2003

On Detecting Bandwidth Theft Attacks and SLA Violations in QoS Networks

Sonia Fahmy
Purdue University, fahmy@cs.purdue.edu

Srinivas R. Avasarala

Venkatesh Prabhakar

Report Number:
03-021

ON DETECTING BANDWIDTH THEFT ATTACKS AND SLA VIOLATIONS IN QoS NETWORKS

Sonia Fahmy
Srinivas R. Avasarala
Venkatesh Prabhakar

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

CSD TR #03-021
June 2003

On Detecting Bandwidth Theft Attacks and SLA Violations in QoS Networks

Sonia Fahmy, Srinivas R. Avasarala, Venkatesh Prabhakar
Department of Computer Sciences, Purdue University
May 11, 2001

Abstract— **This paper designs a detection system to measure QoS in differentiated services networks. The system detects a number of bandwidth theft and denial of service attacks. We use a distributed monitoring approach, with measurement agents collecting information about the traffic characteristics at all the nodes of a DS domain, and reporting to a central management station that performs analysis to detect possible SLA violations, bandwidth theft or denial attacks. We are implementing and testing the system on a differentiated services testbed to evaluate if we can effectively detect such attacks. Our system can be useful in developing mechanisms for response and damage control in QoS enabled networks.**

Keywords— **service level agreements, differentiated services, network monitoring, network security**

I. Introduction

INTERNET security lapses have cost U.S. corporations 5.7 percent of their annual revenue, as reported by University of California at Davis economist Frank Bernhard. With the proliferation of heterogeneous applications and high speed networks increasing the demand for high QoS in the Internet, continuous monitoring of network activity is required to maintain confidence in the security of networks with QoS support.

The differentiated services (DS) framework for QoS provides several classes of service to meet varied user requirements of network characteristics such as bandwidth, loss rate, and latency. Packets entering a DS domain are classified and the DS field in the IP header, called the Differentiated Services Code Point (DSCP) [1], is marked at the boundary nodes. The packets then experience specific per-hop behaviors (PHBs) as they are forwarded by the interior nodes of the domain depending on their DSCP. Meaningful services can be built on these PHBs. Currently the Expedited Forwarding (EF) PHB [2] and the Assured Forwarding (AF) PHB [3] have been defined. The EF PHB can be used to build a low loss, low latency, end-to-end service through the DS domains. The AF PHB offers different levels of forwarding assurances, each with a certain drop precedence. Typically a customer has a service level agreement (SLA) with a provider that describes the expected service, customer traffic profile, and charging models. The provider uses these SLAs along with other mechanisms to provision the network appropriately. The DS architecture is shown in Figure 1.

The differences in the charging model for the various 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 from their hosts, or spoof the identity of other valid customers. As the DS framework is based on aggregation of flows into service classes based on the DSCP, valid customer traffic may experience degraded QoS as a result of the injected traffic. Taken to an extreme, the attacks can be aimed to result in denial of service altogether. This creates a need for developing an effective defense mechanism that can detect and respond to attacks on the QoS provisioned in DS networks.

In this work, we design a detection system using a distributed monitoring approach. It is distributed in the sense that we employ agents in all the nodes of the DS domain to measure traffic characteristics of the network, such as packet delays and packet loss rates. These measurements are sent to a central management station (CMS), which analyses them and detects attacks by comparing them against the negotiated SLAs.

The remainder of this paper is organized as follows. Section II discusses related work. Section III explains the design methodology of our system. Section IV explains functionality of our system components using pseudocode. Section V discusses our testbed and planned experiments. Finally, section VI lists our conclusions and discusses future work.

II. Related Work

A preliminary security analysis for the differentiated services framework is provided by [4]. The authors classify QoS attack approaches into two kinds: attacking the **network provisioning process** and attacking the **data forwarding process**. Network provisioning involves configuration of DS nodes from policy distribution points in the network called Bandwidth Brokers (BBs). This is

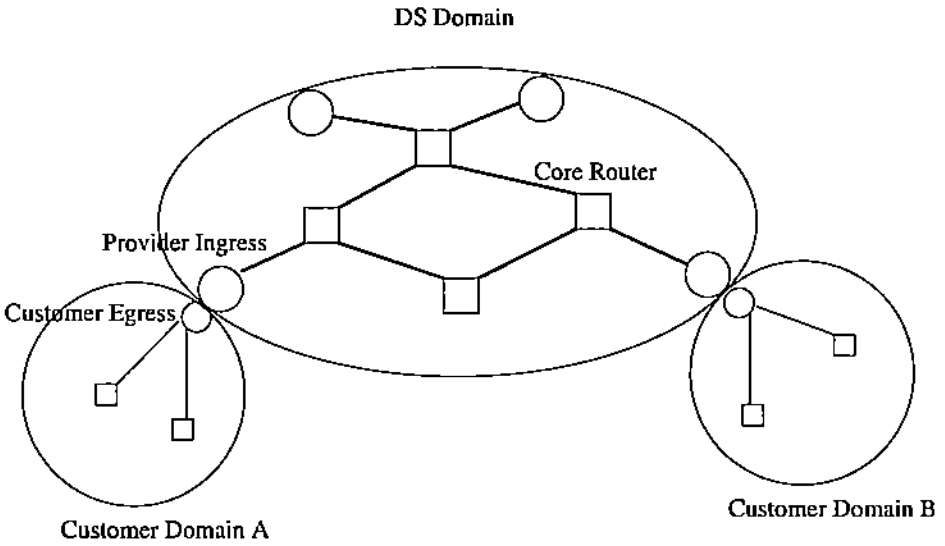


Fig. 1. DS architecture

done through automatic signaling protocols like RSVP or SNMP. 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 these 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 cause QoS degradation by causing other customer flows to experience longer delays, higher loss rates, and lower throughputs. The authors suggest that the need for intrusion detection and response systems to protect QoS in such cases.

Recent work [5] studies SLA validation. The focus is on measurement-based approaches for efficient provisioning of network resources. Their algorithm measures network characteristics, such as loss rate and delay, on a hop-by-hop basis and uses them to compute end-to-end measurements. These are used in validating the end-to-end SLA requirements. In 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 thus use an aggregation and refinement based monitoring approach. Their approach assumes that the routes used by SLA flows are known, citing VPN and MPLS provisioning. Though this is true for double ended SLAs that specify both an ingress and egress point in the network, it is not so in cases where the scope of the service is not limited to a fixed egress point.

In our design, we consider that there could potentially be different ingress and egress points for any given SLA. In addition to ensuring that at every ingress point the traffic profile conforms to the SLA, we also ensure that sum of the traffic profiles entering at different ingress points conform to the SLA. We do this by using a centralized management station which monitors the domain for SLA violations (Figure 2). Packet delay and loss rate measurements are sent by the edge and core routers respectively. This enables the management station to detect violations.

III. Design

The DS architecture [6] achieves scalability by forcing complexity out of the core of the network into boundary devices, which process lower volumes of traffic and smaller number of flows. The boundary nodes where traffic enters a domain, called Ingress routers, perform complex traffic conditioning that consists of: (1) traffic classification based on multiple fields in the packet header, (2) traffic metering to ensure conformance to profile, (3) DSCP marking, and (4) dropping, shaping or remarking out-of-profile traffic. The internal nodes, called Core routers, perform simple forwarding based on the DSCP value. SLAs between customer and provider networks are used to derive filter rules for traffic classification at ingress routers. Therefore, ingress routers with appropriate configuration of filter rules prevent traffic without valid SLAs from entering the DS domain.

Though ingress routers serve as a good first line of defense, attackers can still succeed in injecting traffic into a DS domain in a variety of ways such as:

1. Attackers can impersonate a legitimate customer by spoofing flow identity information like IP addresses, protocol and port numbers. Network filtering [7] at routers in the customer network can detect such spoofing if the attacker and the impersonated customer are on different

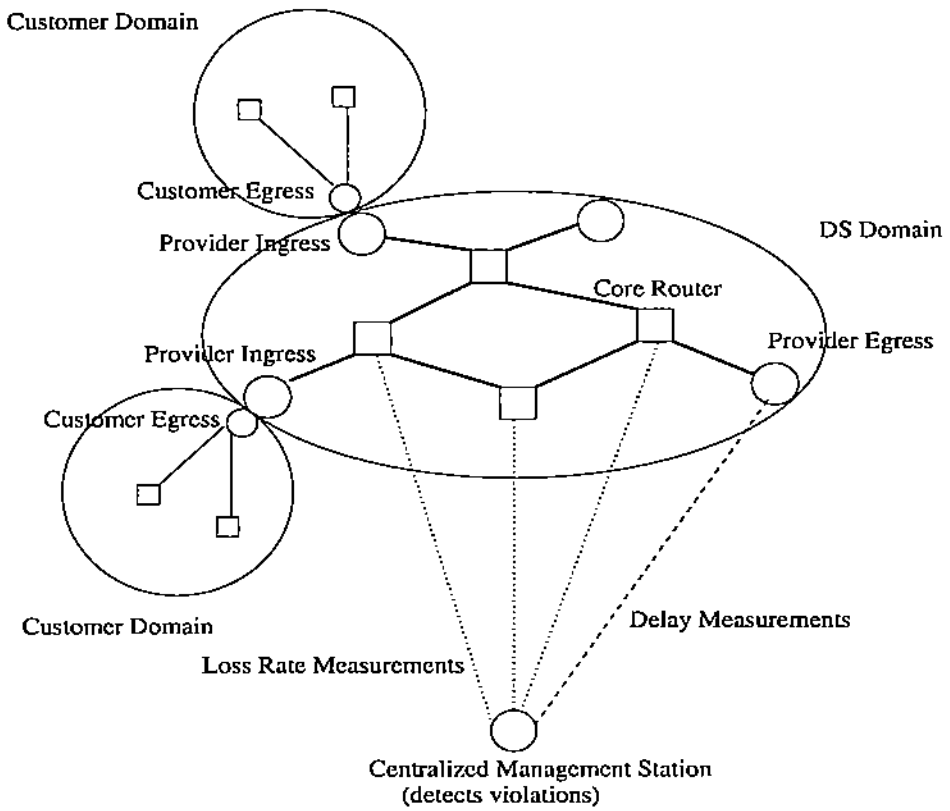


Fig. 2. An architecture for detecting SLA violations

subnets. But the attacks go unnoticed if the attacker is on the same subnet as the customer.

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.

3. If a customer has a geographically distributed network with multiple entry points into the DS domain, he/she can inject traffic into the network such that traffic profiles are not violated at any ingress router, but the sum total of such traffic exceeds the profile dictated by the SLA and disturbs the QoS of other flows.

Such intelligent attacks escape detection at the ingress router and succeed in injecting traffic into the DS domain. They require co-ordination between boundary routers and the support of core routers for detection. The changes that can be observed due to the increased traffic in the network include longer per-packet delays, higher average buffer occupancy, and higher packet drop rates. We make use of these characteristics, specifically delays and loss rate, to detect attacks and violations.

Figure 2 depicts our architecture for detecting SLA violations. We use a CMS to monitor the DS domain. Egress and core routers send delay and loss measurements respectively to the CMS for the flows in the domain. The CMS maintains a table of delays and loss rates that are updated by the egress and core routers. In addition they maintain the guarantees that are provided by the SLA for the customers. By identifying the SLA to which the flow belongs, and by comparing the delay and loss measurements against the SLA guarantees, we can identify the SLA violations.

A. *Delay Measurements*

Delay bound guarantees made by a provider network to customer traffic flows are for the delays experienced by the flows while traversing between the ingress and egress edges of the provider's domain. Delay measurements can be performed using either real customer traffic or artificially injected traffic. The first is called an **intrusive** approach. It is difficult to implement because encoding timestamps into the data packets would require changing the packets at the ingress, and rewriting the packets back to their original content at the egress after appropriate measurements. The second approach is **non-intrusive** in that we can inject packets with desired control information, so an egress router can recognize such packets, perform measurements, and remove them from the traffic stream. We adopt this second approach in our design. For each packet passing through an ingress router, with a certain pre-configured probability p_{probe}, the ingress copies the packet's IP header into a new packet, encodes the current timestamp $t_{ingress}$ into the payload, and marks the proto-

col field of the IP header with a new value, so that an egress router can recognize such packets and remove them. Additionally, the egress router computes delay for a packet of flow i as:

$$delay^i = t^i_{egress} - t^i_{ingress}$$

The egress sends a message with the packet details and the measured delay to the CMS. The format of the control messages is given later.

Synchronization between the ingress and egress routers is done by making the encoded timestamp follow a well known format like UTC, or by using a standard protocol like NTP to obtain the timestamp value at the edge routers. At the CMS, we classify the packet as belonging to a particular SLA with a customer c, and update the average packet delay of the customer's traffic as an exponential weighted moving average (EWMA):

$$avg\_delay^c_{new} = \alpha \times avg\_delay^c_{old} + (1 - \alpha) \times delay^i$$

If this observed packet delay exceeds the delay guarantee in the SLA, we conclude that it is an indication of a violation or an attack. The probability p_{probe} with which we inject control packets can be configured in a way to achieve a desired level of granularity for measurements, as well as a desired amount of artificial traffic carried by the domain.

B. Loss Measurements

Packet loss guarantees made by a provider network to a customer are for the packet losses experienced by its conformant traffic inside the provider's domain. Since packet losses in the domain are due to packet drops at core routers, we can make use of the buffer management schemes at core router queues to detect such losses. Depending on the PHB of the particular packet being dropped, we can initiate appropriate action to detect attacks and violations.

The EF PHB is typically supported by a FIFO scheme with minimal buffer depth to deliver a low delay and guaranteed throughput service. It is assumed that the network is provisioned appropriately to guarantee no packet losses for all well-behaved EF flows. Therefore, any dropped packet in the EF queue is an indication of either excess traffic from a customer or an attacker's traffic. We send a message with the details of the dropped packet from the core router informing the CMS of the drop. The CMS subsequently concludes that a violation has occurred in the domain.

The AF PHB is typically supported using a 3-color RED scheme. RED gateways detect congestion by computing the average queue size of packets at the router. When the average queue size exceeds a preset threshold, the gateway drops each arriving packet with a certain probability that is a function of the average queue size. Therefore, the packets that are dropped by a core router can be either from valid customer traffic experiencing packet drops within the limits dictated by its SLA, or due to increased congestion caused from the traffic injected by an attacker. Reporting only the dropped packet would not enable the CMS to detect attacks or violations. We must be able to distinguish between the two scenarios. We send additional information along with the dropped packet, including the loss rate of the dropped packet's flow, for that purpose.

There are several design choices for measuring the loss rates. In the first case, the core routers measure the loss for every flow as an EWMA:

$$LR^i_{new} = \alpha \times LR^i_{old} + (1 - \alpha) \times dropvalue$$

Here, $dropvalue$ is 1 for a packet that is dropped and 0 otherwise. We typically give a higher weight to the recent measurement. This eliminates the need for using a counter for measuring the total number of packets. Such a counter could typically wrap around during the life time of a flow. The measured values are reported to the CMS. At the CMS, we take the maximum of the all the reported values. If that exceeds the loss rate specified in the SLA, then we report a violation.

In the second case, we measure the number of packets dropped, and the total number of packets that traversed each core router for each flow, and report them to the CMS. At the CMS, we sum the packets dropped at different core routers for a given flow to obtain the total number of packets dropped. To obtained the total number of packets that are in the network at any time for a given flow, we need to take the maximum of the values sent by the core routers as the count of packets would be duplicated, and we cannot merely sum the total number of packets as seen by each core router. Dividing the total number of packets that are dropped by the total number of packets in the network gives us a measure of the loss ratio, which we can compare against the value specified in the SLA to detect a violation.

In the third case, we arrive at the total number of packets by measuring them at the ingress rather than at the core. This eliminates the need for the core to perform this computation for every packet that passes through it.

In the fourth case, we report the number of packets that traverse both the ingress and the egress during specific intervals of time, and calculate loss ratios as their difference divided by the number of packets seen by the ingress. The drawback of this approach is that the packets seen by the ingress in a given interval may still be in the domain, and thus can be incorrectly accounted for as dropped packets.

In the final case, we use probes just as the delay probes mentioned before. Here the ingress copies the header of

the packet with a probability p_{probe}, and fills in a special protocol field that is recognized by the egress. At specific points of time, the ingress sends the count of the number of probes sent, and the egress reports the number of probes it has received. The loss ratio can then be calculated by the CMS from this data by dividing their difference by the total number of probes sent. The disadvantage with this approach is that we are not measuring the actual loss, but heuristically estimating it by using probes. This technique works well for measuring delays. But for measuring loss rates, this may not yield accurate results. We are currently investigating a similar technique for inference of per-segment losses using Internet tomography [8].

We have adopted the first scheme in our design, as it gives a good measure of the loss, and avoids the wrap around problems associated with counters.

When making loss measurements for the AF flows, we have two important and inherently conflicting goals. One is to minimize the data exchange overhead between the core routers and the management station, and the other is to limit the amount of state and computation at the core routers. Forwarding data about all packet losses from the core router to the CMS would relieve it of all state maintenance, leaving all computation of loss ratios to the CMS, but resulting in increased data exchange overhead. Alternatively, maintaining state information about all flows including loss ratio computation would reduce the data exchange overhead, but burden the core routers. We attempt to balance the tradeoff between the two goals in our design.

We reduce the data exchange between the core routers and the CMS by sending update messages about losses at specified intervals of time. The entries in the update messages include those flows who have their loss ratios/rates within a fraction, typically 0.8, of the flow with the highest loss ratio/rate. This avoids the problem of excessive and cascaded messages that would have been sent in case we had decided to report every loss. The fraction 0.8 was used to avoid periodic sending of losses for flows with low loss, compared to the other flows. Additionally, periodic transmission of control messages would obviate the need for a reliable transmission medium for the control messages. An alternate design choice would have been to use thresholds and send control messages only in the event of losses exceeding the thresholds. This avoids unnecessary data exchange during periods that no attack or violation is suspected, because the losses are below the threshold value. But due to the inherent difficulty of setting thresholds, and the fact that periodic transmission of messages overcomes the need for reliable transmission media, we followed the approach of sending packets at specified intervals. Again, we synchronize the intervals by adopting a standard protocol like NTP at the core and edge routers to transmit the messages.

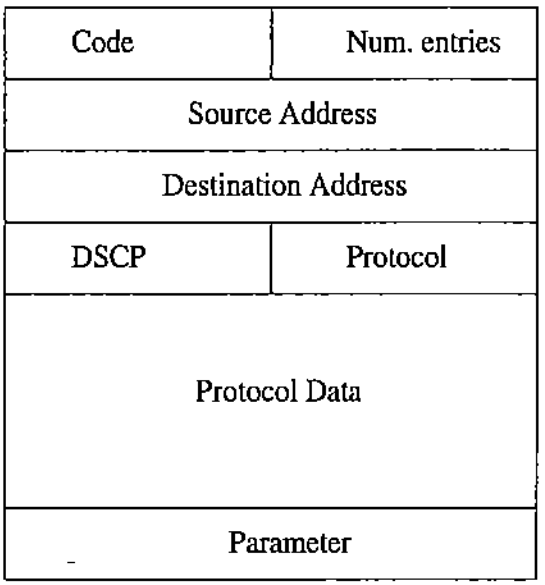
We achieve the second goal of limiting state maintenance by only maintaining state for a subset M of the total number of flows. The M flow are those experiencing the highest losses, where M is a configurable parameter determining the amount of state information to be maintained at the core router. Since we are only interested in flows that result in the aggregate traffic experiencing high loss, we exclude flows with very low loss from further analysis.

At the CMS, we maintain state information for each SLA. We maintain the flow identification information, DSCP, loss ratio/rate bound, and the reported loss for the flow at core routers. On receiving a control message from a core router, we process each entry in the message. For every flow whose loss is reported, we classify the flow as belonging to a particular SLA with a customer, and update the loss ratio/rate of the customer's traffic at this core router. Similar messages from other core routers for the same SLA would update the loss ratio/rate for the customer traffic at those core routers. We take a maximum of the loss ratios/rates of a customer's traffic reported from all the core routers as a measure of the loss experienced by the customer's traffic in the provider's domain. If this loss exceeds the loss supported by the SLA, we can conclude that the customer is in violation of its SLA or is the source of an attack.

In addition, we check to see if the DSCP that is set for the flow is correct. This is done to ensure that an attack does not occur in the form of injecting packets with a higher DSCP than allowed.

C. Control Messages

The format of the control messages is depicted in Figure 3. The egress routers send delay measurements, while the core routers send loss measurements. We use the same control message format for both. The code field distinguishes the type of message sent. A value of 0 indicates delay measurements and a value of 1 indicates loss measurements. Entries for multiple flows can be sent in the same message, and the total number of such entries is provided. The identity of the flow is given by the source address, destination address, the DSCP field, the protocol field, and the protocol data. We have left protocol data open in order to support different protocols over IP. We have considered TCP and UDP in our design, so the protocol data in this case would include the source port and the destination port. At the CMS, we must read the protocol field for each entry in order to determine how many bytes constitute the protocol data. The parameter field gives the



Code	Parameter
0	Delay
1	Loss Rate

Fig. 3. Format of Control Packets

value of the measured entity for the flow namely loss or delay. This should be consistent with the code field mentioned earlier.

IV. Algorithm Pseudo-code

In this section, we give the pseudo-code for the ingress, core and egress routers, and for the CMS.

A. Ingress Router Functionality

```
for (each pkt arrival) {
  /* classify pkt based on multiple fields */
  flow_id = multi_field_classify(pkt);
  /* meter, police, and mark or drop the packet */
  condition_traffic(pkt);
  /* generate control packet to measure delays */
  if (p_rand < p_copy) {
    copy_hdr(newpkt, pkt);
    set_proto(newpkt);
    set_timestamp(newpkt);
    enqueue(newpkt);
  }
}
```

B. Core Router Functionality

```
for (each packet arrival) {
  /* classify the packet based on DSCP */
  dscp = aggregate_classify(pkt);

  if (dscp == EF) {
    fifo_enqueue(pkt);
  }
  if (dscp == AF) {
    red_enqueue(pkt);
  }
}

fifo_enqueue(pkt)
{
  if (fifo_queue == FULL) {
    update_loss_rate(pkt_src, pkt_dst, dscp, protocol, protocol_data, DROP);
    drop(pkt);
```

```
  }
  else {
    update_loss_rate(pkt_src, pkt_dst, dscp, protocol, protocol_data, NO_DROP);
    enqueue(pkt);
  }
}

red_enqueue(pkt)
{
  bool pkt_drop;

  pkt_drop = apply_red(pkt);

  if (pkt_drop == TRUE) {
    update_loss_rate(pkt_src, pkt_dst, dscp, protocol, protocol_data, DROP);
    drop(pkt);
  }
  else {
    update_loss_rate(pkt_src, pkt_dst, dscp, protocol, protocol_data, NO_DROP);
    enqueue(pkt);
  }
}

update_loss_rate(src, dst, dscp, protocol, protocol_data, dropvalue)
{
  fid = lookup_flowid(src, dst, dscp, protocol, protocol_data);
  flow_table[fid].lossrate = alpha*flow_table[fid].lossrate +
(1−alpha)*dropvalue;
}

send_to_cms()
{
  max_rate = get_max_lossrate(flow_table);
  form_lossrate_msg(msg, flow_table, max_rate*0.8);
  send(CMS, msg);
  sleep(interval);
}
```

C. Egress Router Functionality

```
for (each arriving probe packet pkt) {
  flowid = lookup_flowid(pkt);
  delay = ingress_timestamp − current_time;
  form_delay_msg(msg, flowid, delay);
  send(CMS, msg);
  drop(pkt);
}
```

D. Management Station Functionality

```
for (every arriving msg lookup msg→code) {
case 0:
  for (each entry in msg) {
      state_update_delay(entry);
    delay_bound = sla_lookup_delay(entry);
    delay_real = state_lookup_delay(entry);
      if (delay_real > delay_bound) {
        conclude(SLA violation by the flow in the entry);
      }
  }
case 1:
  for (each entry in msg) {
    dscp_allowed = sla_lookup_dscp(entry);
    if (entry→dscp > dscp_allowed) {
        conclude(SLA violation by the flow in the entry);
      }
      if (entry→dscp == EF) {
        conclude(SLA violation by the flow in the entry);
      }
      if (entry→dscp == AF) {
          state_update_lossrate(entry);
      lossrate_bound = sla_lookup_lossrate(entry);
        lossrate_real = state_lookup_lossrate(entry);
        if (lossrate_real > lossrate_bound) {
          conclude(SLA violation by the flow in the entry);
        store(identity of the core router that sent the msg);
        }
    }
  }
}
```

V. Experiments

In this section, we discuss our testbed and planned experiments.

A. Testbed Setup

We use two PCs, wabash and ohio, running Linux, as routers in a testbed network. Wabash is used as an ingress router with filters configured for traffic classification, and Ohio is configured as a core router. We require minimal egress functionality (only to process the delay probes), and we perform that on Ohio. We use the Differentiated Services support for the Linux implementation [9]. The implementation makes use of the traffic control framework

available in recent Linux kernels, and adds support for differentiated services. The authors of that implementation use GRED, a generalized RED mechanism with a configurable number of drop priorities to support the AF PHB. Token Bucket Filters are used to police the traffic. A user level program is provided to configure the kernel to act as an Edge router or as a Core router. The CMS is run as a user process on a Sun Ultra 10 machine, ganges, running Solaris. Control messages are sent by the kernel in the core router over UDP to the CMS which, in turn, keeps the user process waiting on a fixed UDP port to receive the messages. Traffic is designed to flow from a source through the ingress router (Wabash) and then through the core router (Ohio, which also performs egress functionality), to the destination.

B. Configuration

SLA parameters at the CMS and the ingress routers, parameters for the buffer management scheme at the core routers, the weights for the EWMAs, and the probability of sending probes, p_{probe}, need to be configured. Configuring SLA parameters involves characteristics of the traffic class that the flow belongs to. For the EF class, it is the maximum data rate that can be supported with the guarantee that no losses will occur. For the AF class, it is the maximum percentage of packets that can be dropped or remarked when the traffic exceeds the profile. The ingress routers need to be aware of the SLAs to perform policing. The CMS needs to be aware of the SLAs to compare the aggregated traffic against them to detect violations. The buffer management scheme we use in the core routers is RED. The RED parameters, namely the minimum and maximum thresholds and the probability of dropping or remarking, need to be configured. These parameters are crucial to our design, because these directly affect loss measurements we make to detect SLA violations. The two EWMAs for measuring loss at the core routers and for updating delays at the CMS need to be configured. We emphasize recent measurements. Hence, we have given them a higher weight. The probability of sending delay probes p_{probe} needs to be high enough to obtain frequent updates, but low enough not to generate excess traffic. We use a probability of 0.01 (one in every hundred packets), for each flow to generate delay probes. All parameter values can, however, be modified according to the characteristics of the DS domain.

C. Traffic Generator

The traffic generator is a user level process that generates the traffic for the various flows. The input is a configuration file that has the the source IP address, destination IP address, DSCP field, data rate, and the time for which traffic should be generated for each flow. The traffic generator reads the configuration file and spawns a process for every flow for that time duration. Each process forms the appropriate IP packet and sends it using raw sockets.

D. Experiments

We are implementing our scheme in the Linux kernel. We are performing experiments to verify the correctness of our approach. Using the traffic generator, we generate data that does not conform to the SLAs for both delay and loss metrics. We will verify that such violations are detected by the CMS. Additionally, we will measure the amount of time taken to detect the violation after the time the violation occurs. This lapse in time is expected, due to the fact that we report measurements at periodic intervals of time, and typically violations can go undetected for the short span of time taken to send and process the update messages.

VI. Conclusions and Future Work

We have investigated methods to detect service level agreement violations in QoS networks. The proposed techniques can aid in detecting attacks such as service and bandwidth theft and malicious traffic remarking or injection to steal or deny service.

We plan to investigate edge-to-edge network tomography to infer per-segment loss rates, in order to eliminate the need for core router assistance. Recently proposed methods, such as [8], use a stripe of back-to-back unicast packets, for per-segment loss inference. We also plan to conduct extensive experiments to analyze the performance of our scheme. We also plan to quantify its complexity and compare it to other approaches, using statistical techniques to reduce overhead when necessary.

Acknowledgments

This work was supported by CERIAS.

References

[1] 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, December 1998, http://info.internet.isi.edu:80/in-notes/rfc/files/rfc2474.txt.

[2] V. Jacobson, K. Nichols, and K. Poduri, "An Expedited Forwarding PHB," RFC 2598, June 1999, ftp://ftp.isi.edu/in-notes/rfc2598.txt.

[3] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski, "Assured Forwarding PHB Group," RFC 2597, June 1999, ftp://ftp.isi.edu/in-notes/rfc2597.txt.

[4] Zhi Fu, S. Felix Wu, T.S. Wu, He Huang, and Fengmin Gong, "Security issues for differentiated service framework," Internet Engineering Task Force Draft, October 1999.

[5] M. C. Chan, Y.-J. Lin, and X. Wang, "A scalable monitoring approach for service level agreements validation," in *Proceedings of the International Conference on Network Protocols (ICNP)*, November 2000, pp. 37–48.

[6] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An architecture for differentiated services," RFC 2475, December 1998, http://info.internet.isi.edu:80/in-notes/rfc/files/rfc2475.txt.

[7] P. Ferguson and D. Senie, "Network ingress filtering: Defeating denial of service attacks which employ ip source address spoofing agreements performance monitoring," ftp://ftp.isi.edu/in-notes/rfc2827.txt, May 2000.

[8] Nick Duffield, Francesco Lo Presti, Vern Paxson, and Don Towsley, "Inferring link loss using striped unicast probes," in *Proceedings of the IEEE INFOCOM*, Anchorage, Alaska, April 2001, http://www.ieee-infocom.org/2001/papers/687.pdf.

[9] W. Almesberger, J. H. Salim, and A. Kuznetsov, "Differentiated services on linux," Internet Draft, June 1999, ftp://icaftp.epfl.ch/pub/linux/diffserv/misc/dsid-01.txt.