A-Serv: A Novel Architecture Providing Scalable Quality of Service

Dr. Yuke Wang *yuke@utdallas.edu*
Computer Science Department
University of Texas at Dallas
Richardson, TX 75083
USA

Dr. Lie Qian *lqian@se.edu*
Department of Chemistry, Computer & Physical Sciences
Southeastern Oklahoma State University
Durant, OK 74701
USA

Mr. Alberto Conte *Alberto.Conte@alcatel-lucent.com*
Alcatel-Lucent Bell Labs
Villarceaux
France

Dr. Xiaoyu Song *song@ee.pdx.edu*
Department of Electrical and Computer Engineering
Portland State University
Portland, OR 97207
USA

Abstract

QoS architectures define how routers process packets to ensure QoS service guarantees enforced. Existing QoS architectures such as Integrated Services (IntServ), Differentiated Services (DiffServ), and Dynamic Packet State (DPS) share one common property that the packet structure and the function of the routers are closely connected. Packets of one data flow are treated the same all the time at different routers. We propose to decouple such connection between packet structures and router functions. In our solution, packets carry as much information as possible, while routers process packets as detailed as possible until their load burden prohibits. We call such novel QoS architecture Adaptive Services (A-Serv). A-Serv utilizes our newly designed Load Adaptive Router to provide adaptive QoS to data flows. Treatments to data flows are not predefined but based on the load burden in Load Adaptive Routers. A-Serv overcomes the scalability problem of IntServ, provides better service guarantees to individual data flows than DiffServ and can be deployed incrementally. Our empirical analysis results show that compared with DiffServ architecture, A-Serv can provide differentiated services to data flows in the same DiffServ class and can provide better guaranteed QoS to data flows. Furthermore, A-Serv provides better protection to data flows than DiffServ when malicious data flows exist.

Keywords: Network QoS, QoS Architecture, Internet, Packet Format

1. INTRODUCTION

Today's Internet only provides best-effort service, where traffic is processed as quickly as possible, and there is no guarantee to the quality of service (QoS) for data flows. Here, a data flow is composed of packets with same flow ID, which is normally represented by 5-tuple (source IP address, destination IP address, transport protocol, source transport protocol port, and destination transport protocol port). QoS refers to the capability of a network to provide better service to selected network traffic. Services provided by QoS can be described by parameters

such as delay, packet loss rate, jitter, throughput, etc.

QoS architectures define mechanisms and functions to ensure that service guarantees are indeed enforced. In order to provide QoS in Internet, many QoS architectures have been proposed such as Integrated Services (IntServ) [1], Differentiated Services (DiffServ) [2] and Dynamic Packet State (DPS) [3]. However, IntServ has the problem of poor scalability; DiffServ treats all data flows as aggregate traffic, which cannot guarantee QoS to individual data flows. IntServ schedules each individual data flow in every router to ensure guaranteed services. With large number of data flows in a network, routers in IntServ face the scalability problem in order to maintain its per-flow scheduling. In DiffServ, multiple data flows are aggregated in a class and being processed as aggregated traffic. Data flows in the same class will receive the same service. This reduces the service differentiation between individual data flows. Furthermore, when malicious data flows in a class send uncontrolled data volume, all other data flows in the same class will be punished in their performance. DPS needs long time for data flows to get their fair rate, requires special scheduling scheme to be installed in each core router to perform scheduling operation and requires modification in packet header formats.

IntServ, DiffServ, and DPS architectures share one common property that packet structures and router functions are closely connected. Therefore treatments to data flows are predefined. In IntServ, per-flow treatment is binded with the 5-tuple flow ID in each packet. In DiffServ, class based aggregated treatments are binded with the DiffServ class ID in each packet. In DPS, treatments are binded with a 17 bit DPS header defined in IP header. In these QoS architectures, packets of one data flow are treated the same all the time at different routers based on their packet header structure. We propose to decouple such connection between packet structures and router functions. Packets will carry as much information as possible, while routers process the packets as detailed as possible until their load burden prohibits. We call such novel QoS architecture Adaptive Services (A-Serv).

In this paper, we first introduce a Load Adaptive Router, which treats data flows differently according to the load burden of the router. Then we propose a novel QoS architecture, Adaptive Services (A-Serv), which uses Load Adaptive Routers as core routers. Based on router's processing capability, A-Serv keeps as much data flow state information as possible at core routers. In A-Serv, a data flow can be treated either as per flow or aggregate traffic in the core routers depending on the core routers' load burden. Behavior of Load Adaptive Routers in A-Serv architecture is presented in pseudo codes. A new protocol is also proposed to be used in A-Serv to provide guaranteed per-flow treatment for data flows with high priority.

A-Serv is free from Intserv's scalability problem because it adjusts its treatments to data flows based on its actual processing, storage and scheduling burden. On the other hand, A-Serv can provide better service guarantee to individual data flow than DiffServ by trying to make full usage of routers' processing ability to guarantee services of as many as possible data flows. In addition, A-Serv can be deployed incrementally on existing QoS architectures. Our empirical analysis results show that A-Serv can provide differentiated service to data flows in the same DiffServ class and protect data flows better than DiffServ from malicious data flow's interference.

The rest of this paper is organized as following: Section 2 overviews the network model, related works of network QoS and existing QoS architectures. In Section 3, we introduce the Load Adaptive Router, A-Serv architecture, routers' behavior in A-Serv and the proposed protocol to ensure end-to-end per-flow treatment. The empirical analysis results are presented in Section 4. The conclusion and future work are given in Section 5.

2. RELATED WORKS

In this section, we present our network model, and a brief review of network QoS and existing QoS architectures, such as IntServ [1], DiffServ [2] and DPS [3].

2.1 Network Model

In our network model, all routers in a domain are categorized into edge routers and core routers. Edge routers are boundary points at which data flows enter (ingress edge router) or leave (egress edge router) this domain. Edge routers connect to access networks or to edge routers in other domains. Core routers are internal routers that connect different edge routers in the same domain.

2.2 Network QoS

With the appearance of more and more network applications requiring contents to be delivered with ensured quality, best effort service is not capable of handling all kinds of network applications anymore. Quality of Service (QoS) refers to the capability of a network to provide better service to selected network traffic or data flows. QoS is the ability of a network element (e.g. an application, host or router) to have some degree of assurance that its traffic and service requirements can be satisfied [13]. Components in QoS include 1) QoS architectures, 2) admission control, 3) scheduling and policing, and 4) QoS routing.

QoS architectures define how routers process packets to ensure that QoS service guarantees are enforced. In order to provide QoS on the Internet, many QoS architectures have been proposed such as Integrated Services (IntServ) [1], Differentiated Services (DiffServ) [2] and Dynamic Packet State (DPS) [3], etc.

To fulfill QoS requirements, admission control processes are used to decide if the admission of a new incoming flow will affect the QoS of existing traffic in the network. Admission control algorithms can be classified into two categories: traffic descriptor-based and measurement-based admission control. Traffic descriptor-based admission control [14-21] uses the traffic characterizations of both new and existing flows to estimate the achievable QoS performance and perform admission control. Measurement-based admission control algorithms [22-31] use traffic characterizations of new incoming flows and measurements of existing traffic as inputs to perform admission control.

QoS scheduling determines the sending sequence of packets from different data flows on a link with fixed bandwidth. Different data flows receive their shares of the overall bandwidth through the scheduling mechanisms in a router. Scheduling mechanisms have been widely studied in literatures [31-47].

When a data flow is admitted in a network through admission control processes, the parameters provided by the data flow for admission control become a contract between the source of the admitted data flow and the network. Data flows violating the contract will affect the service quality received by other admitted traffic in the network. Thus regulators are needed to detect and regulate the traffic that violates pre-defined parameters, which is also referred as traffic policing. Some proposed policing mechanisms include Leaky Bucket, Token Bucket, the Triggered Jumping Window [48], Dual Leaky Bucket [49], BLLB [50], FBLLB [51], etc.

The current Internet uses shortest path routing, in which packets are always delivered on the shortest path between a source and a destination. While QoS is considered, the shortest path for a data flow may not be acceptable or optimal in resource usage due to different resource consumptions on its path. In QoS routing, paths for data flows would be determined based on the knowledge of resource availability in the network, as well as the QoS requirements of data flows. Thus the original shortest path routing problem becomes a multiple constraints path finding problem, which is known as NP-complete [52]. To solve the NP-complete multiple constraints path finding problem, many heuristics or approximation algorithms have been proposed to solve it sub-optimally [53-62].

2.3 QoS Architectures

IntServ architecture is characterized by resource reservation for each data flow through RSVP signaling protocol [4, 5]. A data flow requesting specific QoS guarantees is required to initiate a

setup procedure using RSVP. RSVP sets up "soft states" in each router along the path from source to destination specifying the resource requirements of the initiating data flow. The reservations remain valid as long as the data flow is active, and expire if not refreshed periodically. All routers, including edge routers and core routers, keep the per-flow state information and allocate resources (such as buffer space and link bandwidth) to each passing data flow. This per-flow service model achieves the maximum flexibility as it can meet QoS requirements of individual flows. In IntServ, Packets are identified by their flow ID (5-tuple with 104 bits in IPv4 or 296 bits in IPv6 [6]). Guaranteed services can be provided to each individual data flow. The major scalability problem of IntServ is that data flow state information increases proportionally with the number of data flows.

In DiffServ, packets are marked in their DS field (6 bits differentiated services code point defined in IPv4 and IPv6 packet header) with different DiffServ class IDs to create 2-8 DiffServ classes. Packets are classified and assigned DiffServ class ID at ingress edge routers. Subsequent packet classification and forwarding in core routers are based on the DiffServ class ID in each packet. Data flows in the same DiffServ class are treated as aggregate traffic in core routers. Data packets with different class IDs receive different services (e.g. Expedited Forwarding (EF) [7] and Assured Forwarding (AF) [8]). DiffServ is scalable in core routers for the limited number of DiffServ classes, which bounds the amount of state information maintained by each core router. One problem of DiffServ is that service to data flows in aggregate traffic can be affected by other data flows in the same DiffServ class and therefore QoS requirements of an individual data flow cannot be guaranteed even with EF treatment [9,10]. The scalability achieved by DiffServ is at the expense of reduced performance [11].

While maintaining the benefit of scalability, as a variant of DiffServ, proportional QoS model has been proposed [63]. This model provides network operators with adjustable and consistent differentiation between service classes. With this QoS model, the service differentiation level between classes can be adjusted according to prespecified factors. Various QoS metrics for packet loss and delay have been investigated in proportional paradigm [63-66].

In the DPS architecture [3], packets, instead of routers, carry flow state information. Ingress edge routers insert data flow state information into each packet (17 bits defined in DPS architecture). Core routers process each packet according to the attached state information and the routers' internal state information, which does not increase with the number of data flows. Before forwarding a packet, core routers update the flow state information in the packet and the internal state information in routers. A general core stateless framework Virtual Time Reference System (VTRS) was developed to ensure end to end delay. VTRS includes 1) edge traffic conditioning, 2) packet state carried by packets, 3) per-hop virtual time reference/update mechanism. VTRS doesn't mandate any scheduling mechanisms. By such means, DPS provides scalable QoS services with improved performance than DiffServ [11]. One problem of DPS is that it is hard for core routers to estimate the fair share rate rapidly and correctly for individual data flows. As a result, it needs long time for data flows to get their fair rates [67]. DPS requires special scheduling scheme to be installed in each core router to perform scheduling operation and modification in each packet's header. Therefore deploying DPS requires changing all edge and core routers in one domain and gradual deployment is not achievable.

3. NEW ARCHITECTURE USING LOAD ADAPTIVE ROUTER

In this section, we first introduce a newly designed Load Adaptive Router. Load Adaptive Routers can be deployed as core routers in single or multi QoS domains and can provide scalable, differentiated and adaptive services. Then, we use Load Adaptive Routers to design a new adaptive QoS architecture, A-Serv. Behavior of routers in A-Serv is presented by pseudo code. A protocol is designed to provide end-to-end per-flow treatment to data flows.

3.1 Load Adaptive Routers

Load adaptive routers are designed to be deployed as core routers in QoS capable domains as

shown in Figure 1. ER1~ER7 are 7 edge routers and the rests are core routers, which can be implemented as load adaptive routers.

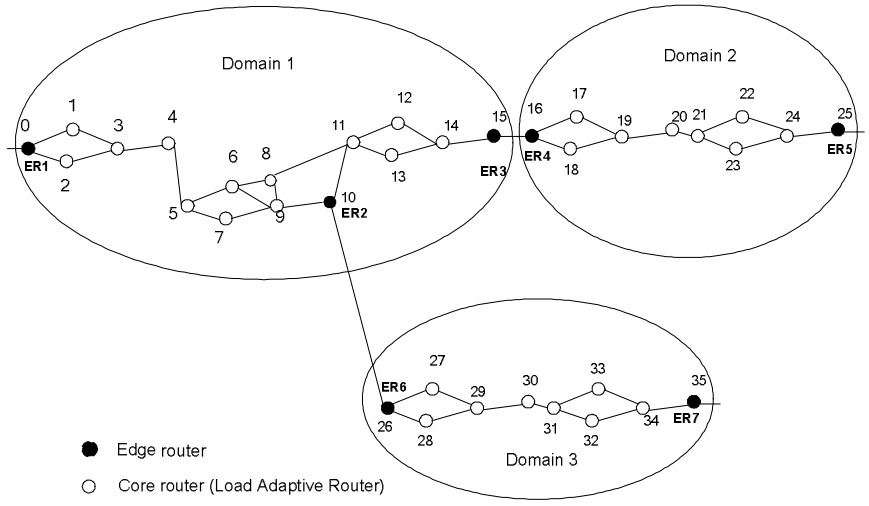

FIGURE 1: Load Adaptive Router in QoS Capable Domain

The key idea of this design is to decouple packet structure from the way packets are processed in core routers. In IntServ, routers always treat packets based on their flow ID. By identifying the data flow each packet belongs to, the packet is treated based on the resources reserved by that data flow. In DiffServ, packets are treated in core routers based on their DiffServ class ID in header. In DPS architecture, packets are processed by routers based on the scheduling information in packet headers. In our newly designed load adaptive router, a packet can be treated differently in routers depending on the load of each router while the packet carries as much information as needed for all possible processing.

Here we use the packet header formats of TCP/IP packet in IPv4 and IPv6 shown in Figure 2 to illustrate the load adaptive idea. One thing worth mentioning is that the load adaptive idea itself doesn't involve any assumption from current IP network and it is not limited in TCP/IP packet and can be easily adapted to other existing or future network standards. When the load adaptive scheme is implemented in TCP/IP network, packets can be treated based on their flow ID (104 bits in IPv4 or 296 bits in IPv6 in Figure 2) as a per flow, which is equivalent to IntServ, or be treated based on class ID (6 bits in DS field in Figure 2) as aggregate traffic, which is equivalent to DiffServ, or anything in the middle by identifying a number of bits in the packet header (the number of bits between 6 and 104, which can be inserted in the option field in IPv4 and IPv6). Instead of having 2-8 classes as in DiffServ, we could have a class to contain only one data flow if that data flow is more important. Instead of having one per-flow state for each data flow, we could aggregate data flows together into different kinds of classes, which are then treated as aggregate traffic, but not necessarily to be the DiffServ classes with 6 bit DiffServ class ID.

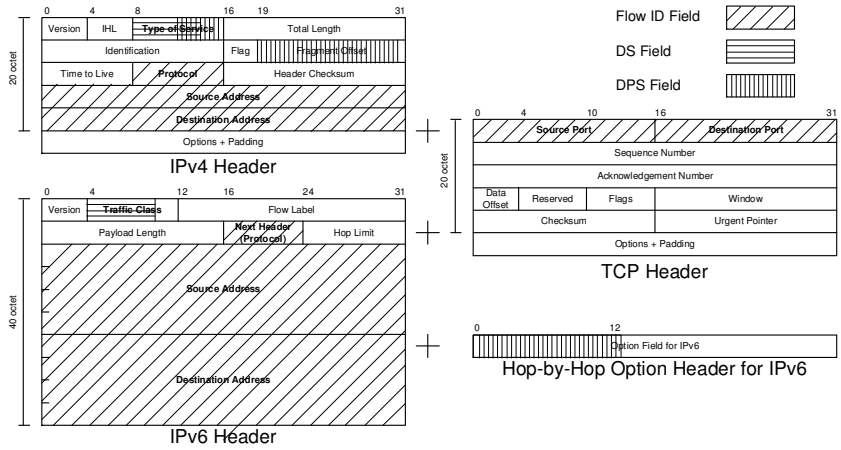

FIGURE 2: 5 -Tuple Flow ID, DS and DPS Field in TCP/IP Header

Using load adaptive routers, one data flow can be in different classes at different time in the same router. When a load adaptive router's load burden decreases, data flows will be treated based on more bits in the packet header, and one existing class may be divided into several sub-classes with resource reserved for each sub-class. When the load burden increases, data flows will be treated based on fewer bits in the packet header, and some existing classes are combined into one class. Furthermore, one data flow can receive different treatments in different load adaptive routers. At load adaptive routers with heavy load burden, a data flow has more chances to be treated as aggregate traffic. At load adaptive routers with light load burden, a data flow has more chance to be treated as per flow. The processing received by data flows is not predefined by the packet header structure such as in IntServ, DiffServ and DPS. The processing is decoupled from the packet header structure and is adaptive to router's load burden, which prevents the scalability problem and makes best use of the router's processing capability.

We give an example to illustrate the basic scheme of a load adaptive router. Assume except the 5-tuple data flow ID, there are 6-bit DS field and 17-bit DPS ID in each packet's header as shown in Figure 2. At ingress edge routers, each packet is assigned 6-bit DS field and 17-bit DPS ID (which has different content as that in DPS architecture and we don't assume any specific marking scheme for generality). If a load adaptive router is lightly loaded, it treats most data flows as per flows based on 5-tuple data flow ID only. Resources, such as bandwidth and buffer, are reserved for each individual data flow in this router. On the other hand, a router can treat most data flows based on only the 6-bit DS field as in DiffServ when it is extremely heavily loaded. Other than these two extreme situations, load adaptive routers can treat data flows as aggregate traffic based on the 6-bit DS field together with any number of bits in the 17-bit DPS ID. The more bits used in DPS ID, the more sub-classes exist in each class (e.g. 131072 sub-classes for 17 bits, 32 sub-classes for 5 bits) and cost more storage space and scheduling power in the router. In such way, each load adaptive router can adjust the number of bits it uses in packet header adaptively based on its available storage space and scheduling power.

3.2 A-Serv: Adaptive Services Provided by Load Adaptive Router

In this subsection, we introduce a new QoS architecture called A-Serv, which utilizes load adaptive routers as core routers to provide Adaptive Services to data flows. In A-Serv, edge routers insert default data flow aggregation information (such as class ID) into packets' header. The default data flow aggregation can be the DiffServ class or any other aggregation scheme defined in the future.

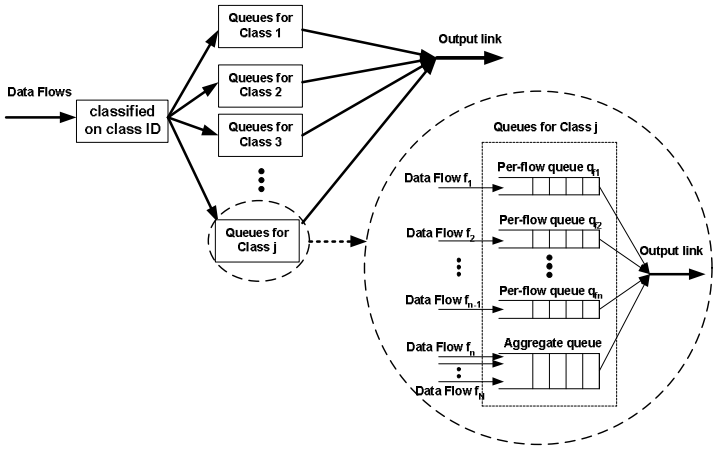

FIGURE 3: Data Flow Treatment in A-Serv

A. Adaptive Service

In A-Serv, data flows' treatments in a load adaptive router are shown in Figure 3. A load adaptive router first classifies data flows based on its default class ID. Within each class, a data flow is treated either as per flow or aggregate traffic. For example, data flow f_1 is treated as per flow in class j in Figure 3. A separate queue q_{f1} with desired buffer and bandwidth assignment would be set up for data flow f_1. Thus f_1 receives guaranteed service at this load adaptive router. Here we say f_1 is treated as per flow (f_1 receives per-flow treatment) and q_{f1} is the per-flow queue for data flow f_1. Packets belonging to f_1 are inserted into queue q_{f1} in this router after being identified by their class and flow ID. For data flow f_n, all packets of f_n would be identified by the class ID only in this router and be inserted into the aggregate queue for class j, which is responsible for forwarding all the data flows being treated as aggregate traffic in class j. Here we say f_n is treated as aggregate traffic (f_n receives aggregate treatment) in class j. A-Serv is a two level adaptive implementation of load adaptive routers.

B. Load Burden Estimation

Load adaptive routers use load burden to determine treatments to data flows (per-flow or aggregate). In our design, load burden is evaluated through the number of existing queues in load adaptive routers. The number of existing queues affects processing complexity, scheduling complexity and consumed storage space of load adaptive routers. The more existing queues in a router, the more storage space is used to store state information for each queue which records queue's resource reservation; the more time needed to perform classification for incoming packets to decide destination queues to insert the packets; the more time spent on scheduling the packets transmission. For example, in General Processor Sharing (GPS) scheduling the start/finish virtual time for each packet is computed for scheduling purpose. Such computation has complexity of O(logn), where n is the number of queues in scheduling process [68].

However, number of queues is not the only choice for load burden estimation. Other factors that constraint the scalability can also be adopted into the load burden criteria. For example, the load burden criteria could be the utilization of buffers, if the buffer space used to store per flow information becomes the bottleneck and limits the scalability. Also more sophisticated situation in load burden evaluation arises when we consider the data flow's traffic properties such as bustiness, traffic volume, packet size, etc.

In this paper, we only consider the situation when the number of queues in the router cases scalability problem and we use it as load burden criteria in our design.

C. General Functions in a Load Adaptive Router

In this part, we present pseudo codes for general functions in a load adaptive router, including

new data flow establishment, incoming packet processing and existing data flow's termination.

In A-Serv architecture, there is an upper bound N on the number of existing queues in a load adaptive router. N is decided by the router's processing ability and storage capacity. Different load adaptive routers could have different values for N and this enables gradual deployment and upgrade. In one router r, N is partitioned into n_j, j=1,2,..k, when there are k default classes defined in total. n_j is the number of queues that could be supported simultaneously in this router for class j. Each default class reserves one queue for aggregate traffic. Therefore, simultaneously at most n_{j-1} data flows in class j can be treated as per flows.

Initially, the resources such as buffer space and output link capacity assigned for each class j are all assigned to the single aggregate queue of class j in a router. When a new per-flow queue is created, the load adaptive router extracts corresponding resources from the aggregate queue and assigns them to the new per-flow queue. When the number of per-flow queues in class j reaches its upper bound n_{j-1}, new data flows arriving later in class j will be put into the aggregate queue for class j and be treated as aggregate traffic. In this case, we don't consider priority differences between the data flows in the same class. Per-flow treatments are provided to data flows in each class in First Come First Serve mode.

The resource allocation is shown in Figure 4. The initial resources allocation between classes is determined by the network domain manager or operator. The resources within each class are assigned to per-flow queues and the aggregate queue as we described above.

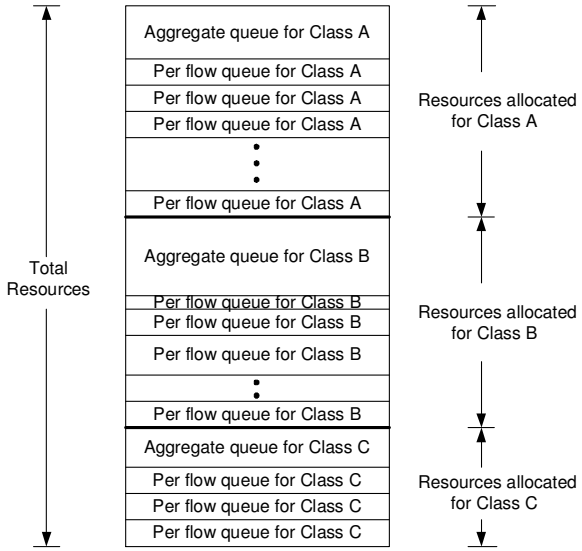

FIGURE 4: Resource Allocation in A-Serv

When a new data flow needs to be established, a request packet (or the first data packet in the data flow) will be processed in the load adaptive routers on its path following pseudo codes in Figure 5. Each router will try to create a per-flow queue for the new data flow. If the per-flow queue quota, n_j, is used up by existing data flows, the new data flow will be treated in aggregate queue.

New Incoming Data Flow Establishment Request Processing

Notations:

$N_{j,current}$: Number of Per-Flow queues in the router for class j
$N_{j,bound}$: Upper bound on the number of queues for class j
$R_{j,aggre}$: Resource (e.g. buffer) allocated to aggregate queue in class j
$B_{j,aggre}$: Bandwidth allocated to aggregate queue in class j
$B_{request}$: Bandwidth requested by New flow i in class j
$R_{request}$: Resource requested by flow i in class j

New Flow Creation Procedure(New flow i in class j):

if ($N_{j,bound}$>$N_{j,current}$+1){
$R_{j,aggre}$=$R_{j,aggre}$-$R_{request}$
$B_{j,aggre}$=$B_{j,aggre}$-$B_{request}$
Create per-flow queue f_i($B_{request}$,$R_{request}$) in class j
Add i's flow ID and f_i's information into forwarding table
$N_{j,current}$ =$N_{j,current}$+1
}
else{do nothing}//Flow i be treated in aggregate queue g_j in class j
Forward request packet to next hop router on its path

End Procedure

FIGURE 5: New Data Flow Establishment

After data flow's establishment, when subsequent data packets come into the router for forwarding, the router will first check the packet's header for class ID, say class j. Then the router will search its flow ID in the forwarding table of class j. If there is a match, this packet will be inserted into the corresponding queue for the matched entry in forwarding table. If there is no match, the packet will be inserted into the aggregate queue for class j. Such procedure for packet forwarding is depicted as Figure 6.

Packet Forwarding Processing

Packet Forwarding Procedure (Packet P: belonging to flow i in class j):

Acquire class id j from packet header

Search i in class j's forwarding table

if (per-flow queue f_i is found)
Insert P into queue f_i
else
Insert P into queue g_j (Aggregate queue for class j)

End Procedure

FIGURE 6: Packet Forwarding in A-Serv

When a data flow treated as aggregated traffic terminates, nothing needs to be done in load adaptive router for resource reallocation or queue removals. When a per-flow treated data flow terminates in class j, the load adaptive router removes the per-flow queue of this terminated data flow and assigns the removed per-flow queue's resources back to the aggregate queue in class j. The per-flow treated quota left by the terminated data flow could be used by another data flow in class j for its per-flow treatment.

There are two options for queue management in A-Serv architecture, preemptive and non-preemptive modes. Whether A-Serv is in preemptive mode or non-preemptive mode determines if the load adaptive router can create a per-flow queue for an existing data flow in aggregate queue.

In preemptive A-Serv, when a per-flow queue quota is available, the load adaptive router doesn't

need to wait for the arrival of a new data flow. The router could create a per-flow queue for the next incoming packet p whose data flow is processed in the aggregate queue in class j. The new per-flow queue is established based on QoS requirement information in packet p's header and treats the data flow of packet p as per flow thereafter. For the preemptive purpose, QoS requirement information needs to be inserted in every packet in preemptive A-Serv, which may cause security problem. In preemptive A-Serv, the same data flow could be moved from aggregate queue to a per-flow queue. Therefore, the same data flow may receive different treatments within the same load adaptive router at different time. Data flow termination under preemptive mode is depicted in Figure 7.

In non-preemptive A-Serv, load adaptive router will not move the aggregate treated data flows to a per-flow queue even if there are per-flow queue quota available. For this reason, no per-flow queue will be setup based on the packet coming after the data flow starts. Thus, no QoS requirement information needs to be maintained in data packet header except the first packet of a data flow or just use signaling protocol such as RSVP before the transmission of a data flow. In non-preemptive A-Serv, it could be inefficient in the router's processing and storage resource usage, because some available per-flow quota could be wasted while some data flows are treated as aggregate traffic. Also, the treatments received by the data flows don't change within the same load adaptive router during its life time. Data flow termination under non-preemptive mode is depicted in Figure 8.

Data Flow Termination Processing

Preemptive Mode

Notations

$N_{j,current}$: Number of Per-Flow queues in the router for class j
$N_{j,bound}$: Upper bound on the number of queues for class j
$R_{j,aggre}$: Resource (e.g. buffer) allocated to aggregate queue in class j
$B_{j,aggre}$: Bandwidth allocated to aggregate queue in class j

$B_{request}$: Bandwidth requested by flow i in class j
$R_{request}$: Resource requested by flow i in class j

$B'_{request}$: Bandwidth requested by flow k in class j
$R'_{request}$: Resource requested by flow k in class j

Existing Flow Termination Procedure(data flow i in class j):

```
if (found i's flow ID and its per-flow queue f_i in forwarding table){
    R_j,aggre=R_j,aggre+R_request
    B_j,aggre=B_j,aggre+B_request
    Delete per-flow queue f_i in class j
    Remove i's flow ID and f_i's information from forwarding table
    while(1){
      Check the next incoming packet p of data flow k
      if(k in class j AND k not in forwarding table) {break}
      else {call packet forwarding procedure(p)}
    }
    Check the header of p to retrieve B'_request and R'_request
    R_j,aggre=R_j,aggre-R'_request
    B_j,aggre=B_j,aggre-B'_request
    Create per-flow queue f_k(B'_request,R'_request)in class j
    Add k's flow ID and f_k's information into forwarding table
}
else{do nothing}
```

End Procedure

FIGURE 7: Data Flow Termination in A-Serv (Preemptive)

Data Flow Termination Processing
Non-preemptive Mode

Notations:
$N_{j,current}$: Number of Per-Flow queues in the router for class j
$N_{j,bound}$: Upper bound on the number of queues for class j
$R_{j,aggre}$: Resource (e.g. buffer) allocated to aggregate queue in class j
$B_{j,aggre}$: Bandwidth allocated to aggregate queue in class j

$B_{request}$: Bandwidth requested by flow i in class j
$R_{request}$: Resource requested by flow i in class j

Existing Flow Termination Procedure(data flow i in class j):
if (found i's flow ID and its per-flow queue f_i in forwarding table){
$R_{j,aggre}=R_{j,aggre}+R_{request}$
$B_{j,aggre}=B_{j,aggre}+B_{request}$
Delete per-flow queue f_i in class j
Remove i's flow ID and f_i's information from forwarding table
$N_{j,current}=N_{j,current}-1$
}
else{do nothing}
End Procedure

FIGURE 8: Data Flow Termination in A-Serv (Non-Preemptive)

D. Coordinated Load Adaptive Router
In order to provide end-to-end per flow service, coordination between loaded adaptive routers is needed to selectively provide data flows with per-flow service. In this subsection, we propose a RSVP [4] like protocol for this purpose.

We add a flag, request type flag, into data flow establishment request packets. If the request type flag is set to 0, we use $P_{flag=0}$ to denote such packet, the new data flow can accept any service on its route, per-flow or aggregate. Request packet $P_{flag=1}$ demands that the data flow is either denied or receive per-flow service in every router on its path. After being established, we also add the request type flag into the forwarding table.

Request packet $P_{flag=0}$ will be processed as Figure 5, in which aggregate or per-flow treatment could be given depending on the per-flow quota availability at that moment.

Request packet $P_{flag=1}$ will be processed as Figure 9, in which the router will try to create a queue for the new data flow. If there is no quota left, the router will try to move one per-flow treated data flow with flag=0 to aggregate queue so that one quota will become available to the new data flow. If no per-flow treated data flow could be moved to aggregate queue (all of them have flag=1), a deny message will be sent back to requesting data flow's source along the reverse route.

Upon the receiving of a deny message for data flow i's request, the router will perform data flow termination for flow i as described in Figure 7 or Figure 8. The new data flow's destination host sends a reply to the source after it receives the request packet to confirm that the route has been established with per-flow service guaranteed in every hop. The source of the data flow can start its data transmission after receiving the confirmation packet.

3.3 Compare A-Serv With Existing QoS Architectures
Using the upper bound to limit the number of queues in the router prevents the router from being overwhelmed by huge number of per-flow treated data flows. Storage for per-flow state information and scheduling burden can be anticipated and controlled to eliminate scalability problem. Furthermore, routers could be configured to maximize the utilization of the router's storage and processing ability.

New Incoming Data Flow Establishment Request Processing
Receive $P_{flag=1}$

Notations:

$N_{j,current}$: Number of Per-Flow queues in the router for class j
$N_{j,bound}$: Upper bound on the number of queues for class j
$R_{j,aggre}$: Resource (e.g. buffer) allocated to aggregate queue in class j
$B_{j,aggre}$: Bandwidth allocated to aggregate queue in class j

$B_{request}$: Bandwidth requested by flow i in class j
$R_{request}$: Resource requested by flow i in class j

New Flow Creation Procedure(New flow i in class j):

```
if (N_j,bound > N_j,current + 1){
    R_j,aggre = R_j,aggre - R_request
    B_j,aggre = B_j,aggre - B_request
    Create per-flow queue f_i(B_request, R_request) in class j
    Add i's flow ID and f_i's information into forwarding table
    N_j,current = N_j,current + 1
    Forward request packet to next hop router on its path
    return
}
Search forwarding table in class j for flow t, whose type flag is 0
if (flow t exists){
    Call "Existing Flow Termination Procedure (t, j)"
    Call "New Flow Creation Procedure (i,j)"
    Forward request packet to next hop router on its path
    return
}
else {Deny the request, send deny message toward sender}
End Procedure
```

FIGURE 9: All Per-Flow Treatment Reservation Procedure in A-Serv

Compared with IntServ, A-Serv architecture doesn't have the problem of scalability at core routers. When the number of data flow exceeds the processing ability or storage space of the load adaptive router, the rest data flows will be treated as aggregate traffic, which won't increase the burden of state information storage and scheduling complexity in routers.

Compared with DiffServ, A-Serv doesn't treat all data flows as aggregate traffic. Load adaptive routers are always trying to fully utilize their processing ability to guarantee the QoS to as many data flows as possible by treating them as per flows. Furthermore, when malicious data flows exist, they will affect the service received by all data flows in the same class in DiffServ. In A-Serv, per-flow treated data flows can be protected from being affected by the malicious data flow. On the other hand, per-flow treated malicious data flow won't be able to affect other data flows.

A-Serv can be deployed incrementally in existing IntServ or DiffServ network. In IntServ architecture, to deal with IntServ's scalability problem, bottleneck core routers in IntServ domain can be replaced with load adaptive routers. After that, data flows can receive per-flow treatments in all IntServ core routers and adaptive services in load adaptive routers. In DiffServ architecture, we can replace any core router in DiffServ domain and deem the unchanged DiffServ core routers as load adaptive routers with very limited processing ability (can only support one queue in each DiffServ class).

4. EMPIRICAL ANALYSIS OF A-SERV ARCHITECTURE

In this section, we present our empirical analysis to evaluate the A-Serv architecture through comparing it with DiffServ. In our analysis, we use the multi-domain topology shown in Figure 1. In DiffServ architecture, we consider data flows in one DiffServ class with 4Mbps bandwidth assigned. In A-Serv architecture, we use the DiffServ class as default aggregation scheme with

4Mbps bandwidth for comparison. NS-2 simulator [12] is used to measure delay suffered by data flows. We consider two kinds of traffic, multiple non-malicious data flows and one malicious data flow. The non-malicious (normal) data flows follow ON-OFF Markov process (average ON time=0.5 second, average OFF time=0.5 second, and peak rate 200Kbps). The A-Serv architecture is implemented in non-preemptive mode. We set the maximum number of queues equals to 20 for each output link in A-Serv.

4.1 Scenario With No Malicious Data Flow

	# of data flows	# of per-flow treatment	# of aggregate treatment	Group
ER1~ER5 (set A)	6	18	0	Group 1
	4	13	5	Group 2
	3	11	7	
	3	6	12	Group 3
ER1~ER7 (set B)	6	15	0	Group 1
	3	10	5	Group 2
	4	8	7	
	3	3	12	Group 3

TABLE 1: Treatments for Non-Malicious Data Flow Case

We first look at the scenario when only non-malicious data flows exist in the network. In this scenario, we setup 64 data flows, among which 32 are multidomain data flows for performance observation and the rest 32 are single domain data flows. Multidomain data flows include 16 data flows from ER1 to ER5 (data flow set A), and 16 data flows from ER1 to ER7 (data flow set B). Single domain data flows include 16 data flows from ER4 to ER5 in domain 2, and 16 from ER6 to ER7 in domain 3.

In A-Serv, with the RSVP like protocol presented in Section 3, different combinations of treatments are realized in set A and B. The treatments summary of data flows in A-Serv architecture is listed in Table I. There are 6 data flows between ER1~ER5 and 6 data flows between ER1~ER7, requested all per-flow treatments along their routes (Group 1). The rest data flows received mixed treatment, some with more per-flow treatments (Group 2); some with more aggregate treatments (Group 3).

Due to the similarity existing between set A and B, we only show the analysis of set A data flow's performance. In A-Serv, the per-flow reservation is 175Kbps for each data flow. Under both DiffServ and A-Serv architectures, we measure the average end-to-end delays, which are plotted in Figure 10.

In Figure 10, under A-Serv, data flows in group 1 (Figure 10.b) have better performance than DiffServ (Figure 10.a) for their per-flow treatments in all routers (Average Delay 0.0032 second VS 0.0084 second in DiffServ). Data flows in group 1 receive the reserved service without interference from any other data flows in any router, which gives the best performance to these data flows. On the contrast, data flows' performance in group 3 (Figure 10.d, Average Delay 0.013 second) is worse than that in DiffServ (Figure 10.a, Average Delay 0.0084 second). These data flows are treated as aggregate traffic in all 12 routers where data flows in other sets exist. The service received by each data flow in the aggregated traffic could not be guaranteed and could be affected by the behavior of other data flows. Therefore, the data flows in the aggregated traffic receive worse service than the data flows treated as per flow. Data flows in group 2 receive similar number of aggregate and per-flow treatments in the 12 routers contended by data flows in other sets. For each data flow in group 2, in the routers where it is treated as per flow, it receives better service than that in DiffServ. In the routers where it is treated as aggregate traffic, it receives worse service than that in DiffServ. Overall, data flows in group 2(Figure 10.c, Average Delay 0.0062 second) have similar average delay as in DiffServ (Figure 10.a, Average Delay 0.0084 second).

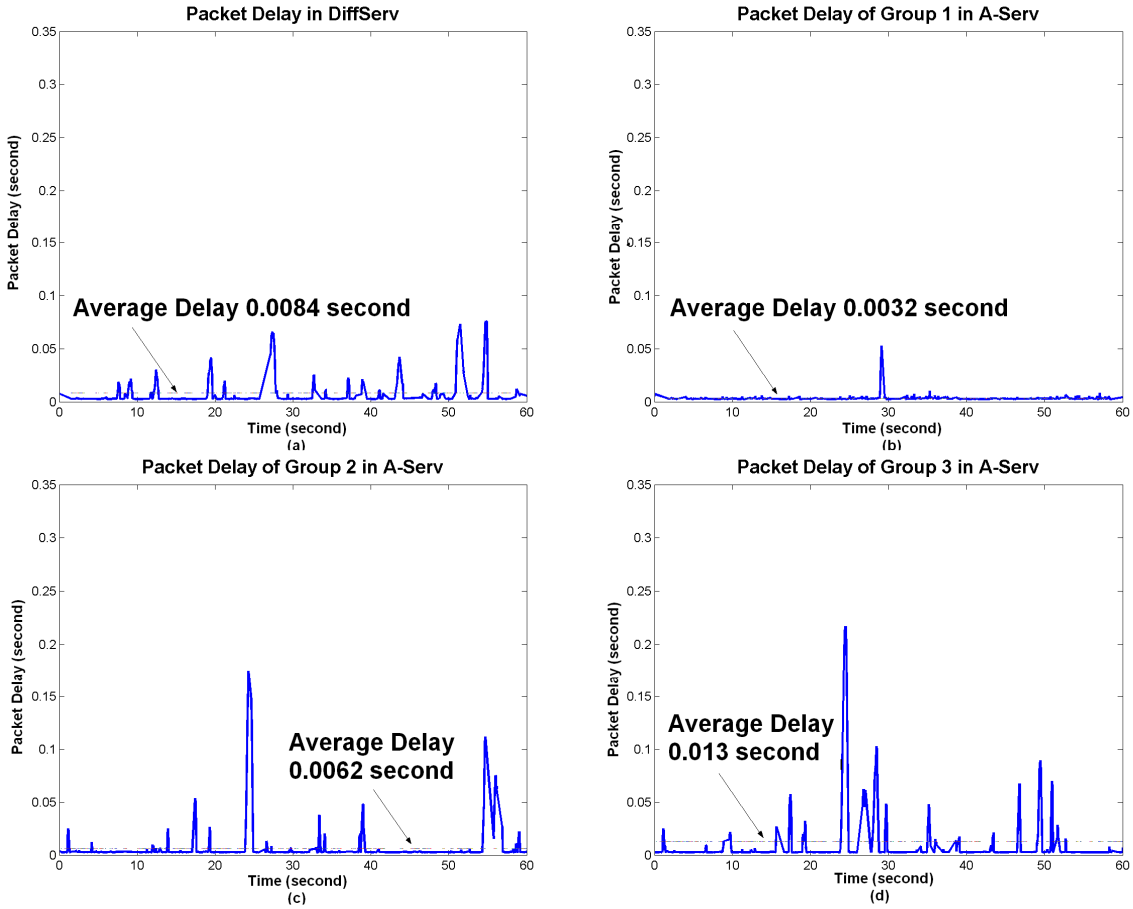

FIGURE 10: Delay Results for Non-malicious Data Flow Case

4.2 Scenarios With Malicious Data Flow

In the scenario with a malicious data flow, we set up one malicious data flow from ER1 to ER2 in addition to the 64 non-malicious data flows. We assume the malicious data flow is admitted in the network in the same way as other non-malicious data flows, which means it is expected to behave in the same way as normal data flows in set A and B. The actual behavior of this malicious data flow follows ON-OFF Markov process with ON time=0.9 second, OFF time=0.1 second, and peak rate 600Kbps. This malicious data flow shares its route in domain 1 with data flows in both set A and B. Therefore normal data flows in both sets could be affected by its malicious behavior. We notice that the performance of data flows in set A and B is similar and we only show the performance of data flows in set B.

The malicious data flow could receive per-flow or aggregate treatments in domain 1. We first consider the case when the malicious data flow is treated as per flow at all routers on its path. The non-malicious data flows receive the same treatment combinations as in Table 1. The end-to-end delays are plotted in Figure 11 for all three groups of data flows between ER1 and ER5 under A-Serv and the delays in DiffServ.

From Figure 11, we can see that when the malicious data flow is treated as per flow, its influence to all other data flows is reduced and all groups' performance (Figure 11.b-d, Average Delay 0.0039/0.0089/0.016 second) is better than that in DiffServ (Figure 11.a, Average Delay 0.058 second). Such performance improvement comes from the isolation of the malicious data flow in a per flow queue, where its malicious behavior will mainly punish its own performance but not

affecting other non-malicious data flows.

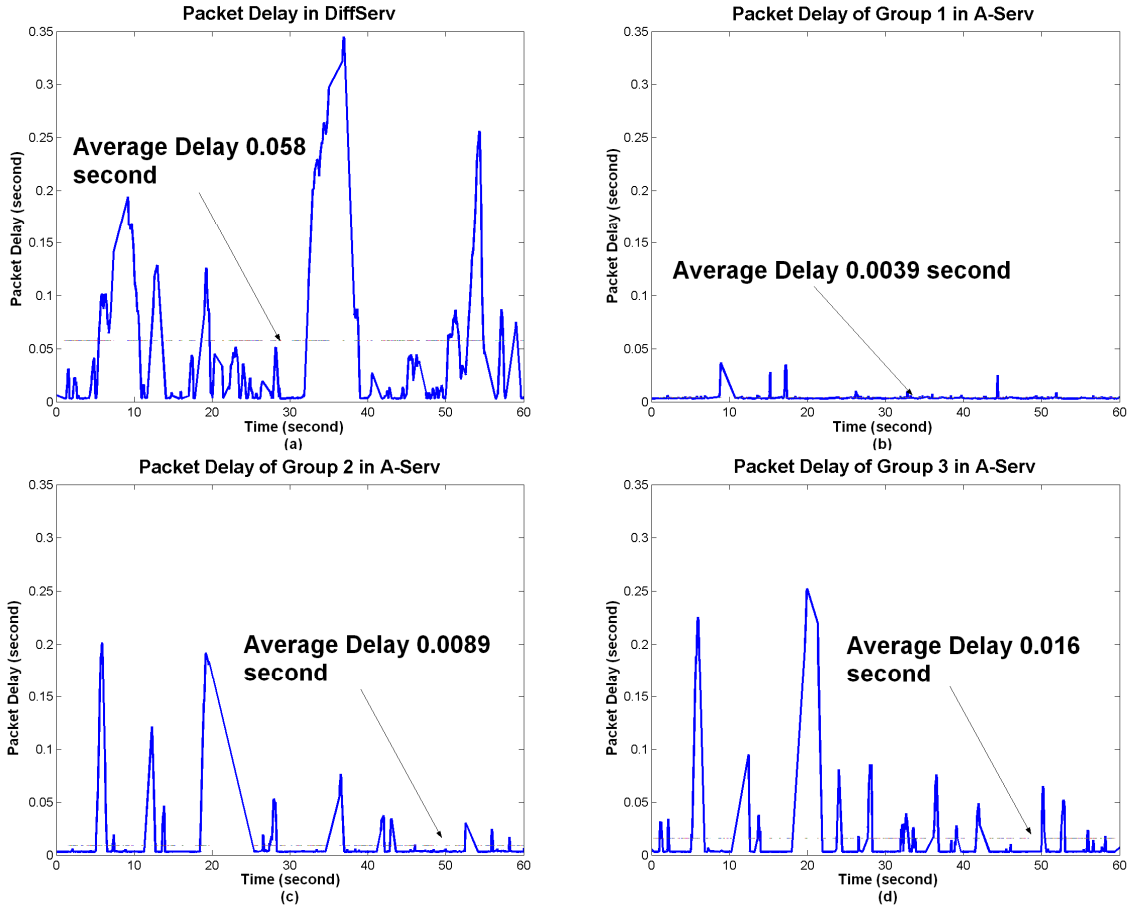

FIGURE 11: Results with a Malicious Data Flow Treated as Per Flow

The performance of group 2 (Figure 11.c Average Delay 0.0089 second) and group 3 (Figure 11.d Average Delay 0.016 second) is worse than their cases in Figure 10c (Average Delay 0.0062 second) and Figure 10d (Average Delay 0.013 second) respectively. This is because the per-flow treated data flows could not make full usage of the reserved bandwidth due to the traffic's burstiness. Such leftover bandwidth is utilized by the aggregated traffic in the same router, which actually improves the data flows' performance in group 2 and 3. However, when malicious data flow exists, the malicious data flow demands more bandwidth than the amount it reserved in its per-flow queue. Therefore, the malicious data flow competes on the leftover bandwidth with the aggregate traffic and degrades performance of data flows in group 2 and 3.

Now we consider the case when the malicious data flow is treated as aggregate traffic between ER1 and ER2. This setting complicates the treatment combinations of data flows between ER1 and ER5. The data flows in group 1 and 3 are the same as in Table 1. The data flows in group 2 are further divided into group 2a and group 2b. Data flows in group 2a are treated as aggregate traffic on the links shared with the malicious data flow and are treated as per flow in the rest routers (4 data flows in the second row of Table 1). The data flows in group 2b are treated as per flow on the links shared with the malicious data flow and treated as aggregate traffic in most of the rest routers (3 data flows in the third row of Table 1). The performance results are shown in Figure 12.

Compared with DiffServ in Figure 11a, the data flows that are treated as per flow on the links shared with the malicious data flow (group 1 in Figure 12a with Average Delay 0.0037 second

and group 2b in Figure 12c with Average Delay 0.0069 second) have better performance than in DiffServ (Average Delay 0.058 second in Figure 11a). These data flows are isolated in per-flow queues in the routers where the malicious data flow exists. Such isolation guarantees the reserved service received by these data flows without interference from the malicious data flow. The data flows that are aggregated with malicious flow (group 2a in Figure 12b with Average Delay 0.053 second and group 3 in Figure 12d with Average Delay 0.084 second) receive degraded service and have similar or even worse performance than that in DiffServ (Average Delay 0.058 second in Figure 11a) because they are aggregated with the malicious data flow and affected by its malicious behavior.

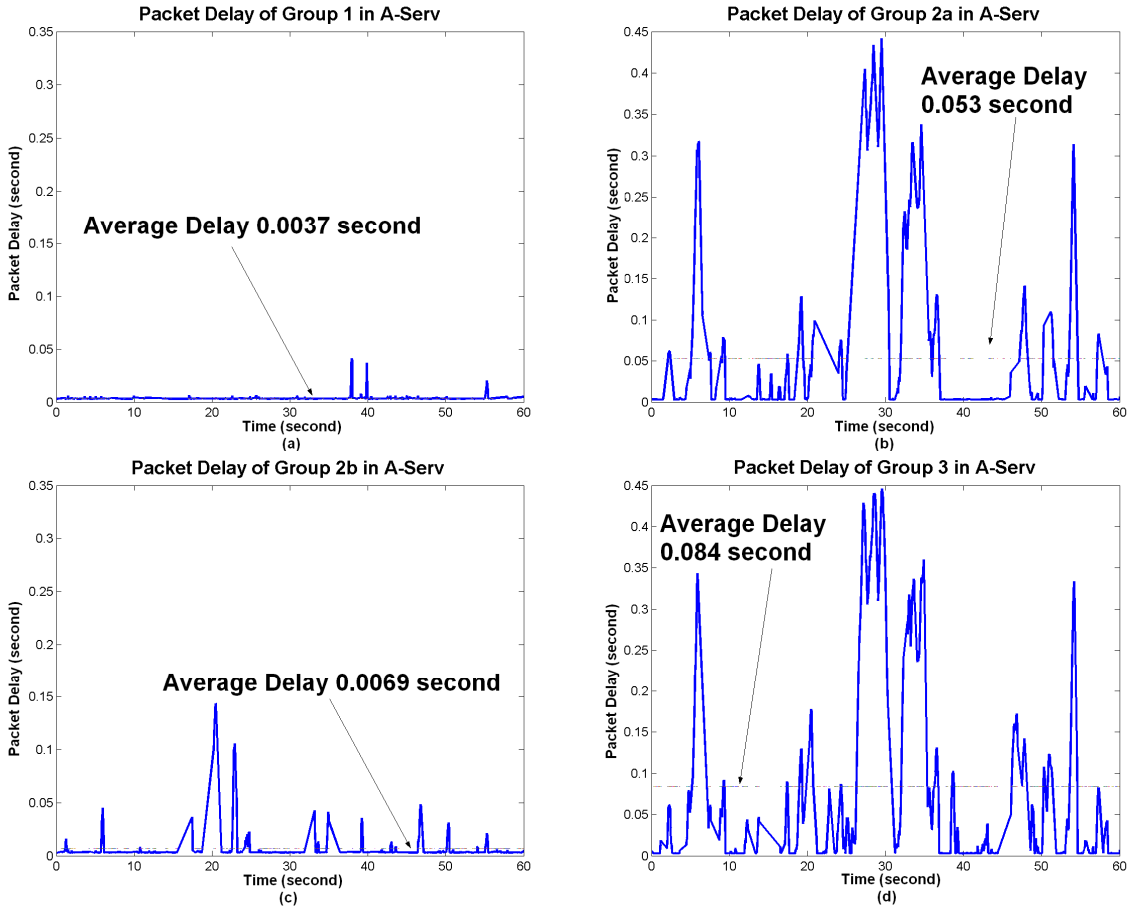

FIGURE 12: Results with a Malicious Data Flow Treated as Aggregate Traffic

5. CONCLUSION AND FURTHER WORK

We introduce a new QoS architecture called A-Serv using load adaptive router to provide Adaptive Services. In the new QoS architecture, we propose to decouple the connection between packet structure and router functions. A-Serv architecture provides different packet treatments to data flows based on router's load burden. A-Serv overcomes the scalability problem of IntServ, provides better QoS guarantee than DiffServ and can be deployed incrementally. The analysis of data flows' performance in A-Serv under both with/without malicious data flow scenarios shows that A-Serv can provide better QoS guarantees than DiffServ without loosing scalability. The differentiated services provided to data flows in the same class in one router offer more options in providing variant and/or prioritized service to end users while maintaining the scalability. Furthermore, the adaptive property of A-Serv simplifies system upgrades. With the development in the router's processing ability, simply changing the per-flow number upperbound enables more data flows to receive guaranteed service without changing network protocols or packet header format.

Our design of A-Serv provides a new adaptive sight in QoS architecture design to deal with the tradeoff between scalability and QoS guarantee. In A-Serv, we constructed a general framework for such design and there are many detailed aspects and options need to be addressed in the future. One research issue is designing priority and scheduling scheme in A-Serv, which extends the load adaptive router's function to support more sophisticated data flow treatment schemes. Another further topic is how to design admission control algorithm to deal with the data flows receiving different services at different core routers. Different options of load burden criteria also need to be explored.

6. REFERENCES

[1] B. Braden, "Integrated services in Internet architecture - an overview," *Internet RFC-1633*, June 1994.

[2] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An architecture for differential services," *IETF, RFC 2475*, December 1998,

[3] I. Stoica and H. Zhang, "Providing guaranteed services without per flow management," *Proc. Of ACM SIGCOMM,* pp. 81-94, September 1999.

[4] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource ReSerVation protocol (RSVP) – version 1 functional specification," *RFC 2205,* Internet Engineering Task Force, September 1997.

[5] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP: a new resource ReSerVation protocol," *IEEE Network*, September 1993.

[6] S. Bradner and A. Mankin, "The recommendation for the IP next generation protocol," *RFC 1752*, Internet Engineering Task Force, June 1995.

[7] V. Jacobson, K. Nichols, and K. Poduri, "An expedited forwarding PHB," *RFC 2598,* Internet Engineering Task Force, June 1999.

[8] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski, "Assured forwarding PHB group," *RFC 2597*, Internet Engineering task Force, June 1999.

[9] J.C.R. Bennett, K. Benson, A. Charny, W.F. Courtney, and J.Y. Le Boudec, "Delay jitter bounds and packet scale rate guarantee for expedited forwarding," *IEEE/ACM Transactions on Networking*, vol. 10, Issue 4, pp. 529-540, August 2002.

[10] A. Charny and J.Y. Le Boudec, "Delay bounds in a network with aggregate scheduling," *Proc. Of First International Workshop of QofIS'2000*, Berlin, Germany, September, 2000.

[11] V. Firoiu, J. -Y. Le Boudec, D. Towsley, Zhi-Li Zhang, "Theories and models for Internet quality of service," *Proc. Of the IEEE,* vol. 90, Issue 9, pp. 1565-1591, September 2002.

[12] Network Simulator 2 - NS2, http://www-mash.cs.berkeley.edu/ns.

[13] "The need for QoS," Stardust.com. White Paper, July 1999, http://www.qosforum.com

[14] D. Ferrari and D. C. Verma, "A scheme for real-time channel establishment in wide-area networks*," IEEE Journal on Selected Areas in Communications,* vol. 8, Issue 3, pp. 368-379, April 1990.

[15] E. W. Knightly, "H-BIND: a new approach to providing statistical performance guarantees to VBR traffic," *Proc. Of IEEE INFOCOM '96,* pp. 1091--1099, March 1996.

[16] E. W. Knightly and N. B. Shroff, "Admission control for statistical QoS: theory and practice," *IEEE Network,* vol. 13, Issue 2, pp. 20--29, 1999.

[17] C. Courcoubetis and R. Weber, "Effective bandwidth for stationary sources," *Prob. Eng. Inf. Sci.,* vol. 9, 285-294, 1995.

[18] R. Guerin, H. Ahmadi, and M. Naghshineh, "Equivalent capacity and its application to bandwidth allocation in high-speed networks," *IEEE Journal on Selected Areas in Communications,* vol. 9, Issue 7, September 1991.

[19] G. Kesidis, J. Walrand and C. Chang, "Effective bandwidths for multiclass Markov fluids and other ATM sources," *IEEE/ACM Transactions on Networking,* vol. 1, pp. 424-428, 1993.

[20] F. P. Kelly, "Effective bandwidths at multi-class queues," *Queueing Systems,* vol. 9, pp. 5-16, 1991.

[21] E. W. Knightly and H. Zhang, "D-BIND: an accurate traffic model for providing QoS guarantees to VBR traffic," *IEEE/ACM Transactions on Networking,* vol. 5, Issue 2, pp. 219-231, April 1997.

[22] G. Mao, and D. Habibi, "Loss performance analysis for heterogeneous on-off sources with application to connection admission control," *IEEE/ACM Transactions on Networking*, vol. 10, Issue 1, pp. 125-138, February 2002.

[23] B. Pang, H. Shao, W, Zhu, and W. Gao, "An admission control scheme to provide end-to-end statistical QoS provision in IP networks," *21st IEEE International Performance, Computing, and Communications Conference,* pp. 399-403, April 2002.

[24] F. P. Kelly, P. B. Key, and S. Zachary, "Distributed admission control," *IEEE Journal on Selected Areas in Communications,* vol. 18, Issue 12, pp. 2617-2628, December 2000.

[25] L. Breslau, E. Knightly, S. Shenker, I. Stoica, and H. Zhang, "Endpoint admission control: architectural issues and performance," *ACM SIGCOMM Computer Communication Review*, vol. 30, Issue 4, 2000.

[26] G. Bianchi, F. Borgonovo, A. Capone, L. Fratta, and C. Petrioli, "Endpoint admission control with delay variation measurements for QoS in IP networks," *ACM SIGCOMM Computer Communication Review,* vol. 32, Issue 2, April 2002.

[27] K. S. Seo, and B. G. Lee, "Measurement-based admission control using maximum burstiness," *IEEE Communications Letters,* vol. 6, Issue 9, pp. 403-405, September 2002.

[28] J. Qiu and E. W. Knightly, "Measurement-based admission control with aggregate traffic envelopes," *IEEE/ACM Transactions on Networking*, vol. 9, no. 2, pp. 199-210, April 2001.

[29] L. Breslau, E. W. Knightly, S. Shenker, I. Stoica, and H. Zhang, "Endpoint admission control: architectural issues and performance," *Proc. Of ACM SIGCOMM'00*, pp. 57-69, September 2000.

[30] R. Gibbens and F. Kelly, "Distributed connection acceptance control for a connectionless network," *Proc. Of ITC'99*, June 1999.

[31] V. Elek, G. Karlsson, and R. Ronngre, "Admission control based on end-to-end measurements," *Proc. Of IEEE INFOCOM*, March 2000.

[32] T. Chen, J. Walrand, and D. Messerschmitt, "Dynamic priority protocols for packet voice," *IEEE Journal on Selected Areas in Communications*, vol. 7, Issue 5, pp. 632-643, 1989.

[33] D. Ferrari and D. Verma, "A scheme for real-time channel establishment in wide-area networks," *IEEE Journal on Selected Areas in Communications*, vol. 8, Issue 3, pp. 368-379, April 1990.

[34] D. Verma, H. Zhang, and D. Ferrari, "Guaranteeing delay jitter bounds in packet switching networks," *Proc. Of TRICOMM*, pp. 35-46, New York, April 1991.

[35] D. Clark, S. Shenker, and L. Zhang, "Supporting real-time applications in an integrated services packet network: Architecture and mechanism," *Proc. Of ACM SIGCOMM '92*, pp. 14-26, Baltimore, August 1992.

[36] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: the single node case," *IEEE/ACM Transactions on Networking*, vol. 1 Issue 3, pp. 344-357, June 1993.

[37] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," *Proc. Of ACM SIGCOMM*, pp. 3-12, 1989.

[38] M. Andrews, "Probabilistic end-to-end delay bounds for earliest deadline first scheduling," *Proc. IEEE INFOCOM 2000*, Tel AVIV, Israel, pp. 603-612, March 2000.

[39] M. Andrews and L. Zhang, "Minimizing end-to-end delay in high-speed networks with a simple coordinated schedule," *Proc. Of IEEE INFOCOM 1999*, pp. 380-388, New York, March 1999.

[40] S. J. Golestani, "A self-clocked fair queueing scheme for broadband applications," *Proc. Of IEEE INFOCOM 1994*, vol. 2, pp. 636-646, June 1994.

[41] L. Georgiadis, R. Guerin, V. Peris, and K. Sivarajan, "Efficient network QoS provisioning based on per node traffic shaping," *IEEE/ACM Transactions on Networking*, vol. 4, Issue 4, pp. 1518-1535, August 1996.

[42] L. Zhang, "Virtual clock: A new traffic control algorithm for packet switching networks," *ACM Transactions on Computer Systems*, vol. 9, Issue 2, pp. 101-124, May 1991.

[43] S. J. Golestani, "Congestion-free communication in high-speed packet networks," *IEEE Transactions on Communications*, vol. 39, Issue: 12, pp. 1802-1813, December 1991.

[44] H. Sariowan, R. L. Cruz, G. C. Polyzos, "SCED: A generalized scheduling policy for guaranteeing quality of service," *IEEE/ACM Transactions on Networking*, vol. 7, Issue 5, pp. 669-684, October 1999.

[45] D. Saha, S. Mukherjee, and S. K. Tripathi, "Carry-over round robin: A simple cell scheduling mechanism for ATM networks," *IEEE/ACM Transactions on Networking*, vol. 6, Issue 6, pp. 779-796, December 1998.

[46] M. Shreedhar and G. Varghese, "Efficient fair queuing using deficit round-robin," *IEEE/ACM Transactions on Networking*, vol. 4, Issue 3, pp. 375 - 385, June 1996.

[47] H. Zhang, D. Ferrari, "Rate-controlled static-priority queueing," *Proc. Of IEEE INFOCOM 1993*, vol. 1, pp. 227-236, March 1993.

[48] E. P. Rathgeb, "Modeling and performance comparison of policing mechanisms for ATM networks," *IEEE Journal on Selected Areas in Communications,* vol. 9, Issue 3, pp. 325-334, April 1991.

[49] J. Turner, "New directions in communications (or which way to the information age)," *IEEE Communication Magazine,* vol. 24, Issue 10, pp. 8-15, October 1986.

[50] M. Salamah, H. Lababidi, "BLLB: a novel traffic policing mechanism for ATM networks," *Proc. Of 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems,* pp. 411-415, August 2000.

[51] M. Salamah, H. Lababidi, "FBLLB: a fuzzy-based traffic policing mechanism for ATM networks," *ACS/IEEE International Conference on Computer Systems and Applications*, pp. 31-35, June 2001.

[52] Z. Wang and J. Crowcroft, "QoS routing for supporting resource reservation," *IEEE Journal of Selected Areas Communication,* vol. 14, pp. 1228-1234, September 1996.

[53] H. De Neve and P. Van Mieghem, "TAMCRA: a tunble accuracy multiple constraints routing algorithm," *Computer Communication*, vol. 24, pp. 667-679, 2000.

[54] W. Xiao, Y. Luo, B. H. Soong, et al. "An efficient heuristic algorithm for multi-constrained path problems," in *Proc. Of VTC2002-Fall,* Vancouver, British Columbia, Canada, September 2002.

[55] G. Liu and K. G. Ramakrishnan, "A*Prune: an algorithm for finding K sharotest paths subject to multiple constraints," in *Proc. IEEE INFOCOM,* vol. 2, 2001, pp. 743-749.

[56] S. Chen and K. Nahrstedt, "On finding multi-constrained paths," in *Proc. Of IEEE ICC98*, Atlanta, USA, Vol.2, pp. 874-879, June 1998.

[57] X. Yuan, "Heuristic algorithms for multiconstrained quality-of-service routing," *IEEE/ACM Transactions on Networking,* vol. 10, pp. 244-256, April 2002.

[58] L. Guo and I. Matta, "Search space reduction in QoS routing," in *Proc. 19th International Conference of Distributed Computing Systems, III,* pp. 142-149, May 1999.

[59] R. Hassin, "Approximation schemes for the restricted shortest path problem," *Math. Operation Research,* vol. 17, no. 1, pp. 36-42, 1992.

[60] A. Juttner, B. Szviatovszki, I. Mecs, and Z. Rajko, "Lagrange relaxation based method for the Qos routing problem," in *Proc. Of IEEE INFOCOM,* vol. 2, pp. 859-868, April 2001.

[61] D. S. Reeves and H. F. Salama, "A distributed algorithm for delay-constrained unicast routing," *IEEE/ACM Tansactions of Networking,* vol. 8, pp. 239-250, April 2000.

[62] A. Orda, "Routing with end-to-end QoS guarantees in broadband networks," *IEEE/ACM Transactions of Networking*, vol. 7, pp. 365-374, June 1999.

[63] C. Dovrolis and P. Ramanathan, "Proportional Differentiated Services, Part II: Loss Rate Differentiation and Packet Dropping," *Proc. IWQoS, 2000,* pp. 52-61, 2000.

[64] T. Quynh et al., "Relative Jitter Packet Scheduling for Differentiated Services," *Proc. 9th IFIP Conf. Perf. Modeling and Eval. of ATM & IP Networks*, 2001.

[65] [QA-00-14] S. Bodamer, "A New Scheduling Mechanism to Provide Relative Differentiation for Real-Time IP Traffic," *Proc. GLOBECOM, 2000,* vol. 1,pp. 646-650, 2000.

[66] C. Dovrolis, D. Stiliadis, and P. Ramanathan, "Proportional Differentiated Services: Delay Differentiation and Packet Scheduling," *ACM SIGCOMM,* September 1999.

[67] C. L. Lee, J. R. Chen, and Y. C. Chen, "A scalable architecture for differentiated services," *Proc. Of 22nd International Conference on Distributed Computing Systems Workshops, 2002*, pp. 311-316, 2-5 July 2002.

[68] J. C. R. Bennett and H. Zhang, "Hierarchical packet fair queueing algorithms," *IEEE/ACM Transactions on Networking*, vol. 5, Issue 5, pp.675-689, October, 1997.