not marked DP0
    mark packets as DP1 with probability (1-**q**)
    mark packets as DP2 with probability **q**
where **p** and **q** are:
$\mathbf{p} = \frac{(measuredRate - targetRate)}{measuredRate}$
$\mathbf{q} = (\frac{minRTT}{aggregateRTT})^2(\frac{minRTO}{aggregateRTO})$

Fig. 4.2. An RTT-RTO aware Traffi c Conditioner with three drop precedences.

based on the ratio of its own RTT and RTO. Otherwise, the right hand side of (4.11) may become 1, and all packets of the flow with minimum RTT will be marked as $DP2$, which will deteriorate the performance of the flow.

4.4.3 Developing Scalable Conditioner

Each of the techniques described in Section 4.4.1 and 4.4.2 has advantages and limitations. SYN, CWR, and aggregate Target Rate do not need to store per flow information and are simple to implement. On the other hand, SW, Target Rate based on individual information, Burst, and RTT-RTO need to maintain and process per flow information. Storing and processing excessive state information about each micro-flow at the edge routers does not scale well. To overcome scalability problem, the edge router will store per flow information for a certain number of flows based on its available resources. If its per flow state table for N flows is full, the router overwrites previous entries. N here is a router confi guration parameter that depends on router memory size. We choose to maintain state for the N most recent flows, thus implementing a least recently used (LRU) replacement strategy. For an incoming flow, conditioning will be done based on state information if it is present. If there

```
Algorithm: AdaptiveConditioner()
begin
  for each incoming flow do
    if there is a state entry for this flow then
      statePresent = TRUE
      Update the state table to reflect recent information
    else
      statePresent = FALSE
      Add the flow in the state table, overwrite if table is full
    end if
    if statePresent is TRUE then
      Use Standard Traffic Conditioner with SYN, CWR, SW, Burst, RTT, and RTO
    else
      Use Standard Traffic Conditioner with SYN and CWR
    end if
  end for
end
```

Fig. 4.3. Algorithm for Adaptive Traffic Conditioner. Flow characteristics are used when they are available. Otherwise, flow independent conditioning is conducted.

is no state present, conditioning will be done using techniques that do not need per flow information. In this way, the router does not handle state information beyond its capabilities and achieves scalability.

4.5 Simulation Setup

We use the ns-2 simulator [103] for our experiments. For the standard DiffServ implementation, we use software developed at Nortel Networks [104]. We use the combination of TSW tagger [27], a rate estimator, and the TSW3CM marker [89] to refer as a standard conditioner.

4.5.1 Topologies and Configuration Parameters

The simple network topology shown in Figure 4.4 is used to show the basic marking problems with RTT-aware conditioners, and how RTT-RTO and RTT-SW can overcome these problems. We use the multiple domain topology in Figure 4.5 with cross traffic to illustrate more realistic scenarios. Each edge router is connected to a host which sends aggregate flows to simulate different users. The RED parameters $\{min_{th}, max_{th}, P_{max}\}$ used are: for DP0 $\{40, 55, 0.02\}$; for DP1 $\{25, 40, 0.05\}$; and for DP2 $\{10, 25, 0.1\}$ as suggested in [83]. w_q is set to 0.002 for all REDs. TCP New Reno is used with a packet size of 1024 bytes and a maximum window of 64 packets. We use 200 micro-flows (where a micro-flow represents a single TCP connection) per aggregate flow.

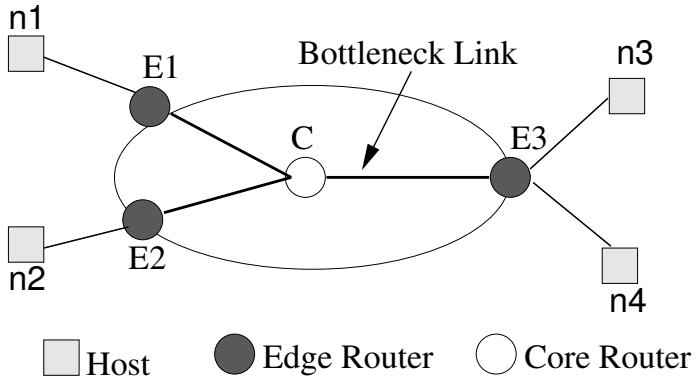


Fig. 4.4. Simple topology to evaluate the basic marking principles by simulation.

4.5.2 Performance Metrics

The metrics we will use to evaluate performance are:

Throughput. This denotes the average bytes received by the receiver application over simulation time. A higher throughput usually means better service for the application (e.g., smaller completion time for an FTP flow). For the ISP, higher throughput is preferable because this means that links are well-utilized.

Packet Drop Ratio. This is ratio of total packets dropped at the core to the total packets sent. A user can specify for certain applications that packet drop should not exceed a

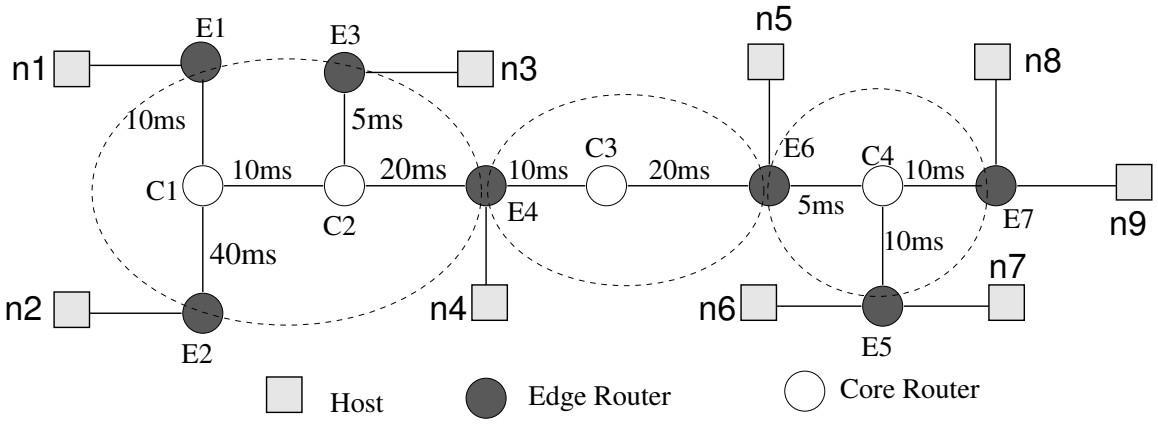


Fig. 4.5. Simulation topology with multiple domains. All links are 10 Mbps.

threshold. This is a metric for both ISP and user. Lower drop reduces bandwidth and other resources wastage on upstream links.

Packet Delay. For delay sensitive application like Telnet, the packet delay time is a user metric. We use this metric to show that user can be benefi ted using traffi c conditioner for Telnet type application.

Response Time. This is the time between sending a request to a web server and receiving the response back from the server.

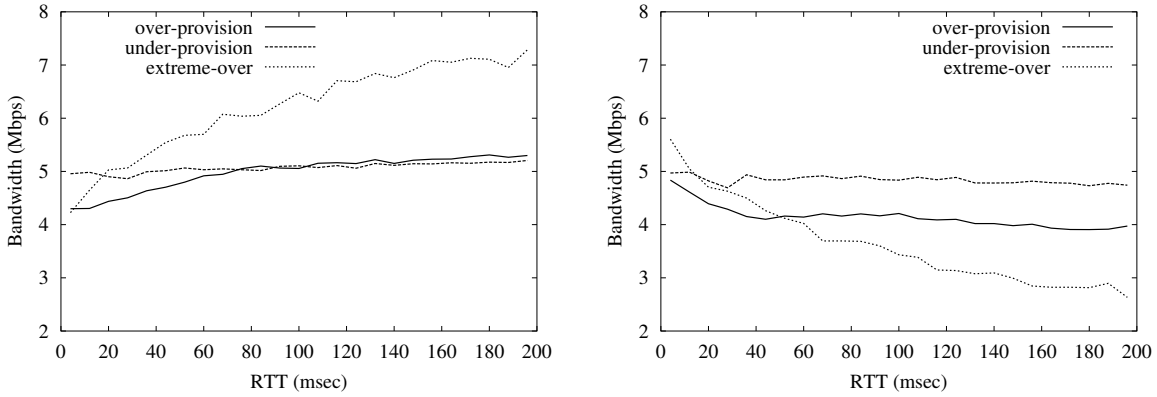
4.6 Simulation Results

We study the behavior of the standard traffi c conditioner and each marking technique individually and in combination. We also study the performance of the proposed adaptive traffi c conditioner with FTP, Telnet and WWW applications. Network hosts and routers are ECN-enabled for all experiments.

4.6.1 Marking Techniques

The objective of our fi rst experiment is to study how each marking technique discussed in section 4.4.1 affects the performance of the standard traffi c conditioner individually and collectively. We vary the RTTs, window size for SW, and target rates in this experiment. The output parameters (metrics) are throughput and packet drop ratio. We use the simple

topology in Figure 4.4 where one aggregate flow, Flow 1-3, is created between nodes $n1$ and $n3$ with RTT 20 ms and another aggregate flow, Flow 2-4, is created between nodes $n2$ and $n4$. The RTT of Flow 2-4 is varied from 4 to 200 ms.



(a) Aggregate Flow 1-3

(b) Aggregate Flow 2-4

Fig. 4.6. Throughput for standard traffic conditioner in over, under-provisioned, and extremely over-provisioned networks for 200 flows.

Standard Conditioner. We test the conditioner for both small (10 micro flows) and large (200 micro flows) number of flows, in under and over-provisioned networks. All flows have the same target rate. For the over-provisioned case, the committed rate CIR is 2 Mbps and peak rate PIR is 3 Mbps for each aggregate flow. For the extremely over-provisioned case, CIR is 0.2 Mbps and PIR is 0.3 Mbps, and for the under-provisioned one, CIR is 6 Mbps and PIR is 10 Mbps.

Figure 4.6 shows the achieved bandwidth for the under, over, and extremely over-provisioned network cases as RTT of Flow 2-4 varies. In the under-provisioned case, both flows achieve close to 5 Mbps, which is a desirable outcome. In the over-provisioned cases, small RTT connections are favored. For example, Flow 1-3 is favored at the expense of Flow 2-4 when its RTT is lower than the RTT of Flow 2-4 (when RTT on the X-axis is greater than 20 ms). If the network is extremely over-provisioned, we see more unfairness and higher packet drop ratio. This is because TCP connections are very aggressive for the flow with small RTT. Due to the fluctuation of the sending rate, TCP loses more packets. As the RTT of Flow 1-3 is fixed, it has almost the same packet drop ratio throughout the

experiment, but the drop ratio decreases when RTT of Flow 2-4 increases. This is because for higher RTT, TCP can estimate the sending rate more accurately.

SYN. SYN is useful for short-lived connections and high degrees of multiplexing. Even though the bandwidth improvement we observed in experiments (not shown) is insignificant (200 kbps for the total), SYN can be used when other expensive techniques (in terms of complexity to deploy) cannot be used. This technique is used in the adaptive conditioner when there is no state information about the flow.

Small Window(SW). Small window works both for small and large number of micro flows as well as short and long lived flows. To study the effect of the window size, k, on achieved bandwidth on both flows, k is varied from 3 to 10. If the window size of a flow is less than k, the flow packets are marked DP0. We have observed that the larger value of k helps the (more aggressive) small RTT connection (Flow 1-3) to achieve more bandwidth at the expense of the large RTT flow (Flow 2-4) due to the preferential drop at the core. This contrast is clearer in an under-provisioned network. The total achieved bandwidth is higher than the standard conditioner and is close to the link capacity. Thus, SW significantly improves utilization. The choice of k depends on policy. A higher value of k such as 7 or 8 may favor short RTT flows and result in more unfairness against long RTT flows, while a lower value of k (e.g., 3) avoids this problem.

Congestion Window Reduction (CWR). Giving priority to CWR packets helps the growth of the congestion window after reductions and reach equilibrium. Results show that CWR helps Flow 2-4 to achieve higher throughput. Flow 1-3 sometimes times out and has high packet drop.

Burst. Avoiding bursty marking and shaping packet bursts improves achieved bandwidth over the standard traffic conditioner. The improvement is more significant for both flows when RTT is low. Flow 2-4 achieves its highest bandwidth in an over-provisioned network when Burst and CWR are combined for low RTT. The "Burst" technique exhibits the lowest packet drop ratio for both flows among other techniques when each is studied separately.

Target Rate. We use a target aware traffic conditioner to divide excess bandwidth in an over-provisioned network in proportion to the subscribed target rates [83]. This feature has no effect in cases of congestion.

Combinations and Overall Performance. Figure 4.7 compares different marking techniques in separate simulation runs. From the figure, it is clear that the "small win-

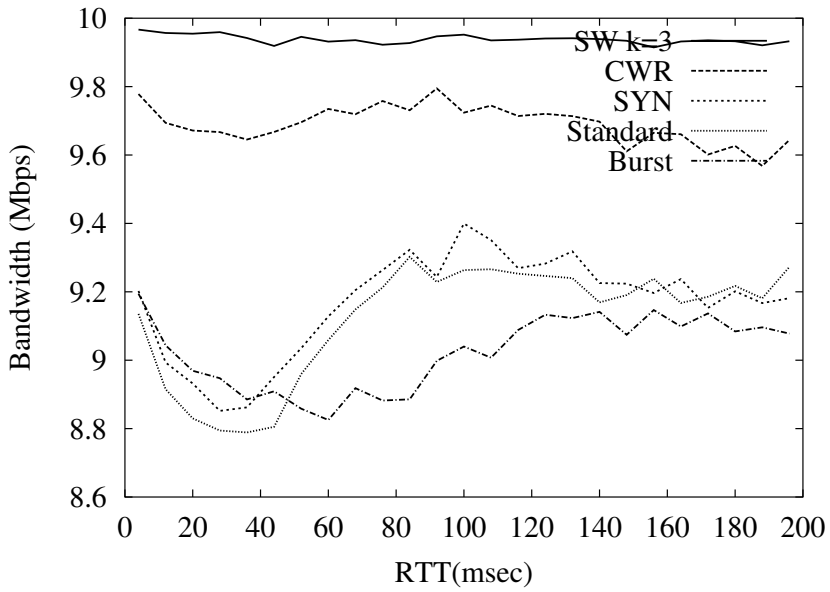


Fig. 4.7. Throughput comparison of the standard traffic conditioner and various marking techniques with 200 flows.

dow" technique contributes most to total bandwidth gain, followed by CWR and SYN. SW favors short RTT connections (Flow 1-3), but it reduces packet drop ratio and timeouts for Flow 2-4 as well, compared to the standard traffic conditioner. "Burst" is effective for short RTT (less than 40 ms). If SW is not used, Burst+CWR achieves higher bandwidth than any other combination. Although SW works better than any other technique alone, using all design techniques together has advantages over SW alone.

As previously discussed, in an under-provisioned network SW increases the throughput of Flow 1-3 at the expense of Flow 2-4. Fluctuations occur when RTT is relatively low for both connections. The fluctuations can be overcome by using the Burst technique. CWR helps Flow 2-4 to achieve more bandwidth as before.

4.6.2 RTT-aware Traffic Conditioners

As previously mentioned, we have observed that a basic RTT-aware conditioner (with both 2 and 3 Drop Precedences) as in [83] is biased when a large number of flows is being multiplexed. Using the same experimental setup as the previous experiment, we observe that Flow 2 (the longer RTT flow) obtains most of the extra bandwidth after target rates

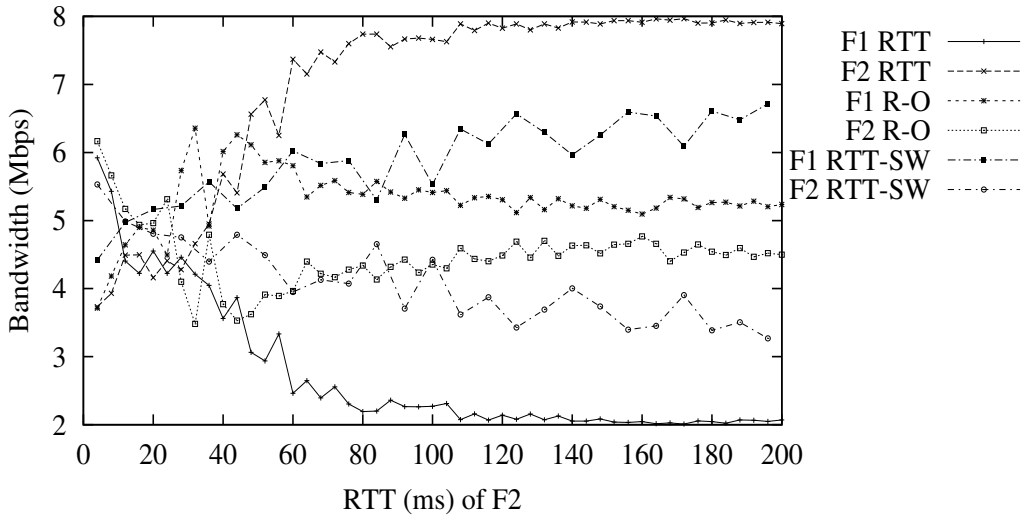


Fig. 4.8. Throughput comparison of basic RTT, RTT-RTO (R-O), and RTT-SW based conditioners. RTT of F1 is 20 ms and RTTs of F2 is shown on the X-axis.

have been satisfied for both aggregates. Figure 4.8 shows that Flow 1 achieves only 2.3 Mbps whereas Flow 2 gets 7.52 Mbps (at Flow 2 RTT=100 ms) with the basic RTT-aware conditioner.

We trace the reason for this behavior to the fact that Flow 2 gets priority over Flow 1 due to its longer RTT, after target rates are satisfied. As a result, many micro flows in the aggregate Flow 1 time-out, and Flow 1 cannot achieve more than its target rate. Figure 4.9 shows that the congestion window (*cwnd*) of a randomly selected micro flow in the Flow 1 aggregate remains small due to timeouts. The figure also shows that incorporating small window protection overcomes this problem.

Figure 4.8 illustrates that our proposed RTT-RTO (R-O) based conditioner (as well as the incorporation of small window protection into the RTT-aware conditioner (RTT-SW)) mitigate this RTT-based unfairness. This is because with a larger number of flows, the per micro flow bandwidth share is small and thus the steady-state *cwnd* is reduced. When *cwnd* is small, there is a higher probability of timeouts in the case of packet drops. Protecting packets (via DP0 marking) when the window is small reduces time-outs, especially back-to-back time-outs. The micro flow also recovers from timeouts when RTO as well as RTT is used to mark packets. The fairness is also improved.

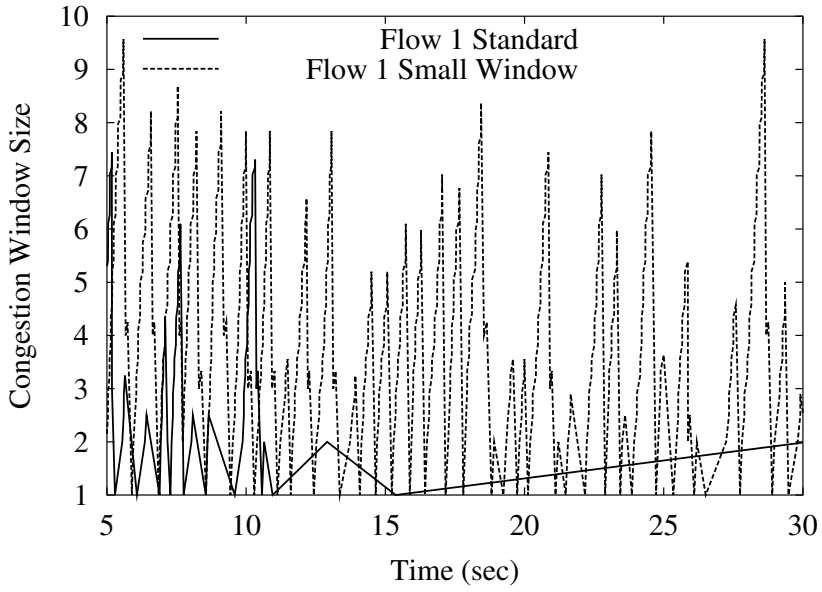


Fig. 4.9. Congestion window size with and without small window protection with RTT-based conditioners for a micro flow of Flow 1.

To examine more realistic scenarios, we use the multiple domain topology shown in Figure 4.5 where flows traverse multiple differentiated services domains. We have created flows $F1$ = $n1$ to $n7$, $F2$=$n2$ to $n8$, $F3$=$n3$ to $n4$, $F4$=$n0$ to $n9$, and $F5$=$n5$ to $n6$. The first two aggregate flows traverse multiple domains while the remaining two act as cross traffic. $F1$ and $F2$ have long RTTs whereas $F3$, $F4$, and $F5$ have short RTTs. Figure 4.10 shows that, with the basic RTT-aware conditioner, $F1$ and $F2$ obtain much higher bandwidth than flows with short RTTs. We discard initial values to reduce transient effects on the result. With the basic RTT-aware conditioner, the excess bandwidth is distributed according to the RTT so that the longer RTT flows get higher share. We do not see this unfairness with the RTT-RTO conditioner or with RTT-SW. With RTT-SW the short RTT flows get much higher bandwidth than long RTT flows. The RTT-RTO based conditioner is fair because long RTT flows do not get higher bandwidth as with the basic RTT-aware conditioner, but also short RTT flows do not steal most of the resources as with RTT-SW. $F1$, $F2$, and $F4$ achieve almost same amount of bandwidth and $F3$ gets little higher, which is fair because this flow has a very short RTT. If the network is extremely over-provisioned, the performance difference is more pronounced. We have observed that flow $F3$ obtains 67 times more

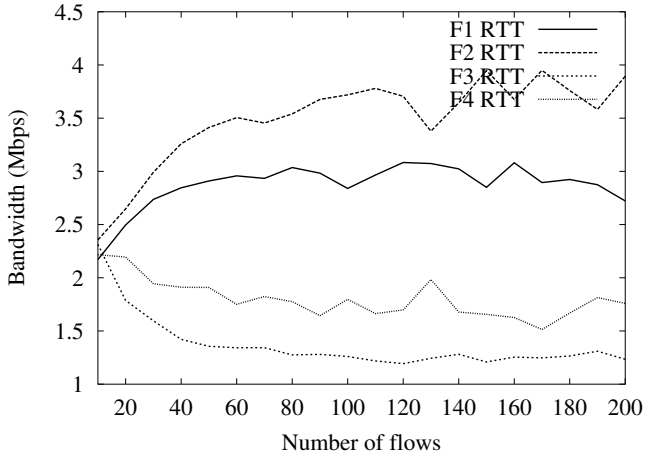
Table 4.1
Per Telnet packet delay (first three columns) and per session delay for Telnet traffic. Number of Telnet sessions = 100.

Conditioner	Delay (sec) F1, F2, F4	Delay (sec) F3, F5	Delay (sec) overall	Delay (sec) / session
Standard	5.36	2.32	3.62	72.11
Basic RTT	5.23	2.18	3.48	69.19
RTT-RTO	5.32	1.98	3.19	68.68
RTT-SW	5.12	1.84	2.89	66.09

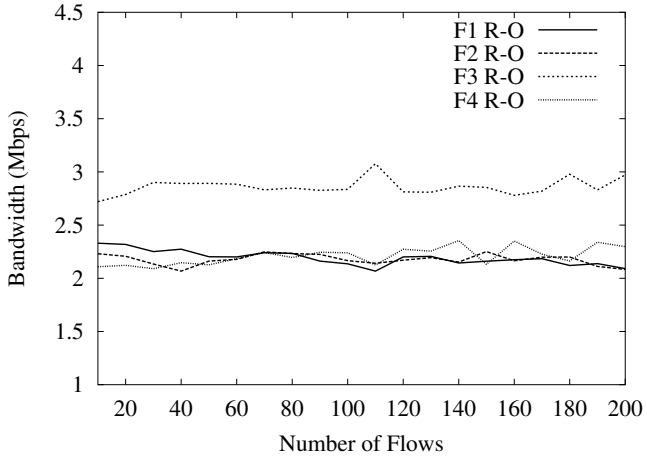
bandwidth than what $F1$ and $F2$ achieved with the standard conditioner, whereas with RTT-RTO the flows achieve very similar bandwidths.

Telnet and WWW Traffic. We compare the performance of Telnet (delay-sensitive) and WWW (response time sensitive) applications with the various RTT-aware conditioner variations. For the Telnet experiment, the metric used is the average packet delay for each Telnet packet. The topology is the same as Figure 4.5, but all links capacities are set to 1 Mbps to induce congestion. We simulate 100 sessions each from node $F1$=$n1 \rightarrow n7$, $F2$=$n2 \rightarrow n8$, $F3$=$n3 \rightarrow n4$, $F4$=$n0 \rightarrow n9$, and $F5$=$n5 \rightarrow n6$. Each session transfers less than 10 to more than 30 TCP packets.

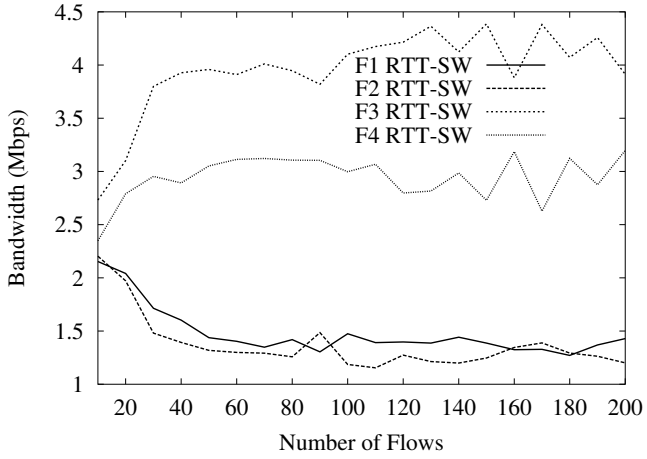
Table 4.1 shows the average packet delay for each Telnet packet. We compare the standard, the basic RTT-aware conditioner, the RTT-RTO conditioner and the RTT-aware conditioner with small window protection (RTT-SW). The delays are long because the network is congested. The standard conditioner has the highest delay for long RTT flows. The RTT-SW has the lowest delay for short RTT flows. This is because with small window protection, short RTT flows get much better service than the long RTT flows. With the RTT-RTO conditioner, the delay for long RTT flows is lower than with the standard and RTT-aware conditioners. In some cases, short RTT flows have higher delay with the RTT-RTO conditioner, which is consistent with the fairness objective of the conditioner. Our experiments show that we can achieve better overall performance with the RTT-RTO conditioner because the delay of long RTT flows is reduced with RTT-RTO aware conditioner



(a) RTT-aware



(b) RTT-RTO



(c) RTT-SW

Fig. 4.10. Throughput of RTT-aware traffi c conditioners in a multiple domain topology (Figure 4.5) for various number of micro-fbws. F1, F2 are long RTT fbws and F3 has very short RTT. F4 is in the middle.

Table 4.2
Response time comparison among different conditioners for WWW traffic. Number of concurrent sessions = 50

Conditioner	Avg response time (sec): first packet	Std dev	Avg response time (sec): all packets	Std dev
Standard	0.75	1.60	2.25	4.79
Basic RTT	0.71	1.52	2.16	4.62
RTT-RTO	0.77	1.64	1.69	3.61
RTT-SW	0.64	1.37	1.80	3.83

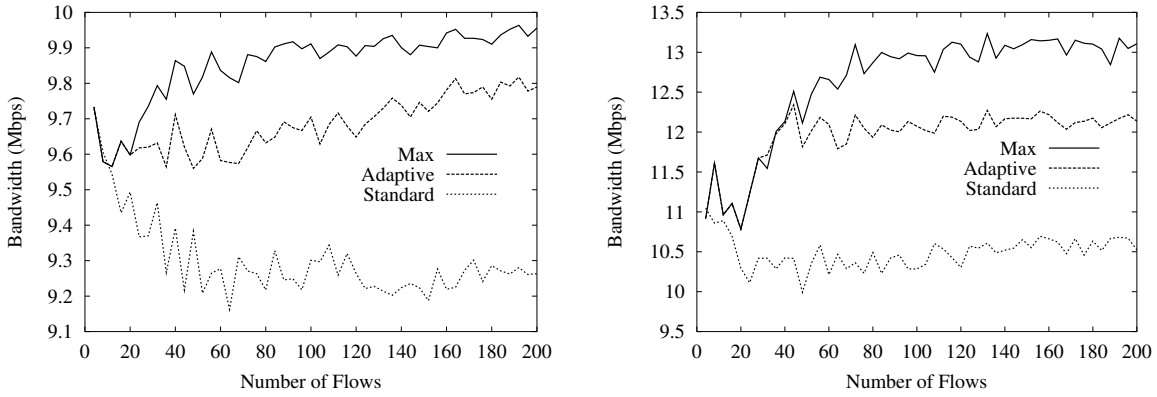
and the overall Telnet packet delay for all flows is minimized. The per Telnet session delay is low with RTT-RTO conditioner.

As web traffic constitutes most (60%-80%) of the Internet traffic, we examine our traffic conditioners with the WWW traffic model in ns-2 [103]. Details of the model are given in [105]. The model uses HTTP 1.0 with TCP Reno. Servers are attached to $n4$, $n7$ and $n8$ of Figure 4.5, while $n1$, $n2$ and $n3$ are used as clients. Each client generates a request for 5 pages with a variable number of objects (e.g., images) per page. We use the default ns-2 probability distribution parameters to generate inter-session time, inter-page time, objects per page, inter-object time, and object size (in kB).

Table 4.2 shows the average response time per WWW request received by the client for 50 concurrent sessions. The network setup is the same as with Telnet traffic. Two response times are shown in the table: one is the time to get the first response packet and another is to get all data. The table shows that the RTT-RTO conditioner reduces total response time over all other conditioners. The RTT-SW conditioner takes less time for the first packet because of the small window protection at the time of connection setup. For 100 concurrent sessions, RTT-RTO conditioner takes the minimum time to get first response. The response time does not differ significantly if the network is not congested.

4.6.3 Adaptive Conditioner

We examine our proposed adaptive conditioner for both the simple and multiple domain topologies. The algorithm used for conditioning is described in Section 4.4.1. Figure 4.11(a) compares achieved bandwidth with the standard, adaptive, and the standard conditioner with all marking techniques at the same time (referred as "Max") for the simple topology (Figure 4.4) with different number of flows. *Max* is the maximum bandwidth achievement by standard conditioner with all existing techniques and using per-flow information. The adaptive conditioner switches the marking techniques based on the availability of state information, however, "Max" has a huge state table so that it can use all marking techniques described in Section 4.4.1 to mark every packet. The adaptive conditioner outperforms the standard one for both aggregate flows. The adaptive conditioner is fair in the sense that Flow 1-3 does not steal bandwidth from flow 2-4, and total achieved bandwidth is close 10 Mbps (bottleneck link speed). Aggregate Flow 2-4 performs better perform with the adaptive conditioner than the performance achieved with "Max."



(a) Simple Topology

(b) Multi-Domain Topology

Fig. 4.11. Achieved bandwidth by the standard conditioner and adaptive conditioner. *Max* is the maximum bandwidth achievement by standard conditioner with all existing techniques and using per-flow information. a) state table size=20 micro-flows b) State table size=50 micro-flows

Figure 4.5 shows our second simulation topology. Three domains are interconnected, with all links being 10 Mbps. We create aggregate flows between nodes $n1 \rightarrow n8$, $n2 \rightarrow n9$, $n3 \rightarrow n4$, $n5 \rightarrow n6$, and $n7 \rightarrow n9$. Flows have very different RTTs and

bottlenecks. Not all flows start/stop transmission at the same time. Short-lived flows last from less than a second to a few seconds. Flows from multiple hosts sometimes traverse the same edge router. $C2 \rightarrow E4$, $E5 \rightarrow C4$ and $C4 \rightarrow E7$ are the busiest links. We measure the total throughput over the simulation time at the receiving end. Figure 4.11(b) shows the total bandwidth gain for this topology comparing different conditioners (note that the solid line here is "Max" and the adaptive uses dotted line). From the figure, the adaptive conditioner works better than the standard one and achieves performance close to the complex conditioner ("Max") and achieves scalability. The adaptive conditioner improved throughput over the standard conditioner, and improves fairness between low and high RTT flows, without requiring large per-micro-flow state tables.

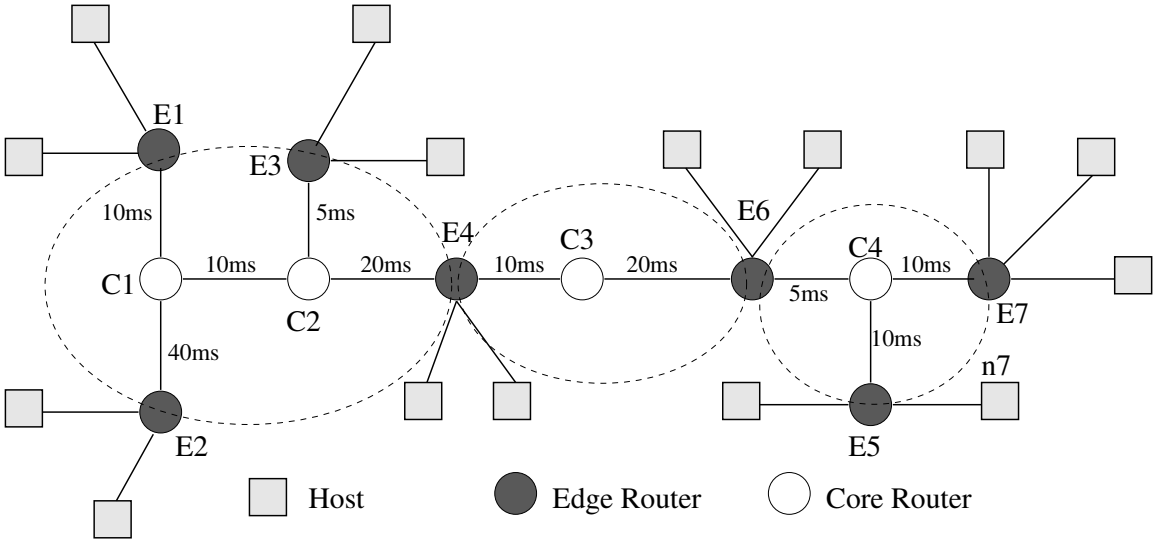


Fig. 4.12. Complex multiple domain topology used for performance evaluation.

Figure 4.12 shows a variation on the multiple domain topology, where many hosts are connected at most edge routers. The link delay between host and the edge is varied from 1 to 10 ms to simulate users at different distances for different hosts. This topology is more realistic and some edges such as $E4$ experience a large number of micro-flows. Since each aggregate flow contains 200 micro flows, the soft state table for the adaptive conditioner covers only a small percentage of the flows passing through it (we use a table for the 50 most recent micro-flows). Table 4.3 shows that the bandwidth achieved with the adap-

Table 4.3
Performance for topology in Figure 4.12. Bandwidth (BW) shown is in Mbps. State table size = 50 micro-flows.

Micro flows	Standard BW gain	Adaptive BW gain	Adaptive (% flows covered at E4)	Max BW gain
10	12.65	12.87	41.16	12.87
50	12.18	13.84	16.66	14.20
100	11.67	13.48	8.33	14.89
200	11.77	13.61	4.16	14.91

tive conditioner is close to the conditioner that uses "Max" techniques, and the adaptive conditioner always outperforms standard conditioner.

Telnet and WWW Traffic. We compare the performance of Telnet (delay-sensitive) and WWW (response time sensitive) applications with the standard conditioner and adaptive conditioner. For the Telnet experiment, the metric used is the average packet delay time for each Telnet micro flow in an aggregate flow. The topology is the same as Figure 4.5, but all links capacities are set to 1 Mbps to introduce congestion. We simulate 200 users each from node $n1 \rightarrow n8$, $n2 \rightarrow n9$, $n3 \rightarrow n4$, $n5 \rightarrow n6$, and $n7 \rightarrow n9$.

Table 4.4 shows the average packet delay of micro-flows from $n1 \rightarrow n8$ with the standard versus adaptive conditioners. The table shows average packet delay and standard deviation per micro-flow, and average delay per TCP packet. Average packet delay in the standard conditioner is higher than the adaptive conditioner. The delays are long because the network is congested. If 10 or 20 flows are used per aggregate flow, both delays are very low. The standard conditioner also shows high standard deviation. The result is similar for other aggregate flows. We have repeated this experiment for 100 micro flows and the outcome is the same: average Telnet packet delay is reduced with the adaptive conditioner.

As web traffic constitutes most (60%-80%) of the Internet traffic, we test our traffic conditioner with the WWW traffic model in ns-2 [103]. (Details of the model are given in [105].) The model uses HTTP 1.0 with TCP Reno. Servers are attached to $n6$, $n8$ and $n9$ of Figure 4.5, while $n1$, $n2$ and $n5$ are used as clients. A client can send a request to any server. Each client generates a request for 5 pages with a variable number of objects (e.g.,

Table 4.4
Micro-flow statistics of aggregate flow $n1$-$n8$ with Telnet traffic. Number of micro-flows = 200.

Conditioner	Avg delay (sec) /micro flow	Std. dev of delay	Total TCP pkts sent	Delay (sec) /TCP pkt
Standard	131.12	115.47	1313	18.77
Adaptive	122.76	107.46	1478	16.61
Max	131.02	116.37	1456	17.81

Table 4.5
Response time of the Adaptive conditioner for WWW traffic. The response time is compared with the maximum possible value. Number of concurrent sessions = 50

Conditioner	Avg response time (sec), first pkt	Std dev	Avg response time (sec), all pkts	Std dev
Standard	0.48	0.17	2.23	0.78
Adaptive	0.45	0.14	2.15	0.75
Max	0.49	0.19	2.15	0.71

images) per page. We use the default ns-2 probability distribution parameters to generate inter-session time, inter-page time, objects per page, inter-object time, and object size (in kB).

Table 4.5 shows the average response time per WWW request received by the client. The network setup is same as with Telnet traffic. Two response times are shown in the table; one is to get the first packet and another is to get all data. The table shows that our conditioner reduces response time over the standard traffic conditioner. The adaptive conditioner does not change the response time significantly if the network is not congested.

4.7 Conclusion

In this chapter, we have discussed techniques to design TCP-aware traffic conditioners and analyzed the performance of the proposed adaptive conditioner for flows with various RTTs and degrees of multiplexing, for over and under-provisioned networks, and for single domain and multiple domain networks. All marking techniques examined improve performance, but the small window (SW) protection contributes the most. Small window, however, appears to mostly favor small RTT flows in our experiments. A lower threshold for the window size reduces this unfairness, without compromising the total bandwidth gain. Congestion Window Reduced (CWR) packet protection favors long RTT flows, while burst avoidance (Burst) is effective when round trip time is small.

We have shown that using a basic RTT-aware traffic conditioner can be unfair by giving all extra bandwidth to long RTT flows when many micro-flows traverse through an edge router. This behavior causes short RTT flows to starve because they frequently time-out and go to slow start. To overcome this unfairness, we present two schemes: one protects flows with small windows, and the other re-designs the conditioner using both RTT and RTO values. Both conditioners are shown to perform well for both small and large numbers of flows. The RTT-RTO conditioner is shown to improve FTP throughput, reduce packet delay for Telnet and response time for WWW traffic.

We design an adaptive conditioner which stores state information based on available resources. When the per-micro-flow state table is full, the conditioner overwrites previous state information based on a least recently used strategy. Marking is based on information in packets, such as SYN and CWR, if state information is not available. The proposed adaptive conditioner has been shown to improve FTP throughput, reduce packet delay for Telnet and response time for WWW traffic.

To avoid RTT-bias, the conditioner requires to determine the RTT of aggregates passing through them. The RTT can be measured by monitoring the flow sequence number in one direction and observing the ACKs in the other direction. This approximation works because the conditioner compares approximate values to each other. It is possible to take a single flow as a representative of the aggregate. As an RTT-aware conditioner also requires the minimum aggregate RTT, the edge routers need to exchange this information. The retransmission timeout can be approximated based on the RTT value using the RTT variance.

We protect flows by giving priority to their critical packets. If a packet is protected (it is re-marked to green when it was yellow or red), the flow profile must still be preserved by marking later packets yellow or red. This ensures that the congestion situation of the network does not deteriorate due to this flow protection.

This conditioner can be used for any architecture that supports service differentiation, or even with active queue management techniques at network routers. For example, the RED algorithm at network routers can itself protect critical packets such as SYN and CWR packets without requiring any additional state. Alternatively, the adaptive conditioner concept can be employed with algorithms like RED. The router would, in this case, store state for the most recently seen flows and use this information to make intelligent dropping decisions.

5. FRAMEWORK OF CONGESTION CONTROL

To avoid a congestion collapse, network flows should adjust their sending rates. Adaptive flows adjust the rate, while unresponsive flows do not respond to congestion and keep sending packets. Unresponsive flows waste resources by taking their share at the upstream links of a domain and dropping packets later when the downstream links are congested.

In this chapter, we devise two algorithms to control unresponsive flows during congestion. The first one takes help from the core routers of Differentiated Services (DiffServ) networks to detect congestion. We describe how core routers detect congestion and inform edge routers about it. The second algorithm uses network tomography—an edge-to-edge mechanism to infer per-link internal characteristics of a domain—to identify unresponsive flows that cause packet drops in other flows. The network tomography is introduced and discussed in Chapter 2. We discuss network tomography briefly in this chapter to clarify how it helps in detecting congestion and unresponsive flows.

We have designed an algorithm to regulate unresponsive flows. The congestion control algorithm is evaluated using both adaptive and unresponsive flows, with sending rates as high as four times of the bottleneck bandwidth, and in presence of short and long-lived background traffic. Our rate control algorithm works well in a variety of situations. The goal of this work is to ensure that TCP does not starve due to unresponsive flows as well as to stop bandwidth waste in the upstream path when packets are dropped in the downstream because of unresponsive flows.

5.1 Introduction

A flow is unresponsive if it fails to decrease its sending rate in response to congestion. During congestion, adaptive flows such as TCP back off and reduce their sending rates. This behavior of TCP prevents congestion collapse in a network. If all flows act in this manner, there should not be any unfairness as well as congestion collapse. However, flows such as UDP send at the same rate even when there exists congestion along the path, be-

cause UDP does not use any feedback mechanism and can not respond to congestion. This behavior may cause TCP flows to starve, and introduces unfairness when various types of flows coexist at the same time in the Internet.

If a packet is dropped at the downstream path, it wastes resources already taken at the upstream. This behavior causes global max-min unfairness [106]. The packets dropped at the bottleneck link have already consumed resources from non-bottleneck links earlier along the path. The unresponsive flows cause this unfairness.

Congestion collapse can be mitigated using improved packet scheduling or active queue management [23–25, 32]. However these techniques can not solve the global max-min unfairness problem, because congestion can be far down along the path, and the upstream queues do not know about this. To solve both problems, we need a mechanism to ensure that the rate at which packets are entering into a network domain should be the same as the rate at which packets are leaving from the domain. We use the DiffServ architecture [5] to address this issue. The DiffServ framework uses edge routers at the border of a network domain and core routers inside the domain.

We design two frameworks to detect and control congestion. The first framework takes the advantage of the core routers to report about congestion to the edge routers. Dropping highest priority packets of each class exhibits that a network is congested [33]. This congestion drop is sent to ingress routers to regulate unresponsive flows. The drops due to shaping at the ingress routers is propagated to egress routers of previous domain to regulate an unresponsive flow at the upstream path. We use this framework to control unresponsive flows, and refer to it as Core-assisted Congestion Control (C3).

The second framework uses the network tomography, an edge-to-edge mechanism to infer per-link characteristics of a network domain, to detect congestion in a network domain. The tomography-based unresponsive flow detection scheme samples incoming flows at the ingress routers, and probes the network with sampled data. The edge-to-edge probing detects excessive packet loss inside of a network domain and the cause behind the loss. To alleviate the congestion, the unresponsive flows are regulated at the edge routers. We refer to this framework as tomography-based congestion control (TCC). We design detection and adaptive control mechanism for the TCC framework. During congestion, the control algorithm regulates the suspected flows such a way that the loss ratio of the congested links drops exponentially with time. In absence of the congestion, the flow rates are increased to the maximum value subscribed by the user. To achieve scalability, the detection and control

processing is done without involvement of core routers. The scheme has been evaluated, and the performance has been tested using the *ns-2* simulator.

5.2 Related Work

Floyd *et al.* discuss congestion collapse from undelivered packets in [63]. This situation arises when bandwidth is continuously consumed by packets at the upstream domains and are dropped at the downstream domains. The authors presented several ways to detect unresponsive flows. It is suggested that routers can monitor flows to detect whether flow is responsive to congestion or not. If a flow is not responsive to congestion, it can be penalized by discarding packets to a higher rate at the router. According to the authors there are some limitations of these tests to identify non-"TCP-friendly flow". It does not help to save bandwidth at the upstream if the flow sees the congestion at the downstream because this solution does not propagate the congestion information from downstream to upstream.

Seddigh *et al.* [107] suggest that if TCP and UDP are put into separate queues or Assured Forwarding classes, they may coexist fairly. This discrimination between TCP and UDP traffic may punish some well-behaved UDP flows. The core router does not know the profile of a flow and can not decide to allocate bandwidth to them fairly. The problem is associated with network load, capacity, and the reaction of different transport protocols to congestion. A dynamic control mechanism can solve this problem.

Albuquerque *et al.* [106] propose congestion avoidance mechanism named Network Border Patrol. To detect congestion, it measures entering rate of traffic to a domain and the leaving rate from the domain. It detects and restricts unresponsive traffic flows and eliminates congestion collapse. The border routers monitor all flows, measure rates, and exchange this information with all edge routers periodically and this can be expensive. Moreover, TCP is responsive so we do not need control mechanism for TCP at the edges.

Chow *et al.* [108] propose a framework where edge routers periodically obtain information from the core by probing, and adjust the conditioner using the traffic dynamics. In this scheme, core needs to maintain all the state information. A simpler scheme can be employed where core sends packets to edge routers only at the time of congestion.

Wu *et al.* propose Direct Congestion Control Scheme (DCCS) in [33]. In this scheme, they detect congestion by observing packet drops with *lowest priority to drop* at the core router. We follow the same rule in our research to detect congestion. Our core is simpler

in the sense that it detects drops of only unresponsive flows. Unlike [33], we design the shaper at the edge that controls the unresponsive flow.

Mahajan *et al.* [68] use Aggregate-based Congestion Control (ACC) to detect and control high bandwidth aggregate flows. They use the history of packet drops over a time interval, and the ACC agent matches prefix of IP destination addresses to detect flows going to the same destination address for Denial of Service(DoS) attacks. The ACC agent controls the flows using a rate-limiter and pushes status messages reporting the aggregate's arrival rate to the upstream routers. We use DiffServ architecture to detect and propagate messages. Our goal is to detect and control unresponsive flows and at the same time it can protect DoS attack by using their idea of prefix matching [68].

5.3 Core-assisted Congestion Control (C3) Framework

This section describes the extension of the DiffServ components that are necessary to support C3 framework. We need to extend core routers so that they can inform the edge routers about the congestion. The edge router has a traffic conditioner that may re-mark a traffic stream or may discard or shape packets to bring the stream into compliance with a traffic profile specified by the network administrator [5]. We have to use a proper shaping algorithm that can control unresponsive flows at the time of congestion. One additional modification is to be done at the edge; the ingress router of one domain informs the egress router of the previous domain about the congestion. Thus, congestion information is propagated to the upstream.

5.3.1 Support from Core Router

Core does not store any per flow reservation information. This makes the DiffServ architecture more scalable. We make little modification at the core routers to inform edge routers about the congestion.

Wu *et al.* [33] suggest that packets dropped at the core with lowest drop precedence, say DP0, indicates that there is a congestion in the network. We use a similar idea to identify the congestion. We detect congestion only for unresponsive flows using protocol information from transport layer. At the core, there is no way to classify packets as responsive or unresponsive. This idea of monitoring all flows vs. unresponsive flows is debatable. But it

is true that responsive flows will back off just after one time-out period. The advantage of monitoring responsive flows is small comparing to the overhead of monitoring it.

In C3 framework, the core routers store the tuple {source addr, destination addr, source port, destination port, protocol, timestamp, outgoing_link_bw} about a dropped packet. The cores send the drop information periodically to the ingress routers when total drop exceeds a local threshold. The first five fields of the tuple are necessary to identify a flow. The outgoing link bandwidth for each flow at the core helps to regulate the flow dynamically. The edge routers can be more aggressive if the core has a thin outgoing link. It can store the outgoing link information based on the core id. The core sends its id to mention the outgoing link the packet is traversing through if it has multiple of those. The modification at the core does not impose a lot of overheads on it because it stores/sends drop information only about unresponsive flows and only at the time of congestion.

5.3.2 Support from Edge Routers

There are two types of edge routers: ingress and egress. Same router can be configured to act as both. We present the extension on each of them separately.

Egress Router. We distinguish two types of drops at the edge routers. First one is a drop due to shaping at the edge, say *sdrop* and the other one is a drop due to congestion at the core/edge router, say *cdrop*. If there is a drop due to congestion, we use more information than just packets dropped to regulate conditioner. The egress router informs both drop information to the previous ingress router separately.

Ingress Router. The modification in the ingress router is to add/modify shaping algorithm. Ingress gets shaping drop, *sdrop*, from egress node and congestion drop, *cdrop*, from cores and egress routers, which are used for shaping. For a particular flow, suppose, the bottleneck bandwidth is bb. The bandwidth of outgoing link of the flow at the edge bo. The flow has an original profile (target rate) of op and adjusted profile of ap. The weighted average rate for this flow is $wavg$. In case of $cdrop$, the profile of the flow is updated using:

$$adjust = cdrop \times packet\_size \times max(1, \gamma\frac{bo}{bb}), \tag{5.1}$$

$$ap = max(0, min(ap - adjust, wavg - adjust)), \tag{5.2}$$

where $0 < \gamma < 1$, γ is aggressiveness to congestion control. Higher value of γ helps to converge the drop adjustment faster. In equation (5.2), adjusted profile is taking non-negative minimum value from current profile or from current average arrival rate. The arrival rate is calculated over a time frame using Time Sliding Window [27] algorithm.

For *sdrop*, the profile is adjusted using equation

$$ap = max(0, ap - sdrop \times packet\_size). \tag{5.3}$$

The ap is initialized at the beginning with op. If the router does not receive any drop information during a time interval, it increases the adjusted profile using equation below:

$$ap = min(op, ap + r). \tag{5.4}$$

The profile ap is adjusted periodically at a certain rate r, where r is initialized to a constant number of packets each time the router gets drop information. In absence of any drop, the rate r is increased using equation below:

$$r = min(\frac{wavg}{f}, 2 \times r), \tag{5.5}$$

where f is a factor that controls how fast the rate can be increased in the absence of any drop feedback. This rate should be bounded by the current average rate. This rate adjustment algorithm follows TCPs congestion control algorithm. The profile increment is doubled each time in absence of any drop until it hit a threshold $\frac{wavg}{f}$ and then it is increased linearly. At the edge, shaping is done based on the current average rate and the adjusted profile using the algorithm shown in Figure 5.1. The algorithm ensures dropping some packets when current rate is higher than the adjusted profile to reduce congestion. The chance of dropping all d packets from a particular flow is low. Too many packets should not be dropped at a time since it may deteriorate the application level quality of the flow.

5.4 Core-assisted Congestion Control (C3): Experimental Study

5.4.1 Simulation Setup

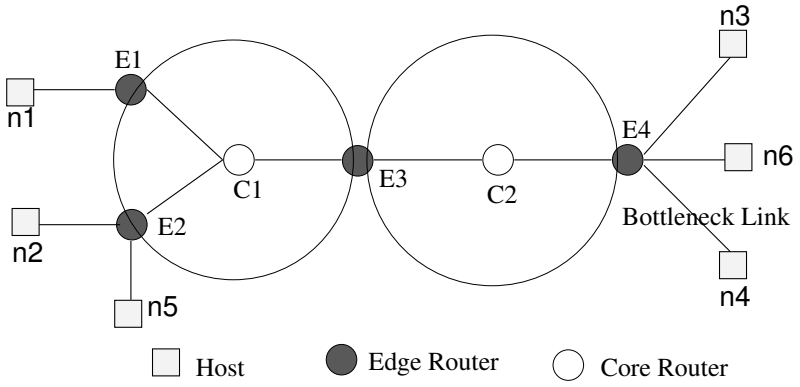
We use the ns-2 simulator [103] for our experiments. For the standard DiffServ implementation, we use software developed at Nortel Networks [104]. We use the TSW tagger meter and TSW3CM marker in the edge device. It has three drop precedences DP0, DP1 and DP2. DP0 means lower precedence to drop and DP2 means higher.

Algorithm:*Shaping*()
begin
  for each incoming fbw **do**
    if $avg > ap$ **then**
      drop next $d = min(\alpha, \frac{avg}{ap})$ packets
      where $\frac{num\_of\_active\_flows}{2} \leq \alpha < num\_of\_active\_flows$
      update average rate /* rate is decreasing over time*/
    else
      do regular marking
    end if
  end for
end

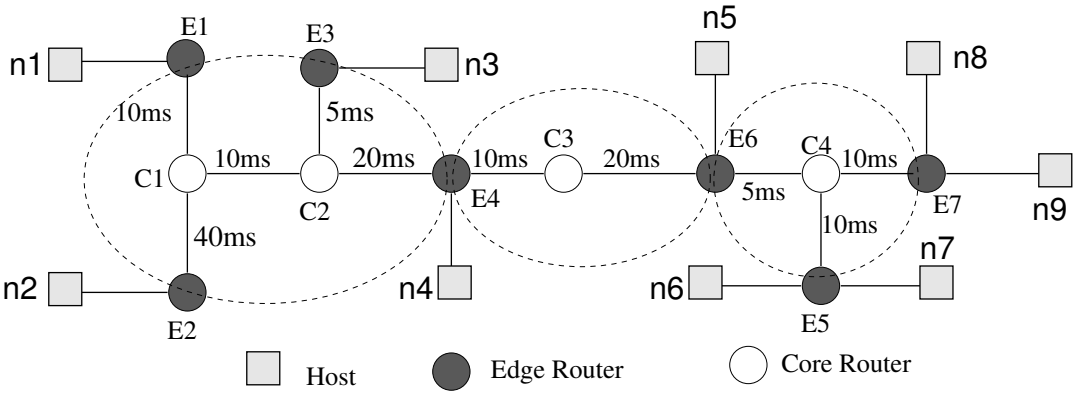
Fig. 5.1. Shaping fbws during congestion based on adjusted profi le of unresponsive fbws.

The simple topology, shown in Figure 5.2(a), has two network domains. We can test both ingress and egress routers to control congestion. The complex topology, Figure 5.2(b), is used later to simulate a more realistic situation. The edge devices implement traffi c conditioning, while the core device implements the Assured Forwarding [18] PHB using three drop precedences. We use throughput and packet drop ratio as metrics to evaluate performance. The parameters for the simulation is shown in Table 5.1. RED [23] parameters in the table are set to get the service differentiation among different types of packets. We use the software implementation of DiffServ network developed at Nortel Networks [104]. The RED parameters are taken from their work.

Initially, we use two aggregate fbws. Flow 1-3 is from node $n1$ to $n3$ and Flow 2-4 is from node $n2$ to $n4$. The number of fbws in each aggregate is varied with time. Normally, we use 10 TCP micro-fbws, where a micro-fbw represents a single TCP connection, as Flow 1-3, and 10 UDP micro-fbws as Flow 2-4. We add background traffi c (both TCP and UDP type) from node $n5$ to $n6$. We change the number of fbws and RTTs in different experiments. The metrics used to evaluate performance include:



(a) Topology 1



(b) Topology 2

Fig. 5.2. Simulation topologies. All links are 10 Mbps except bottleneck links. (a) Simple topology. The bottleneck link is $E4 \rightarrow n4$. (b) Complex topology with multiple domains. The bottleneck link is $E7 \rightarrow n9$.

Table 5.1
Simulation parameters and their typical values for the C3 framework.

Parameters	Value
Packet Size	1024 Bytes
TCP implementation	TCP new Reno
TCP window size	64
TSW window size	1 sec
weighted average w_q	0.002
RED parameters	$\{min_{th}, max_{th}, P_{max}\}$
DP0	{40,55,0.02}
DP1	{25,40,0.05}
DP2	{10,25,0.1}

1. **Throughput.** This denotes the average bytes received by the receiver application over simulation time. A higher throughput usually means better service for the application (e.g., smaller completion time for an FTP flow). For the ISP, higher throughput is preferable because this means that links are well-utilized.

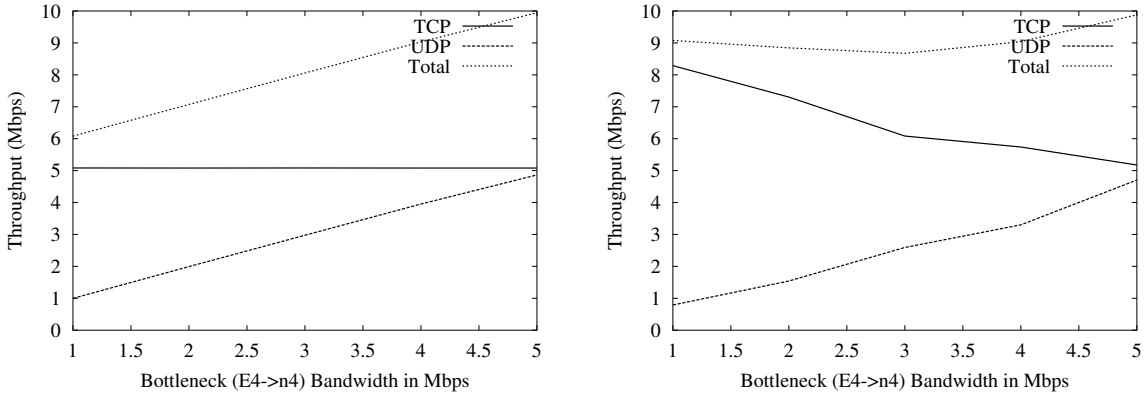
2. **Packet Drop.** We measure packets drop with the lowest precedence to drop, DP0, to show congestion and drops due to the shaping to show that shaping is done at different edges based on congestion information to improve the situation.

5.4.2 Simulation Results

We present a variety of scenarios to show that C3 framework works well. First, we show if there is no flow control, there is a chance for congestion collapse in the Internet. Then, we show that the congestion collapse can be overcome with C3. Both TCP and UDP type traffic are used as background traffic. We show the effect of RTT and number of flows on the flow control algorithm. Finally, simulation is conducted with complex topology and multiple cross traffic across the path of controlled flows.

5.4.3 Congestion Collapse

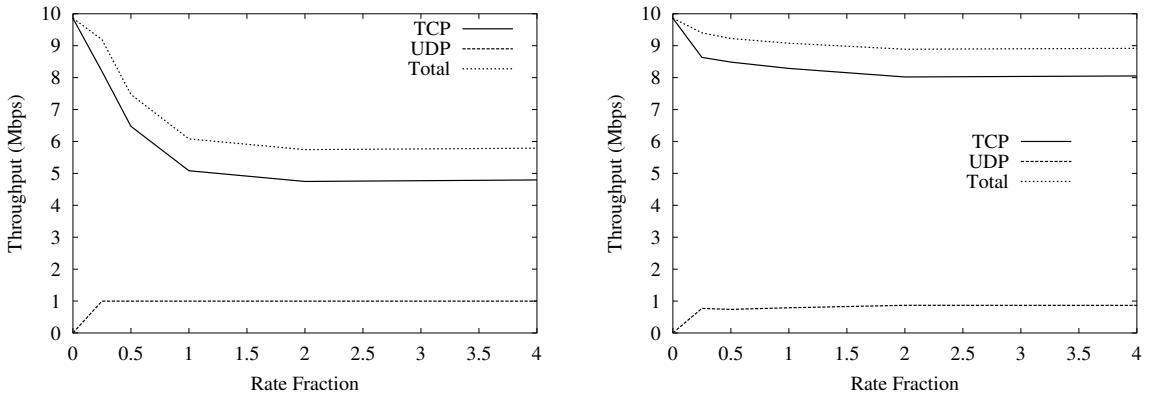
First, we show the congestion collapse due to unresponsive flows. In Figure 5.2 (a), there is an aggregate TCP flow with 10 micro-flows from host $n1 \rightarrow n3$ and a UDP aggregate flow with 10 micro-flows from host $n2 \rightarrow n4$. Both flows have the same profile or target rate (5 Mbps). Figure 5.3 shows how TCP and UDP flows behave with respect to the bottleneck bandwidth (bb), which is varied from 1 to 5 Mbps. The X-axis shows the bb, and the Y-axis shows the throughput achieved by both flows. Figure 5.3(a) shows that TCP flow gets its share of 5 Mbps all the time because it does not go through the congested link. When the bottleneck bandwidth is 1 Mbps, 4 Mbps bandwidth is wasted by UDP flows in the absence of the flow control. If we use C3 framework, it controls the UDP flow rate, and makes the extra bandwidth available for TCP flow. The Figure 5.3(b) shows that TCP flows get more than 8 Mbps when bb is 1 Mbps. The C3 prevents the network from congestion collapse due to undelivered packets. It can not achieve 100% link utilization, which can be achieved with proper tuning of parameters described in C3 framework.



(a) No Flow Control (b) With Flow Control

Fig. 5.3. (a) Without flow control, TCP gets only 5 Mbps when bottleneck bandwidth is 1 Mbps. (b) With Flow control, TCP gets 8 Mbps. Both flows have the same profile.

To show how effectively flow control scheme works, both TCP and UDP flows are assigned the same profile and the sending rate of UDP flows is varied. We define a rate fraction, $R_f = \frac{SendingRate}{Profile}$. For example, $R_f = 0.5$ means that the flow is sending at a rate 50% of its own profile, and $R_f = 4$ means the flow is sending at a rate four times of its



(a) No Flow Control

(b) With Flow Control

Fig. 5.4. UDP sending rate is varied using rate fraction, R_f. UDP sends as high as 20 Mbps (R_f=4), bottleneck ($E4 \rightarrow n4$) bandwidth is 1 Mbps.

own profile. In Figure 5.4, the X-axis shows the rate fraction, R_f, of UDP and the Y-axis shows the bandwidth achieved by both flows. When UDP's sending rate is zero, TCP gets the whole 10 Mbps. If sending rate of UDP is very low and no packet is dropped, there is no shaping (shaping drop is zero) at the edge. Figure 5.4(a) shows that when the sending rate is high enough to drop packet at the bottleneck link (bb= 1 Mbps), there is a congestion collapse in the network. TCP gets only 5 Mbps, and the total is 6 Mbps. With C3, Figure 5.4(b), the high sending rate of UDP does not affect the TCP flow to get extra bandwidth. The sending rate of UDP is increased as high as 4 times of its profile. The profile is 5 Mbps, i.e. UDP is sending at a rate of 20 Mbps, and still there is no congestion collapse with the C3 framework.

Figure 5.5 shows the cumulative packets received successfully at the destination side. It shows that the curve is linear, which means the receiver gets packets at a constant rate. There is no large number of drop in the middle of the network, which may affect the application performance. The application level quality of UDP flows will deteriorate if there is a sharp drop of huge number of packets in the middle of a network.

We use a similar algorithm as TCP's congestion control to adjust the rate of unresponsive flows. When there is a drop, the profile of a flow is adjusted temporarily and shaping is done based on the current average rate of a flow, number of active flows, and the adjusted profile. In the absence of any drop, the profile of the flow is increased periodically

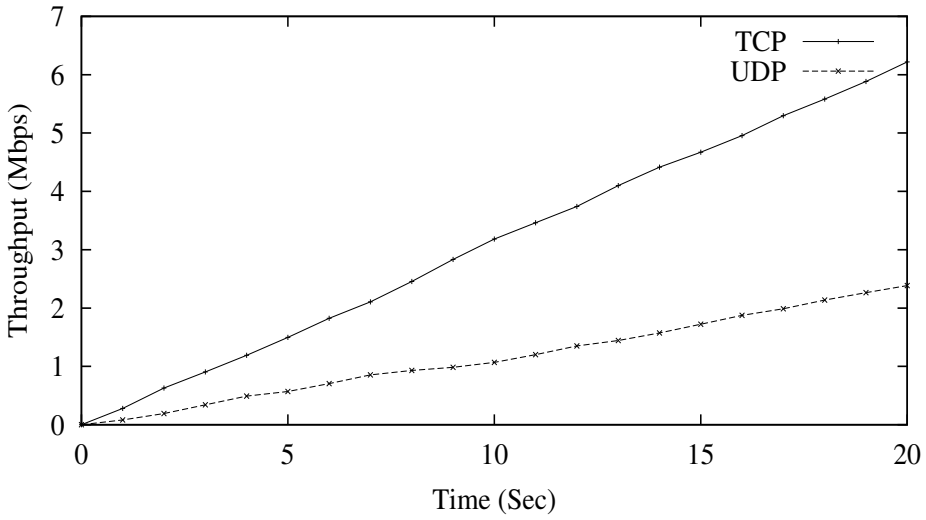


Fig. 5.5. Cumulative receiving rate at the destination. There is no sharp drop during transmission.

by adding a constant. The constant value is doubled (exponential increase) in each time interval until it hits a threshold, provided that there is no drop event (does not get any drop feedback from any core or edge routers). Then, the adjusted profi le is increased linearly. In Figure 5.6, the CBR is sending at a rate that is three times of its profi le, $R_f = 3$. The packet drop rate is increased and decreased based on traffi c changes (shaping information propagates with time and the rate is controlled accordingly). The TCP fbw has only initial drop, and then it does not see much drop. There is no drop with the background traffi c because it does not see any bottleneck on its way. It takes a short period of time at the beginning to make the drop rate stable.

5.4.4 Effect of RTT and Multiple Flows

We analyze the robustness of the fbw control algorithm. We use aggregate TCP fbws from $n1$ to $n3$, UDP fbws from $n2$ to $n4$, TCP as well as UDP fbw are used as background traffi c from node$n5$ to $n6$ of Figure 5.2(a). The RTT is varied by changing the link delay. The link delay for the path of TCP is kept fi xed while it is varied for both CBR and background traffi cs. First, we keep the number of fbws fi xed to 10 micro fbws per aggregate fbw, and later we fi x the RTT and vary the fbws from 5-200.

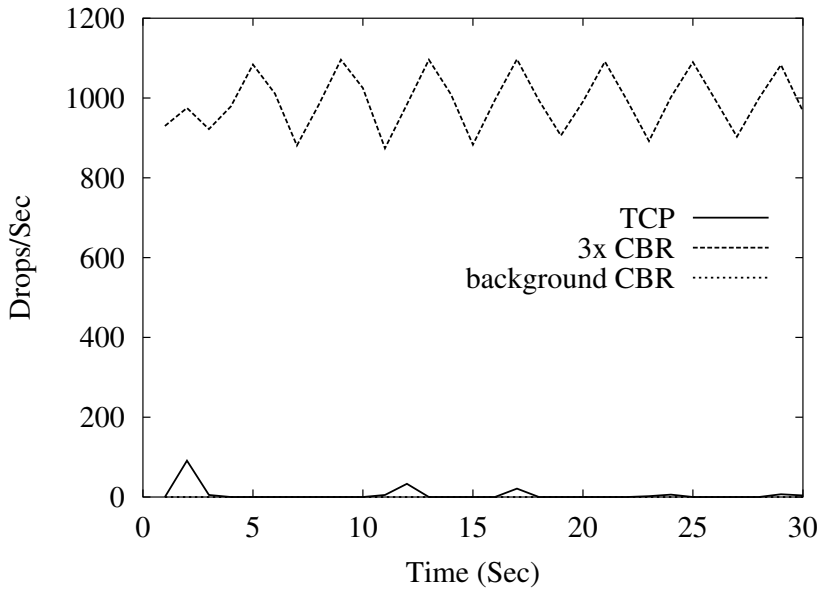
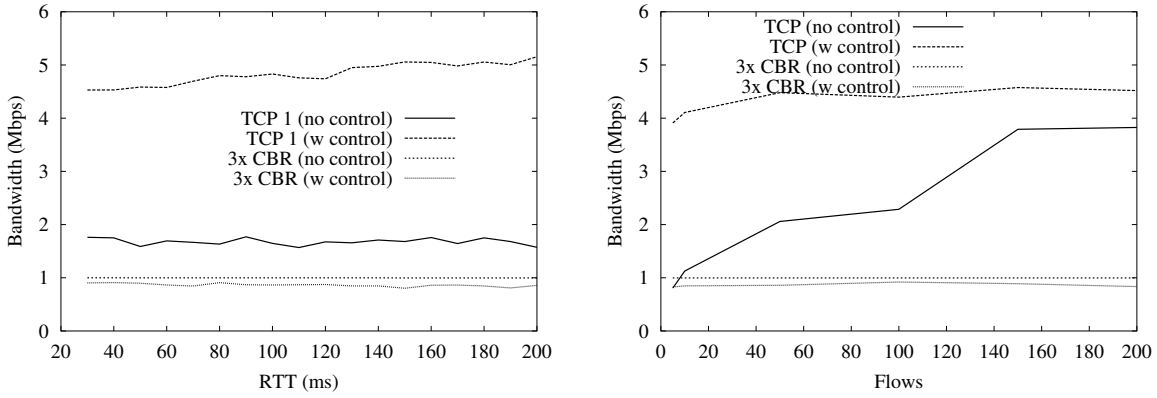


Fig. 5.6. Drop rate of packets for different flows. The TCP flows have very packets for a short period of time because they adjust the sending rate according to the network traffic. On the other hand CBR flows with three times sending rate have very high drop rates. The drop rate changes with time, and follow a saw-tooth like fashion. The background CBR does not have drops because it does not experience congestion.

Figure 5.7 (a) and (b) show the throughput achieved by TCP and CBR flows for varying RTT and varying the number of flows respectively. Background traffic is not shown in the figure. The flow control algorithm works nicely for different RTTs. The throughput does not change significantly with RTT. Figure 5.7(b) is more interesting because the bandwidth achievement of TCP flows changes with the number of active flows. When there is a large number of flows, TCP gets a good share even without having any flow control. It is because when many flows are present in a system, some flows starve and go for long time-out where as others still can get service. High volume of traffic makes it possible to increase the overall gain by the TCP flows. One interesting point we observed is that, the flow control algorithm needs a close approximation of active flows. If there are 200 active flows in the system, the algorithm works fine even if the approximation is 100 but fails to gain good performance if the approximation is 10. It is also true in the reverse manner, that is if there are 10 flows in the system and if the approximation is 200 then the performance of unresponsive flows deteriorate.



(a) Effect of RTTs

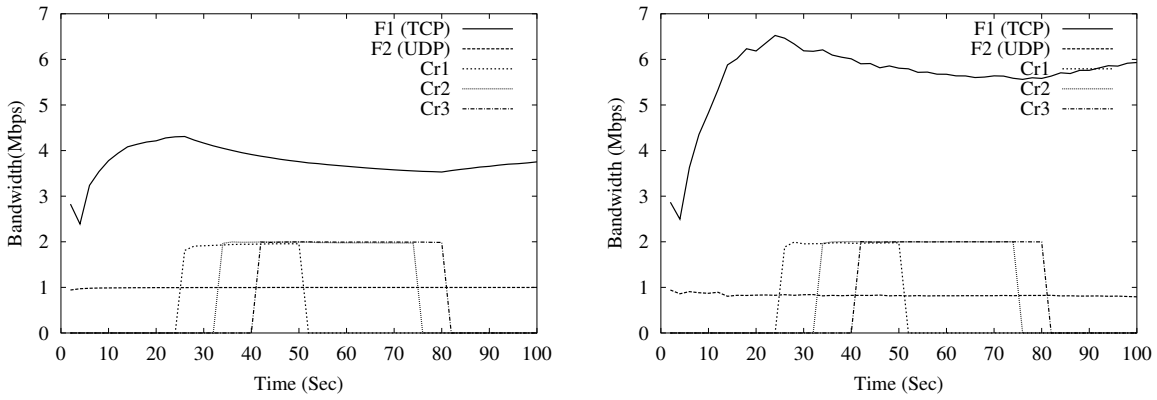
(b) Changing number of Flow

Fig. 5.7. RTTs and number of micro-flows per aggregate flow is varied for both TCP and UDP. Flow control works well with varying RTT and with changing number of micro flows per aggregate flow.

5.4.5 Simulation with Cross Traffic

We use more complex topology with multiple domains and with cross traffic to test our framework. The topology is shown in Figure 5.2(b). There are several aggregate flows present in this case such as TCP flow between $n1 \rightarrow n8$, UDP flow between $n2 \rightarrow n9$, $n3 \rightarrow n4$, $n5 \rightarrow n6$, and $n7 \rightarrow n10$. We label the flows $F1$ between $n1 \rightarrow n8$, $F2$ between $n2 \rightarrow n9$ and $Cr1$, $Cr2$ and $Cr3$ flows between $n3 \rightarrow n4$, $n5 \rightarrow n6$, and $n7 \rightarrow n10$ respectively. These Crs are used as cross traffic. We set the start and the finish time of these Crs flows differently to change the overall traffic situation over the path for the flows $F1$ and $F2$. There are 10 micro flows per aggregate in this setup. Flows $F1$ and $F2$ have same profile of target rate 5 Mbps, and all cross traffic are sending at a rate of 2 Mbps.

Figure 5.8 shows the bandwidth achievement of all aggregate flows mentioned above with and without flow control. The cross traffic achieves the same target in both scheme. This cross traffic does not suffer because the flows do not send more than their profile, and they do not see any bottleneck on their way. If there is no flow control, $F1$ (TCP) can not even get its target 5 Mbps, With flow control mechanism, $F1$ gets more than the target. It is because after controlling the UDP flow, TCP gets some unused bandwidth.



(a) No Flow Control

(b) With Flow Control

Fig. 5.8. Dynamic adjustment of F2 flow works fine in presence of cross traffic. TCP flow (F1) gets more bandwidth with flow control scheme.

5.5 Tomography-based Congestion Control (TCC)

In this framework, we use network tomography to detect congestion and control unresponsive flows. This framework is completely edge-to-edge and does not involve the core routers to achieve scalability. We briefly describe the loss inference mechanism that is used to detect congestion.

5.5.1 Network Tomography and Loss Inference

Network tomography uses correlations among end-to-end measurements to infer per-link characteristics. For example, Duffield *et al.* [29] use unicast packet "stripes" (back-to-back probe packets) to infer a link loss by computing the correlation of a packet loss within a stripe at different destinations. This scheme sends a series of probe packets, called a stripe, with no delay among the transmissions of successive (usually three) packets. For a two-leaf binary tree spanning nodes 0, k, R_1, R_2, as shown in Figure 5.9. The loss ratio of the link $k \rightarrow R_1$, for instance, can be estimated by sending stripes from the root 0 to the leaves R_1 and R_2. The first packet of a 3-packet stripe is sent to R_1, while the last two are sent to R_2. If a packet reaches to any receiver, we can infer that the packet must have reached the branching point k. Further, if R_2 gets the last two packets of a stripe, it is likely that R_1 receives the first packet of that stripe. The packet loss probability is calculated

based on whether all packets sent to R_1 and R_2 reach their destination. Similarly, the loss ratio of the link $k \rightarrow R_2$ is inferred using a complementary stripe, in which the first packet is sent to R_2 and the last two are sent to R_1. The loss ratio of the common path from $0 \rightarrow k$ from the transmission probability as shown below:

$$A_k = \frac{Z_{R_1} Z_{R_2}}{Z_{R_1 \cup R_2}}, \tag{5.6}$$

where Z represents the empirical mean of a binary variable which takes 1 when all packets sent to R_1 reach their destination and 0 otherwise. The mean is taken over n identical stripes. By combining estimates of stripes down each such tree, the characteristics of the common path from $0 \rightarrow k$ is estimated.

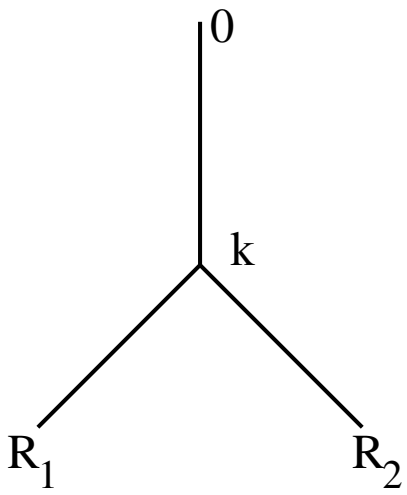


Fig. 5.9. Binary tree to infer loss of each link. The probes are sent from the sender 0 to both receivers R_1 and R_2.

This inference technique extends to general trees. Consider an arbitrary tree where for each node k, $R(k)$ denotes the subset of leaves descended from k. Let $Q(k)$ denote the set of ordered pairs of nodes in $R(k)$ descended from k. For each $(R1, R2) \in Q(k)$, a stripe should be sent from the root to the receivers R_1 and R_2.

5.5.2 Congestion Detection

Congestion detection depends on delay and loss measurements. Delay is the end-to-end latency; packet loss ratio is defined as the ratio of number of dropped packets from a flow

to the total number of packets of the same flow entered the domain. We first describe delay measurements and loss measurements before discussing the detection algorithm.

Delay Measurements

The unresponsive flows are sampled (using transport layer protocol information) at all ingress routers. The header of a sampled packet is used to probe the path of an unresponsive flow. The probe and user traffic follow the same path with a high probability, because the route does not get changed often inside a network domain. This measurement is a close approximation of the delay value that is experienced by the sampled flows in the network domain. This process is similar as it is described in Chapter 2. The main difference is that here only unresponsive flows are sampled.

For delay probing, the ingress routers encode the current timestamp $t_{ingress}$ into the payload, and mark the protocol field of the IP header with a new value. An egress router recognizes such packets, and removes them from the network. Additionally, the egress router computes the edge-to-edge link delay for a packet from the difference between its own time and $t_{ingress}$. We ignore minor drifts of the clocks since all routers are in one administrative domain, and can be synchronized fairly accurately. The egress classifies the probe packet as belonging to flow i, and update the average packet delay, avg_delay^i, for delay sample $delay^i(t)$ at time t using an exponential weighted moving average (EWMA):

$$avg\_delay^i(t) = \alpha \times avg\_delay^i(t-1) + (1-\alpha) \times delay^i(t), \tag{5.7}$$

where α is a small fraction $0 \leq \alpha \leq 1$ to emphasize recent history rather than the current sample alone.

If the average packet delay of path k exceeds the delay guarantee of the path for flow i, it is an indication of congestion. If the network is properly provisioned and flows do not misbehave, there should not be any delay greater than the estimated path delay for any flow i. A flow may experience high delay due to some other flows that cause congestion in the network.

Loss Measurements

If the edge-to-edge link delay is higher than a predefined threshold, we use the loss inference mechanism described in previous subsection (using equation 5.6) to measure loss in links that experience high delay. The objective of loss measurements is to obtain loss ratio of each individual links. The links with high losses are identified, and the loss value is used to control the congestion.

The delay probing identifies the paths that need to be considered for loss measurements. Each loss probing needs one sender and two receiver nodes. The incidence of high delay and the edge-to-edge paths that have high delay are reported to a congestion controller that sits at any edge router. The controller collects a set of paths $\mathcal{P}$ for the loss probing. The set $\mathcal{P}$ and the topology are used to determine the root of the probing tree for stripe-based probing. Probes are sent from the root to all ordered pair of edge routers. The root is selected such a way that it can cover maximum number of links in the set $\mathcal{P}$. If some links in the path set $\mathcal{P}$ are not covered, we need to repeat the loss measurements from another edge router considering as a root of the tree.

Detection

The links with high losses and egress routers through which flows are exiting the domain are identified. At these egress routers, all flows that are consuming high bandwidth are isolated. These rates are sent to the ingress routers through which the flows enter into the domain. The ingress routers compare the rate at which the suspected flows are entering and leaving the network domain. This identifies the flows that are not cooperating with the network to control their rates in response to congestion.

The rate of unresponsive flows can be reported per flow basis or in aggregate. If the number of flows to be reported exceeds a threshold, the feedback is done on an aggregate basis for each ingress router. This aggregation is done based on the traffic class. For each traffic class the unresponsive flows with high bandwidths are reported to the ingress routers. The identity of the ingress routers are obtained from the delay probes, where an identification code is used to relate a flow and its entry point. Otherwise, the egress does not know through which ingress routers the flows are entering into the domain. The detection algorithm runs as follows:

1. Each ingress router samples the user traffic for delay probing. The egress routers report the incidence of high delay and the edge-to-edge path that has high delay to a congestion controller.

2. The controller generates a probing tree using the set of path $\mathcal{P}$ that has high delay. The root of the tree is the sender of the loss probing. These probes are sent to every order pair of the edge routers of the domain. The loss probing obtains the loss ratio of each individual link of the path in $\mathcal{P}$.

3. Using the links with high loss ratio and the topology tree, a set of egress routers $\mathcal{E}$ are obtained through which the unresponsive flows are leaving the domain. These flows need to be controlled because they are causing congestion in the domain.

4. Flows are analyzed at each egress router of the set $\mathcal{E}$. The flows that are having high bandwidth are reported to the ingress routers through which flows are entering into the domain.

5.5.3 Congestion Control

To control the unresponsive flows, the inferred loss ratio is used. As it is discussed in detection mechanism, the loss ratio is sent to the ingress routers to control the flows. For each flow that has high packet loss inside a domain, the router reduces the rate proportionally to the packet drop rate inside the network. Suppose that a flow has an original profile (target rate) of $targetrate$. In case of the packet drop ratio $lossratio$, the profile of the flow is updated temporarily (to yield the rate $newprofile$) using $newprofile = targetrate \times lossratio$.

The congestion control algorithm adjusts the rate of the flows (that are causing drops to other flows) such a way that the loss ratio inside a network domain converges to a low predefined threshold LLOSSTHR. If the loss ratio decreases with time due to the current control setup, the control parameters are not changed until the loss ratio converges to a value. If the converged loss value is higher than the LLOSSTHR, the rate is controlled based on the loss ratio. If the loss ratio stays below LLOSSTHR for a while, the control algorithm allows more traffic to enter into the domain. The rate is increased linearly until it crosses the LLOSSTHR. In this way, the loss oscillates towards the LLOSSTHR parameter.

The control algorithm runs as follows:

1. If the loss ratio jumps to a high value, the control algorithm decreases the incoming rate of the unresponsive flows. This control decreases the loss ratio exponentially with time. If the loss decreases with time, the control algorithm does not change the rate control (RC) parameter. We refer to this decreasing loss ratio direction as DOWNWARD direction.

2. When the loss ratio converges to a value higher than the LLOSSTHR, the algorithm decreases the rate again based on current loss ratio. If the converged loss ratio is below the LLOSSTHR for a specified time, the algorithm allows more traffic to enter into the domain.

3. If the loss ratio curve goes UPWARD (opposite of DOWNWARD) direction, the rate control is increased with the current loss ratio.

Therefore, the rate control algorithm conducts Additive Increase and Multiplicative Decrease (AIMD). In presence of loss, the rate is control aggressively, and in absence of loss the rate is increased linearly. Thus, the rate adjustment algorithm is similar to TCP congestion control algorithm. At the edge router, shaping is done based on the RC parameter. The value of the RC parameter varies from $0 \rightarrow 1$. To shape a flow, a random number is generated. If the random number is less than RC, a packet from this flow is dropped. Otherwise, the packet is admitted into the domain.

5.6 TCC: Experimental Study

5.6.1 Setup

Using the *ns-2* [103] simulator, we evaluate the performance of our unresponsive flow control scheme. To test our framework, we have used a topology shown in Figure 3.4. The same topology is used in [29] to infer loss. We generate several TCP and UDP type aggregate flows in the network. Each aggregate flow contains 10 to 100 micro flows. Cross traffic is used to vary the background traffic by setting the start and the finish times of these flows differently, in order to change the overall traffic situation over the paths of all flows. The sending rate and the round trip time (RTT) of different flows are changed over time to show the robustness of the control mechanism for a variety of flows.

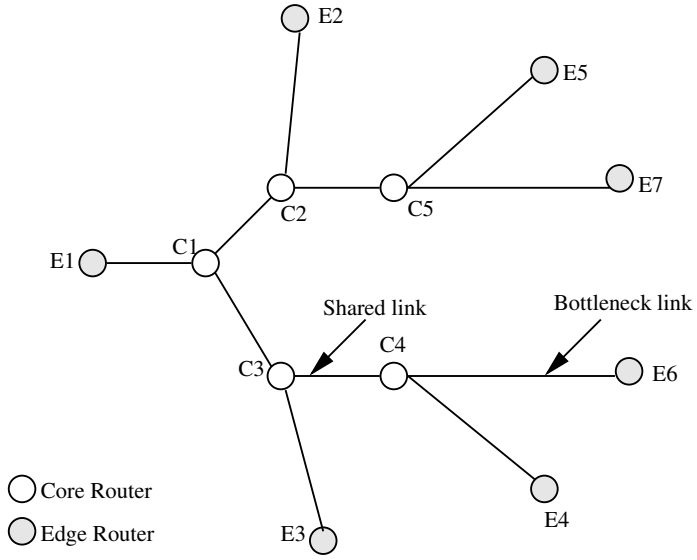


Fig. 5.10. Simulation topology. Each edge router is connected with multiple domains. $C4 \rightarrow E6$ is the bottleneck link in the setup. Unresponsive flows take their share from the shared link $C3 \rightarrow C4$, and their packets are dropped in the bottleneck link.

We conduct a series of experiments to show congestion collapse in the absence of flow control mechanism, the effectiveness of the tomography-based inference of network parameters, and the unresponsive flow detection and control in a variety of scenarios.

5.6.2 Congestion Detection

The congestion is detected using edge-to-edge link delay and link loss ratio. If some links are congested, the edge-to-edge delay through these links become very high. This high delay is used as an indication of congestion. Figure 5.11 shows delay of the path $E1 \rightarrow E6$. The latency of this path is 100 ms when the links are idle. However, the delay goes as high as one second due to the congestion. The figure also shows that with proper congestion control the delay can be reduced to a desired level.

The congested links are identified using stripe based unicast probing. The probes are sent to obtain loss ratio of the links that lie on the high delay path. In our experiment, we send probes from $E1$ to all other edge routers to obtain the loss of the links on the path $E1 \rightarrow E6$. The inferred loss for the link $C4 \rightarrow E6$ is shown in Figure 5.12. It shows loss inference for the topology described above for 3-packet stripes. First experiment has

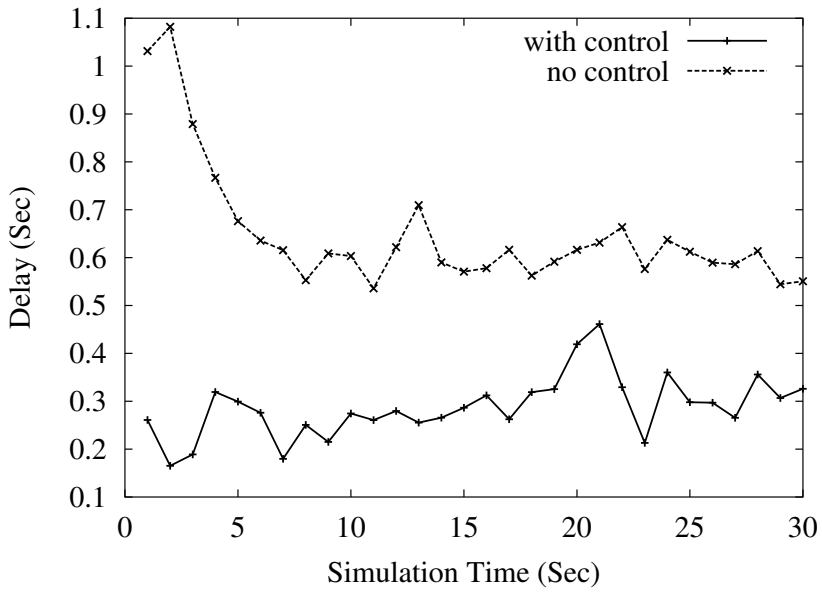


Fig. 5.11. Delay pattern changes with excessive traffic. This high delay is an indication that the edge-to-edge path is congested. The flow control mechanism alleviates the congestion, and reduces the delay.

fewer number of flows to cause packet drops inside the network domain. Second and third experiments have enough flows to cause huge packets drops in the network. The figure shows loss inference is close to the actual loss in most of the cases. In few cases, it over-estimates or under-estimates the loss. We can reduce this effect by increasing the time interval to measure probe loss. 4-packet stripe has little advantage on 3-packet stripe in our experiment.

Upon detection of the congested links, we identify the unresponsive flows at the egress routers, and pass this information to the appropriate ingress routers to control them. The inference mechanisms converge to real measurements value in 15-20 seconds. The detection mechanism is effective if the congestion continues for a while. If the congestion lasts only a few seconds, we do not need to control that.

5.6.3 Congestion Control

The unresponsive flows are controlled using a shaper. The shaping algorithm drops packets based on the service level agreement (SLA) parameters of the flow, the drop rate,

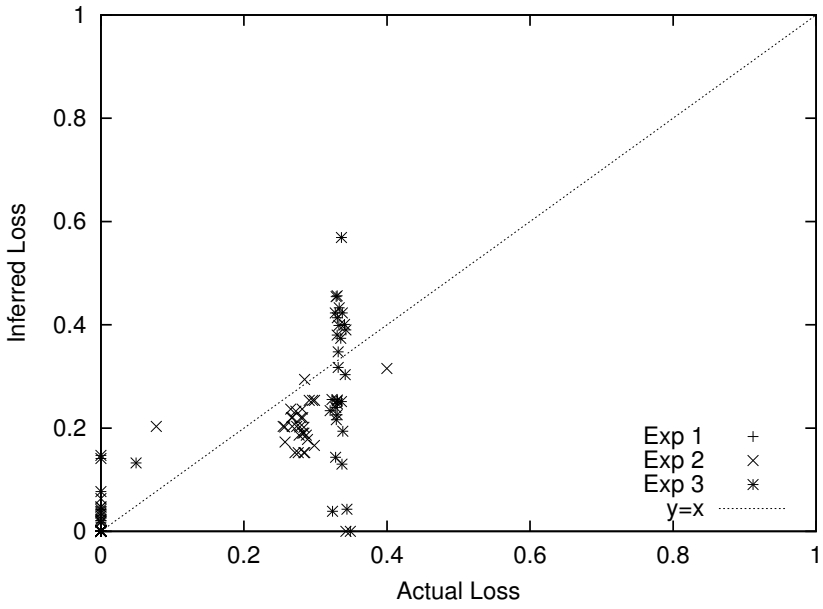


Fig. 5.12. Inferring loss using unicast stripe-based probing. The actual loss is close to the inferred loss.

and the sending rate of the flow. First, we show that in absence of congestion control there might be a congestion collapse in a network domain. Second, we show the performance of an adaptive congestion control algorithm. Third, the robustness of the algorithm is shown with varying number of micro flows, where a micro flow is defined with five tuples (source addr, source port, dest addr, dest port, and protocol).

Congestion Collapse. Congestion collapse due to undelivered packets wastes resources in a network. In our experiments, TCP and UDP flows share the link $C3 \rightarrow C4$. Then, the UDP flows experience congestion at the link $C4 \rightarrow E6$, which causes huge amount of packet drops. As, these UDP flows take the equal share with TCP flows of the link $C3 \rightarrow C4$, the resources are wasted in the next link. If we know the packets will be dropped any way at the link $C4 \rightarrow E6$, it is better to drop them earlier at the ingress router so that the TCP flows can get the wasted share of the link $C3 \rightarrow C4$, which increases the application level quality of the TCP flows. Figure 5.13 shows that without any flow control, the TCP flows obtain less than 1.5 Mbps, whereas, with flow control mechanism the bandwidth gain goes higher than 3 Mbps. The congestion window of a sampled TCP flow is shown with or without flow control in Figure 5.14. Flow control helps to increase the congestion window of the TCP flows.

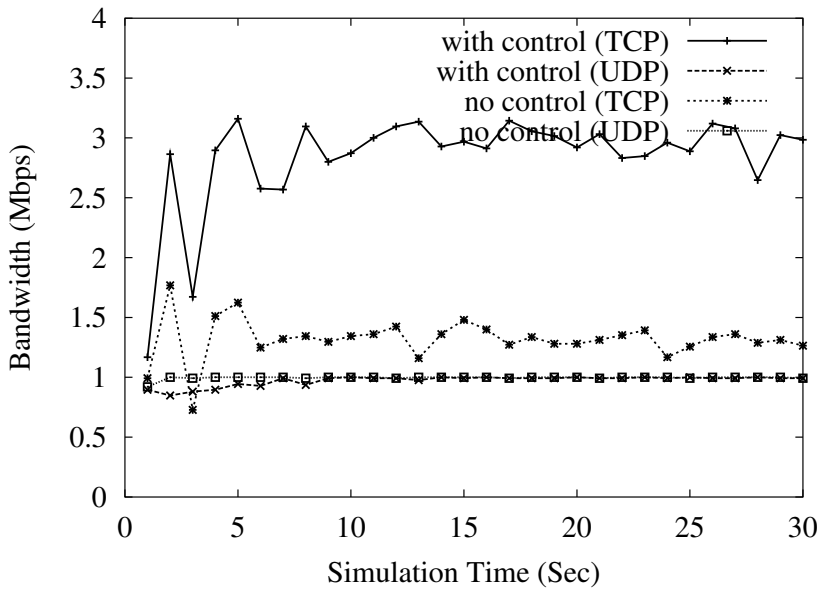


Fig. 5.13. Congestion collapse if there is no flow control. TCP gets the wasted bandwidth by the UDP flows when flow control mechanism is used.

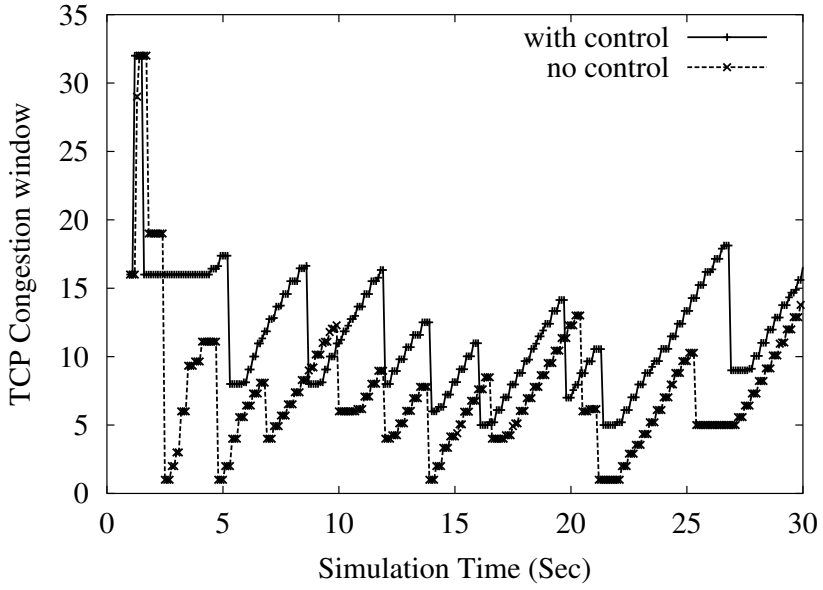


Fig. 5.14. Congestion window of a TCP flow with or without flow control. The congestion window is reset to one several times if there is no flow control.

Adaptive Control. The adaptive control algorithm throttles the rate of unresponsive flows during congestion so that the loss ratio converges to a low and predefined value. The algorithm needs to infer loss periodically. An interval of 30 seconds is used to infer loss, and high loss is informed to the appropriate flow controller. We run the experiment for 1200 seconds to evaluate the performance of the adaptive congestion control. Figure 5.15 shows that this mechanism helps the TCP flows to obtain 2.5 - 3 Mbps bandwidth. The UDP flows face more aggressive drops for a while, because the algorithm tries to determine the control rate at which the unresponsive flows should be shaped.

Figure 5.16 shows the loss pattern during this congestion control. Initially, the loss drops exponentially. When the loss ratio hits the lowest acceptable level at 600 seconds, the control mechanism allows more traffic into the domain. In this way, the rate control parameter is adjusted, and the loss ratio oscillates towards the LLOSSTHR value, which is 0.1 in our experiments.

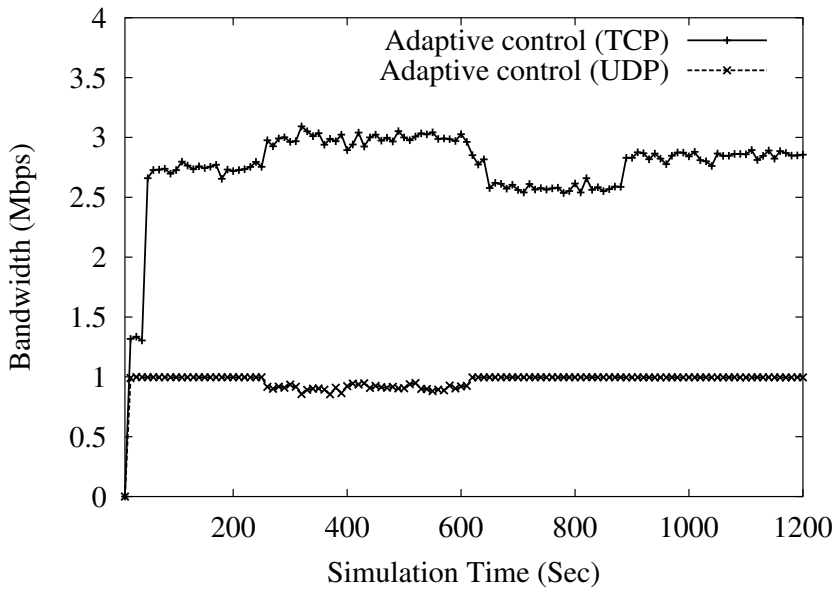


Fig. 5.15. Bandwidth gain by TCP and UDP flow with adaptive flow control.

Robustness. To show the robustness of the control algorithm, we vary the number of micro flows per aggregate flows from 10 to 100. We sample the bandwidth of TCP and UDP flows from 50 second to 60 second, and take the average to plot in Figure 5.17. This figure shows that increasing number of flows does not hurt the performance of the

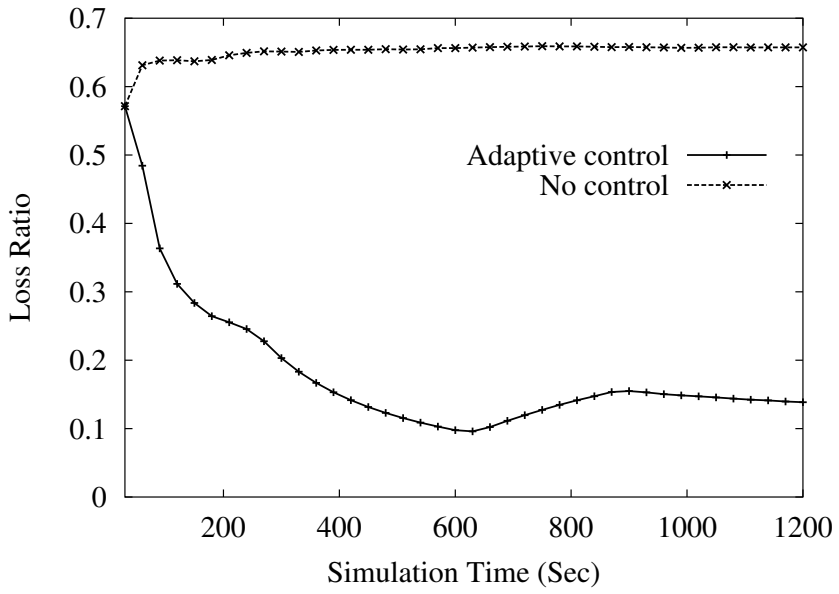


Fig. 5.16. Loss ratio with adaptive flow control. Initially the loss decays exponentially, and the loss converges with time to a low value.

congestion control algorithm. Experiments are conducted with different network dynamics that change the congestion dynamically to test the robustness of the control algorithm.

5.6.4 Overhead

The overhead of our scheme is low. If all links are OC3 type, on average each link experiences probe traffic less than 0.015% of the link capacity in the network domain shown in Figure 5.10. Our mechanism in non-intrusive, i.e., the injected traffic does not change the characteristics of the network domain.

5.7 Summary

We have proposed and evaluated frameworks to detect and regulate unresponsive flows to prevent congestion collapse due to undelivered packets. The C3 framework requires little extension at the core, and does not introduce a lot of overhead. The changes at the edges (ingress and egress) is not major. The implementation is simple and the deployment will be easy. In our scheme, core/egress routers sends control packets only at the time of

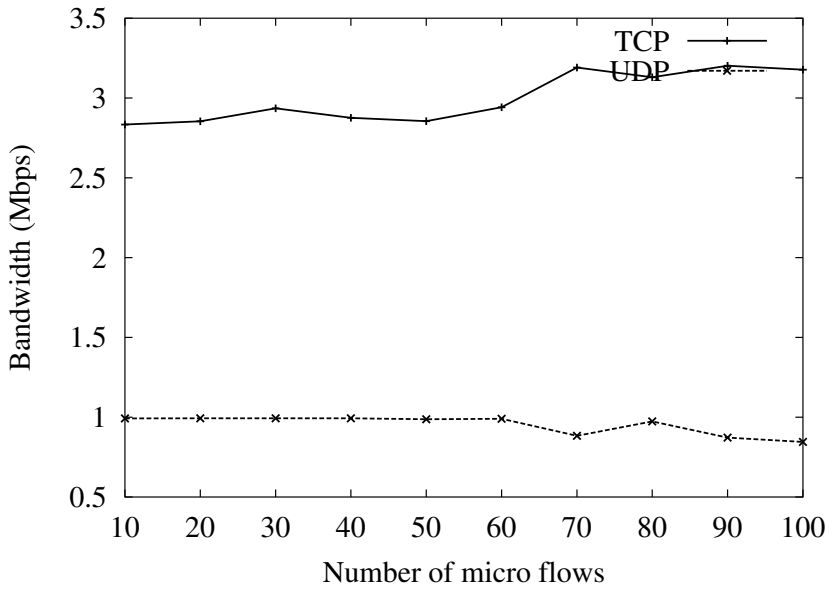


Fig. 5.17. Bandwidth achieved by TCP and UDP flows with varying number of micro flows.

congestion. For this reason, this scheme will not introduce a lot of control packets into the network.

The TCC is a new and scalable way to detect and regulate unresponsive flows to prevent congestion collapses due to undelivered packets. This scheme does not require any help from the core routers and introduces a very low overhead. The changes require at the edges (ingress and egress) are minor. The implementation is simple and the deployment should be easy. As the probe sizes are not huge, consumption of user profile by this probing is insignificant.

6. CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

In this thesis, we provide the answer of the question "How can we protect a network domain, and serve the users with their quality of service (QoS) requirements?" To solve the problem, we need to perform the followings:

1. A monitor is required to ensure that no one is stealing bandwidth and/or injecting packets through multiple ingress routers to create distributed denial of service (DDoS) attacks. The monitor always observe a network domain for changes of its internal characteristics. Flow control mechanism is used to recover from attacks.

2. All edge routers need traffic conditioner to ensure every user is following their specified service level agreements (SLAs). Traffic exceeds their SLAs needs to be policed and shaped.

3. Unresponsive flows do not response to a congestion status, rather than, sends packets at the same rate. This can lead to a congestion collapse in a network. We proposed algorithms to detect and control unresponsive flows in a QoS network.

Our experimental results show that we can detect change of internal characteristics of a network due to attacks that inject excessive packets in a network. The monitoring mechanism is able to detect service violations and DoS attacks. The stripe-based scheme probes the network on demand to reduce the total probe required for monitoring. The overlay-based scheme reduces the overhead even more. Our analytic analysis (verified by simulation) shows that even if 20% links of a network are congested, the status of each link can be identified with probability $\geq .98$. If the network is 40% congested, this probability is still high $(.65)$. However, if the network is more than 60% congested, this method can not achieve anything significant since almost every edge-to-edge path has one or more congested links. This new tomography scheme requires only $O(n)$ probes when less than

20% links are congested, where n is the number of edge routers. For an OC3 link, the probe traffic is 0.002% of the link capacity. The distributed monitoring requires $O(n^2)$ in worst case in contrast to $O(n^3)$ probes required by the stripe-based monitoring to detect attacks in both directions of all links. The distributed monitoring conducts probing in parallel enabling the system to perform real time monitoring. The monitoring scheme has less computation overhead than filtering or traceback mechanisms. However, monitoring schemes inject probe in the networks and other schemes do not.

We improve the traffic conditioner that intelligently uses transport layer flow characteristics to improve application level QoS. The conditioner achieves scalability avoiding using excessive per-flow information. We minimizes RTT bias on a TCP flow so that long-RTT flows do not suffer much while short-RTT flows steal all the available resources. The conditioner is tested for data extensive traffic such as FTP and delay sensitive applications such as Telnet and Web traffic.

We have shown mechanism to detect loss during congestion and use this information to control unresponsive flows. The tomography based congestion control is the first scheme that detects unresponsive flows using only edge-to-edge measurements. The flow control improves the overall bandwidth achievement because the resources wasted by unresponsive flows are saved and given to other flows.

6.2 Future Work

6.2.1 Network Provisioning

As an extension of this research, a network provisioning component will be integrated with other components of the edge routers of a Differentiated Services (DiffServ) domain. A service provider can maximize the profit by proper network provisioning. Provisioning is an important issue for a QoS network domain. Without proper provisioning, it is impossible to maintain the SLA bounds promised to each user. An admission control is necessary to provide end-to-end quality of service (QoS) guarantees in term of maximum delay and loss probability.

Guaranteed service [109] provides hard QoS bounds for IP networks. However, it requires to store excessive flow information at the routers and does not scale well in the Internet. The DiffServ [60] came along to address scalability by aggregating flows into fixed number of classes. A DiffServ network provides guaranteed service EF and a group

of AF class traffic. The EF class traffic is similar as the guaranteed service provided by Integrated Service architecture [3]. The EF traffic needs a requirement that the sum of incoming EF traffic must not exceed the reserved rate of the output link capacity. It is shown that keeping sum of peak input rates less than the output capacity does not ensure low delay or jitter when traffic are combined inside a network domain. Again, no loss requirement increases the delay to unacceptable even for few hops. An admission control based on worst case analysis to provide hard delay and loss bound keeps the utilization of a network very low. This is because the worst case analysis uses maximum delay, which is usually far away from delay distribution [110]. To obtain a more realistic resource utilization efficiency, the zero loss requirements need to be eliminated. In our model, we assume non-zero loss for EF class with very low end-to-end loss probability. The statistical bound of EF traffic improves the resource utilization significantly [111, 112] without hurting the objective of the traffic class.

The AF model is not well studied yet. A simple performance model of AF is studied by May *et al.* in [62]. We will follow same way to derive our model. This model will lead us to delay and loss bound of AF traffic. These bounds are necessary in profit maximization formulation as constraints.

A conceptual framework for Service Level Management is discussed in [113] and used software development paradigm. Internet pricing and provisioning are discussed in [114, 115]. Semret *et al.* [115] discuss about Internet provisioning using the game theoretic model. This discussion is mainly about optimizing the capacity of provider resources and the demands of the user for the raw-capacity of the network. End to end network provisioning requires signaling mechanism to make reservation in all domains from source to destination. The commonly used protocol for this purpose is COPS [116]. AT&T and IBM work on the policy based networking for automation of network provisioning [117]. Other tools for automation of dynamic service provisioning and network configuration are discussed in [118, 119]. Our research task is to formulate profit maximization using admission rate of traffic of different classes and assigning weights to queues at each internal router. If necessary, we divert traffic inside the network domain for better link and router utilization having the constraint of the users QoS parameters, delay, jitter, loss requirements. Recent work of Liu *et al.* [120] distribute user request to different web server to maximize profit. We want to use similar approach in router level. To model the incoming traffic to each

edge and core router and to provide statistical bounds of QoS parameters are challenging research tasks.

We aim to design different components of an edge router with the flow control mechanism where each part is scalable. In future, we plan to

- Derive statistical model for delay and loss bounds of AF class traffic. Validate all models for realistic network and traffic models.
- Formulate SLA profit maximization and simulate with multiple domains networks to evaluate the performance.

6.2.2 Monitoring Overlay and Sensor Networks

It is important to maintain the overlay network working properly. The monitoring scheme detects failure of the peers, and provides this information to the agents that are involved in collective and opportunistic information aggregation. We propose a low overhead and scalable monitoring scheme for overlay networks. The research problem we plan to address in this area is to design a low overhead and scalable monitoring scheme for an overlay network. Monitoring a large scale overlay network poses several challenges. First, the topology of the network changes with time because peers join and leave the system often in a dynamic peer-to-peer (P2P) networks. Second, the number of peers are extremely high in a P2P network in contrast to the number of edge routers of a network domain. Thus, the overlay-based network monitoring might not be efficient to monitor an overlay network.

Monitoring wireless sensor networks is different from monitoring wired networks. Our monitoring schemes are not directly applicable to the wireless sensor networks. Recent works propose several techniques to monitor the health of sensors. Residual energy scanning [121], aggregates computation [122], and distributed monitor [123] are some of them. We plan is to devise a new monitoring mechanism that is energy efficient. The concept of parameter inference will be explored to obtain the status of a sensor. The status will be sent to the data center only when it is necessary.

LIST OF REFERENCES

LIST OF REFERENCES

[1] "Internet engineering task force, http://www.ietf.org."

[2] X. Xiao and L. Ni, "Internet QoS: A big picture," *IEEE Network*, vol. 13, 2, pp. 8–19, Mar./Apr. 1999.

[3] R. Braden, D. Clark, and S. Shenker, "Integrated services in the Internet architecture: An overview." RFC 1633.

[4] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource ReSerVation Protocol (RSVP)." RFC 2205, Sept. 1997.

[5] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An architecture for Differentiated Services." RFC 2475, Dec. 1998.

[6] K. Nichols, V. Jacobson, and L. Zhang, "A two-bit differentiated services architecture for the Internet." RFC 2638, July 1999.

[7] R. Callon, P. Doolan, N. Feldman, A. Fredette, G. Swallow, and A. Viswanathan, "A framework for multiprotocol label switching." Internet draft, Nov. 1997.

[8] I. Stoica and H. Zhang, "Providing guaranteed services without per flow management," tech. rep., CMU-CS-99-133, May 1999.

[9] Y. Bernet, P. Ford, R. Yavatkar, F. Baker, L. Zhang, M. Speer, R. Braden, B. Davie, J. Wroclawski., and E. Felstaine, "Interoperation of RSVP/Intserv and Diffserv networks." Internet draft, draft-ietf-issll-diffserv-rsvp-01.txt, 1999.

[10] D. Awduche, J. Malcolm, M. J. Agogbua, M. O Dell, and J. McMaus, "Requirements for traffic engineering over MPLS." Internet draft, draft-ietf-mpls-traffic-eng.00.txt, Oct. 1998.

[11] P. Vaananen and R. Ravikanth, "Framework for traffic management in MPLS networks." Internet draft, draft-vaananen-mpls-tm-framework-00.txt, Mar. 1998.

[12] R. J. Gibbens, S. K. Sargood, F. P. Kelly, M. Azmoodeh, R. Macfadyen, and N. Macfadyen, "An approach to service level agreements for IP networks with differentiated services."

[13] Y. Bernet, S. Blake, D. Grossman, and A. Smith, "An informal management model for diffserv routers." Internet Engineering Task Force Draft, July 2000.

[14] R. Yavatkar, D. Pendarakis, and R. Guerin, "A framework for policy-based admission control." RFC 2753, Jan. 2000.

[15] I. Yeom and N. Reddy, "Marking for QoS improvement," tech. rep., Texas A & M University College Station, Dept. of Elec. Engg., TX 77843-3128, 1999.

[16] I. Yeom and N. Reddy, "Impact of marking strategy on aggregated flows in a Differentiated Services network," in *Proc. IEEE/IFIP Eighth International Workshop on Quality of Service (IWQoS)*, (London, United Kingdom), May 1999.

[17] V. Jacobson, K. Nichols, and K. Poduri, "An Expedited Forwarding PHB." RFC 2598, June 1999.

[18] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski, "Assured Forwarding PHB Group." RFC 2597, June 1999.

[19] J. Ibanez and K. Nichols, "Preliminary simulation evaluation of an Assured Service." Internet draft, draft-ibanez-diffserv-assured-eval-00.txt, Aug. 1998.

[20] O. Elloumi, S. de Cnodder, and K. Pauwels, "Usefulness of three drop precedences in Assured Forwarding service." Internet draft, draft-elloumi.diffserv-threevstwo-00.txt, 1999.

[21] M. Goyal, P. Misra, and R. Jain, "Effect of number of drop precedences in assured forwarding," in *Proc. Globecom '99*, (Rio de Janeiro, Brazil), Dec. 1999.

[22] K. Nichols, S. Blake, F. Baker, and D. Black, "Definition of the differentiated service field (DS field) in the IPv4 and IPv6 headers." RFC 2474, Dec. 1998.

[23] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, pp. 397–413, Aug. 1993.

[24] W. Feng, D. Kandlur, D. Saha, and K. Shin, "BLUE: A new class of active queue management algorithms," tech. rep., U. of Michigan CSE-TR-387-99, Apr. 1999.

[25] T. Ott, T. Lakshman, and L. Wong, "SRED: Stabilized RED," in *Proc. IEEE INFOCOM*, (New York, New York), Mar. 1999.

[26] Z. Cao, Z. Wang, and E. Uegura, "Rainbow fair queueing: Fair bandwidth sharing without per-flow state," in *Proc. IEEE INFOCOM*, (Tel-Aviv, Israel), Mar. 2000.

[27] D. Clark and W. Fang, "Explicit allocation of best effort packet delivery service," *IEEE/ACM Transactions on Networking*, vol. 6, 4, pp. 362–374, 1998.

[28] M. Thottan and K. Swanson, "Sequin: Scalable edge-based qos for intra-domain networks." http://www.bell-labs.com/org/1134/, 2001.

[29] N. G. Duffield, F. L. Presti, V. Paxson, and D. Towsley, "Inferring link loss using striped unicast probes," in *Proc. IEEE INFOCOM*, (Anchorage, Alaska), Apr. 2001.

[30] Y. Vardi, "Network tomography: Estimating source-destination traffic intensities from link data," *Journal of the American Statistical Association*, vol. 91, pp. 365–377, Mar. 1996.

[31] M. Coates and R. Nowak, "Network tomography for internal delay estimation," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing*, (Salt Lake City, Utah), May 2001.

[32] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. K. Ramakrishnan, S. Shenker, and J. Wroclawski, "Recommendations on queue management and congestion avoidance in the Internet." RFC 2309, Apr. 1998.

[33] H. Wu, K. Long, S. Cheng, and J. Ma., "A Direct Congestion Control Scheme for Non-responsive Flow Control in Diff-Serv IP Networks." Internet draft, draft-wuht-diffserv-dccs-00.txt, Aug. 2000.

[34] IEPM, "Internet End-to-end Performance Monitoring." http://www-iepm.slac.stanford.edu/, 2002.

[35] J. Case, M. Fedor, M. Schoffstall, and J. Davin, "A Simple Network Management Protocol (SNMP)." IETF RFC 1157, May 1990.

[36] Waldbusser, S., "Remote network monitoring management information base." IETF RFC 2819, May 2000.

[37] Cisco, "Netflow services and applications." http://www.cisco.com/, 2002 May 2000.

[38] E. Al-Shaer, H. Abdel-Wahab, and K. Maly, "HiFi: a new monitoring architecture for distributed systems management," in *Proc. IEEE 19th International Conference on Distributed Computing Systems (ICDCS '99)*, (Austin, Taxas), pp. 171–178, May 1999.

[39] R. Subramanyan, J. Miguel-Alonso, and J. A. B. Fortes, "A scalable SNMP-based distributed monitoring system for heterogeneous network computing," in *Proc. High Performance Networking and Computing Conference (SC 2000)*, (Dallas, Texas), 2000.

[40] A. Liotta, G. Pavlou, and G. Knight, "Exploiting agent mobility for large-scale network monitoring," *IEEE Network*, vol. May/June, 2002.

[41] K. G. Anagnostakis, S. Ioannidis, S. Miltchev, J. Ioannidis, M. Greenwald, and J. M. Smith, "Efficient packet monitoring for network management," in *Proc. IEEE Network Operations and Management Symposium (NOMS)*, (Florence, Italy), Apr. 2002.

[42] V. Paxson, *Measurement and Analysis of End-to-End Internet Dynamics*. PhD thesis, University of California, Berkeley, Computer Science Division, 1997.

[43] V. Paxson, G. Almes, J. Mahdavi, and M. Mathis, "Framework for IP performance metrics." IETF RFC 2330, May 1998.

[44] Y. Breitbart, C. Y. Chan, M. Garofalakis, R. Rastogi, and A. Silberschatz, "Efficiently monitoring bandwidth and latency in IP networks," in *Proc. IEEE INFOCOM*, (Anchorage, Alaska), Apr. 2001.

[45] M. Dilman and D. Raz, "Efficient reactive monitoring," in *Proc. IEEE INFOCOM*, (Anchorage, Alaska), Apr. 2001.

[46] J. Jiao, S. Naqvi, D. Raz, and B. Sugla, "Toward efficient monitoring," *IEEE Journal on Selected Areas in Communications*, vol. 18(5), May 2000.

[47] M. C. Chan, Y.-J. Lin, and X. Wang, "A scalable monitoring approach for service level agreements validation," in *Proc. International Conference on Network Protocols (ICNP)*, (Osaka, Japan), pp. 37–48, Nov. 2000.

[48] A. Adams, T. Bu, R. Caceres, N. Duffi eld, T. Friedman, J. Horowitz, F. L. Presti, S. B. Moon, V. Paxson, and D. Towsley, "The use of end-to-end multicast measurements for characterizing internal network behavior," *IEEE Communications*, vol. 38, no. 5, May 2000.

[49] T. Bu, N. Duffi eld, F. L. Presti, and D. Towsley, "Network tomography on general topologies," in *Proc. ACM SIGMETRICS*, (Marina del Rey, California), June 2002.

[50] R. Cáceres, N. G. Duffi eld, J. Horowitz, and D. Towsley, "Multicast-based inference of network-internal loss characteristics," *IEEE Transactions on Information Theory*, vol. 45, pp. 2462–2480, Nov. 1999.

[51] G. Spafford and S. Garfi nkel, *Practical Unix and Internet Security*. O'Reilly & Associates, Inc, second ed., 1996.

[52] F. Zhi, S. W. Felix, T. S. Wu, H. Huang, and F. Gong, "Security issues for differentiated service framework." Internet Engineering Task Force Draft, Oct. 1999.

[53] W. Stallings, "SNMPv3: A Security enhancement for SNMP," *IEEE Communications Surveys , Vol.1 No. 1*, 1998.

[54] G. Sager, "Security fun with OCxmon and cflowd." Intenet2 working group meeting, Nov. 98.

[55] R. Stone, "Centertrack: An IP overlay network for tracking DoS floods," in *Proc. USENIX Security Symposium*, (Denver, Colorado), Aug. 2000.

[56] S. Seshan, M. Stemm, and K. R. H., "Spand: Shared passive network performance discovery," in *Proc. USENIX Symposium on Internet Technologies and Systems (USITS '97)*, Dec. 1997.

[57] S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, A. Vahdat, G. Voelker, and J. Zahorjan, "Detour: a case for informed Internet routing and transport," *IEEE Micro, v 19, no 1*, Jan. 1999.

[58] D. Anderson, H. Balakrishnan, F. Kaashoek, and R. Morris, "Resilient overlay network," in *Proc. ACM Symp on Operating Systems Principles (SOSP)*, (Banff, Canada), Oct. 2001.

[59] N. G. Duffi eld and M. Grossglauser, "Trajectory sampling for direct traffi c observation," *IEEE/ACM Transactions on Networking*, vol. 9, pp. 280–292, June 2001.

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

[61] P. Ferguson and D. Senie, "Network ingress fi ltering: Defeating denial of service attacks which employ IP source address spoofi ng agreements performance monitoring." IETF RFC 2827, May 2000.

[62] M. May, J. Bolot, A. Jean-Marie, and C. Diot, "Simple performance models of differentaited services schemes for the Internet," in *Proc. IEEE INFOCOM*, (New York, New York), Mar. 1999.

[63] S. Floyd and K. Fall, "Promoting the use of end-to-end congestion control in the Internet," *IEEE/ACM Transactions on Networking*, vol. 7, no. 4, pp. 458–472, Aug. 1999.

[64] K. G. Anagnostakis, M. B. Greenwald, and R. S. Ryger, "On the sensitivity of network simulation to topology," in *Proc. of the 10th IEEE/ACM Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS 2002)*, Oct. 2002.

[65] Y. Zhang, N. G. Duffield, V. Paxson, and S. Shenker, "On the constancy of Internet path properties," in *Proc. ACM SIGCOMM Internet Measurement Workshop*, Nov. 2001.

[66] S. Savage, "Sting: a TCP-based network measurement tool," in *Proc. USENIX Symposium on Internet Technologies and Systems (USITS '99)*, (Boulder, Colorado), Oct. 1999.

[67] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with rocketfuel," in *Proc. ACM SIGCOMM*, (Pittsburgh, Philadelphia), Aug. 2002.

[68] M. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker, "Controlling high bandwidth aggregates in the network," *ACM Computer Communication Review*, vol. 32, no. 3, pp. 62–73, July 2002.

[69] J. Jung, B. Krishnamurthy, and D. Rabinovich M., "Flash crowds and denial of service attacks: Characterization and implications for cdns and web sites," in *Proc. World Wide Web (WWW)*, (Honolulu, Hawaii), May 2002.

[70] D. Moore, G. M. Voelker, and S. Savage, "Inferring Internet denial-of-service activity," in *Proc. USENIX Security Symposium*, (Washington D.C), Aug. 2001.

[71] L. Garber, "Denial of Service attacks rip the Internet," *IEEE Computer* , vol. 33,4, pp. 12–17, Apr. 2000.

[72] S. Savage, D. Wetherall, A. Karlin, and T. Anderson, "Network support for IP traceback," *IEEE/ACM Transaction on Networking*, vol. 9:(3), pp. 226–237, June 2001.

[73] C. L. Schuba, I. V. Krsul, M. G. Kuhn, E. H. Spafford, A. Sundaram, and D. Zamboni, "Analysis of a denial of service attack on tcp," in *Proc. IEEE Symposium on Security and Privacy*, (Oakland, California), May 1997.

[74] C. Barros, "A proposal for ICMP traceback messages." Internet draft, http://www.research.att.com/lists/ietf-itrace/2000/09/msg00044.html, Sept. 18, 2000.

[75] V. Paxson, "An analysis of using reflectors for distributed denial-of-service attacks," *ACM Computer Communication Review*, vol. 31 (3), July 2001.

[76] S. M. Bellovin, "ICMP traceback messages." Internet draft, draft-bellovin-itrace-00.txt, Mar. 2000.

[77] A. Snoeren, C. Partridge, L. Sanchez, W. Strayer, C. Jones, and F. Tchakountio, "Hashed-based IP traceback," in *Proc. ACM SIGCOMM*, (San Diego, California), Aug. 2001.

[78] H. Burch and H. Cheswick, "Tracing anonymous packets to their approximate source," in *Proc. USENIX LISA*, (New Orleans, Lousiana), pp. 319–327, Dec. 2000.

[79] K. Park and H. Lee, "On the effectiveness of probabilistic packet marking for IP traceback under Denial of Service attack," in *Proc. IEEE INFOCOM*, (Anchorage, Alaska), Apr. 2001.

[80] S. Institute, "Egress filtering v 0.2." http://www.sans.org/y2k/egress.htm, Feb. 2000.

[81] K. Park and H. Lee, "A proactive approach to distributed DoS attack prevention using route-based packet filtering," in *Proc. ACM SIGCOMM*, (San Diego, California), Aug. 2001.

[82] V. Paxson, "End-to-end Internet packet dynamics," in *Proc. SIGCOMM '97*, (Cannes, France), 1997.

[83] B. Nandy, N. Seddigh, P. Pieda, and J. Ethridge, "Intelligent Traffic Conditioners for Assured Forwarding based Differentiated Services networks," in *Proc. IFIP High Performance Networking (HPN)*, (Paris, France), June 2000.

[84] N. Seddigh, B. Nandy, and P. Pieda, "Bandwidth assurance issues for TCP flows in a Differentiated Services network," in *Proc. Globecom 99*, (Rio de Janeiro, Brazil), Dec. 1999.

[85] W. Lin, R. Zheng, and J. Hou, "How to make Assured Services more assured," in *Proc. International Conference of Network Protocols (ICNP)*, Oct. 1999.

[86] J. Heinanen and R. Guerin, "A single rate Three Color Marker." RFC 2697, Sept. 1999.

[87] J. Heinanen, T. Finland, and R. Guerin, "A two rate Three Color Marker." RFC2698, Sept. 1999.

[88] O. Bonaventure and S. de Cnodder, "A rate adaptive shaper for Differentiated Services." Internet draft, draft-bonaventure-diffserv-rashaper-00.txt, 1999.

[89] W. Fang, N. Seddigh, and B. Nandy, "A Time Sliding Window Three Colour Marker." RFC 2859, June 2000.

[90] I. Yeom and N. Reddy, "Realizing throughput guarantees in a Differentiated Services network," in *Proc. IEEE Int. Conf. on Multimedia Comp. and Systems*, June 1999.

[91] A. Feroz, S. Kalyanaraman, and A. Rao, "A TCP-Friendly traffic marker for IP Differentiated Services," in *Proc. IEEE/IFIP Eighth International Workshop on Quality of Service (IWQoS)*, (Pittsburgh, Philadelphia), June 2000.

[92] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, "The macroscopic behavior of the TCP congestion aviodance algorithm," *ACM SIGCOMM Computer Communication Review*, vol. 27, 3, pp. 67–82, 1997.

[93] I. Andrikopoulos, L. Wood, and G. Pavlo, "A fair traffic conditioner for the Assured Service in a Differentiated Services Internet," in *Proc. IEEE International Conference on Communications - ICC'99*, June 1999.

[94] H. Kim, "A fair marker." Internet draft, Apr. 1999.

[95] W. Feng, D. Kandlur, D. Saha, and K. Shin, "Understanding and improving TCP performance over networks with minimum rate guarantees," *IEEE Transactions on Networking*, vol. 7, 2, pp. 173–186, 1999.

[96] V. Jacobson, "Congestion avoidance and control," in *Proc. ACM SIGCOMM*, (Palo Alto, California), pp. 158–173, Aug. 1988.

[97] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgement options." RFC 2018, Oct. 1996.

[98] K. Fall and S. Floyd, "Simulation-based comparisons of Tahoe, Reno, and SACK TCP," *ACM Computer Communication Review*, vol. 26, 3, pp. 5–21, 1996.

[99] K. Ramakrishnan and S. Floyd, "A proposal to add Explicit Congestion Notification (ECN) to IP." RFC2481, Jan. 1999.

[100] S. Floyd and K. K. Ramakrishnan, "TCP with ECN: The Treatment of Retransmitted Data Packets ." Internet draft, draft-floyd-tcp-ecn-00.txt, Oct. 2000.

[101] S. Fahmy, *New TCP standards and flavors, High Performance TCP/IP networking*, ch. 13. Prentice Hall, Inc., 2001.

[102] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP throughput: A simple model and its empirical validation," in *Proc. ACM SIGCOMM '98*, (Vancouver, Canada), Sept. 1998.

[103] S. McCanne and S. Floyd, "Network simulator ns-2." http://www.isi.edu/nsnam/ns/, 1997.

[104] F. Shallwani, J. Ethridge, P. Pieda, and M. Baines, "Diff-Serv implementation for ns." http://www7.nortel.com:8080/CTL/#software, 2000.

[105] A. Feldmann, A. C. Gilbert, P. Huang, and W. Willinger, "Dynamics of IP traffic: A study of the role of variability and the impact of control," in *Proc. ACM SIGCOMM*, (Cambrdige, Massachusetts), pp. 301–313, 1999.

[106] C. Albuquerque, B. Vickers, and T. Suda, "Network Border Patrol," in *Proc. IEEE INFOCOM*, (Tel-Aviv, Israel), Mar. 2000.

[107] N. Seddigh, B. Nandy, and P. Pieda, "Study of TCP and UDP interaction for the AF PHB." Internet draft, 1999.

[108] H. Chow and L.-G. A., "A feedback control extension to differentiated services." Internet draft, draft-chow-diffserv-fbctrl-00.pdf, Mar. 1999.

[109] S. Shenker, C. Partridge, and R. Guerin, "Specification of guaranteed quality of service." RFC 2212, Sept. 1997.

[110] M. Listanti, F. Ricciato, S. Salsano, and L. Veltri, "Multistage deterministic worst case analysis," in *Proc. MQOS Workshop*, Jan. 2000.

[111] M. Listanti, F. Ricciato, and S. Salsano, "Delivering statistical QoS guarantees using expedited forwarding PHB in a differentiated services network," in *Proc. MQOS Workshop*, Jan. 2000.

[112] T. Bonald, A. Proutiére, and J. W. Roberts, "Statistical performance guarantees for streaming flows using expedited forwarding," in *Proc. IEEE INFOCOM*, (Anchorage, Alaska), Apr. 2001.

[113] L. Lewis, "Service level management for enterprise networks." Artech House, 1999.

[114] D. Clark, *Internet Economics*, ch. "Internet cost allocation and pricing". Eds McKnight & Bailey, MIT press, 1997.

[115] N. Semret, R. R.-F. Liao, T. Campbell, and A. A. Lazar, "Peering and provisioning of differentiated Internet services," in *Proc. IEEE INFOCOM*, (Tel-Aviv, Israel), Mar. 2000.

[116] D. Durham, K. Boyle, R. Cohen, S. Herzog, R. Rajan, and A. Sastry, "The COPS (Common Open Policy Service) Protocol." IETF RFC 2748, Jan. 2000.

[117] R. Rajan, D. Verma, S. Kamat, E. Felstaine, and S. Herzog, "A policy framework for integrated and differentiated services in the Internet," *IEEE Network*, vol. 13, 5, pp. 36–41, Sept./Oct. 1999.

[118] G. D. Rodosek and L. Lewis, "Dynamic service provisioning: A user-centric approach," in *Proc. 12th International Workshop on Distributed Systems: Operations and Management*, Oct. 15-17, 2001.

[119] T. Naik, K. Swanson, S. Mazumdar, P. Krishnan, R. Sequeira, and R. Ganesan, "IP network configurator: An Infrastructure for IP network and services provisioning," in *12th International Workshop on Distributed Systems: Operations and Management*, Oct. 15-17, 2001.

[120] Z. Liu, M. S. Squillante, and J. L. Wolf, "On maximizing service-level-agreement profits," in *Proc. ACM Conference of Electronic Commerce (EC)*, (Tampa, Florida), Oct. 2001.

[121] Y. Zhao, R. Govindan, and D. Estrin, "Residual energy scans for monitoring wireless sensor networks," in *Proc. IEEE Wireless Communication and Networking Conference*, Mar. 2002.

[122] Y. Zhao, R. Govindan, and D. Estrin, "Computing aggregates for monitoring wireless sensor networks," in *Proc. IEEE International Workshop on Sensor Network Protocols and Applications (SNPA'03)*, (Atlanta, Georgia), May 2003.

[123] C. Hsin and M. Liu, "A distributed monitoring mechanism for wireless sensor networks," in *Proc. ACM WiSe*, (Atlanta, Georgia), Sept. 2002.

VITA

VITA

Md Ahsan Habib was born in Jessore, Bangladesh. He received B.Sc. degree in Computer Science and Engineering from Bangladesh University of Engineering and Technology (BUET) in 1996. He joined to the Department of Computer Science and Engineering at BUET as a lecturer. During that period, he taught undergraduate courses, and he was a member of the consultative committee to computerize the Cabinet Division of Bangladesh. Ahsan came to Virginia Polytechnic Institute and State University, Blacksburg, Virginia in 1997, and received his masters in Computer Science in 1999. He joined the Department of Computer Sciences at Purdue University, West Lafayette, Indiana in 1999. He received his Ph.D. in 2003.

His research interests include quality of service and security issues of networking, network tomography, network economics, peer-to-peer networks, and distributed systems. He is a member of ACM, IEEE, and Upsilon Pi Epsilon honor society.