Achieving Quality of Service through SCalable Aggregate Reservations

Daniel Reid[1]
Michael Katchabaw[2]
The University of Western Ontario
Department of Computer Science
London, Ontario, Canada N6A 5B7
[1]dreid28@csd.uwo.ca
[2]katchab@csd.uwo.ca

Abstract — Previous attempts at providing widespread Quality of Service (QoS) for the Internet have met with only limited success. Integrated Services and Differentiated Services, the two most popular architectures proposed, suffer from scalability and flexibility concerns respectively. More recently, numerous other architectural proposals have been introduced, but with limited success. This paper introduces a new approach which addresses several issues others have failed to solve effectively. This approach, SCalable Aggregate Reservations (SCAR) is highly scalable, offers additional functionality, and is quite flexible and robust in supporting QoS for networked applications.

Keywords*: QoS, aggregate reservations, stateless networking*

Received January 27, 2006 / Accepted August 03, 2006

1. Introduction

Originally designed as a best-effort service, the Internet has grown significantly, and so have its needs. Today, many applications require a service level better than best-effort can offer. In effect, these applications require a high level of Quality of Service (QoS), which is a term used to describe the overall experience a user or application receives over a network [20]. Over the Internet, this is typically measured in terms of bandwidth, loss, delay, jitter, and availability.

Since the Internet does not intrinsically support any preferential treatment of traffic, there can be significant problems with new QoS sensitive applications, such as teleconferencing and IP telephony. Currently, most networks are simply pushing for greater network capacity to alleviate congestion problems. While this option is currently cheaper than deploying a widespread QoS infrastructure, it does not allow prioritization of flows or guarantees of service. Simply "throwing more bandwidth" at the problem is merely a short term fix, and will not address any long term issues [17].

QoS models strive to take a best-effort network and transform it into one which can provide bandwidth and delay assurances to its applications. There have been several fields of thought on providing QoS. By far, the two most popular and accepted philosophies are the Integrated Services Model (IntServ) [7] and Differentiated Services Model (DiffServ) [6]. However, neither IntServ nor DiffServ have gained widespread acceptance due to scalability and flexibility concerns respectively. Several lightweight protocols have emerged, however they lack the functionality and acceptance for a truly integrated services network.

This paper presents a stateless QoS architecture for the Internet that can provide end-to-end QoS guarantees to multiple flows on demand. The architecture, called SCalable Aggregate Reservations (SCAR), achieves scalability by aggregating flows into predefined classes. It requires little processing at intermediate nodes and can additionally enable billing functions, security functions, and mechanisms to assist the optimization of resource allocations [23]. SCAR belongs to a variant of the IntServ philosophy; one that aims to provide the same flexible service, but in a stateless network environment. This stateless property is one that addresses scalability concerns that have arisen with traditional IntServ. SCAR combines the scalability of DiffServ and the functionality of IntServ into one flexible package.

A new signaling protocol, the SCAR Signaling Protocol (SCAR-SP), was developed to enable signaling within this new architecture. Its reversible design allows either the sender or receiver to make reservations. It presents many additional advantages, including the ability for senders to specify their unique traffic characteristics, and a soft-state approach which can remove reservations if not properly terminated. Experimentation has shown that nodes can handle hundreds of thousands of flows simultaneously with little impact on local node performance.

Over the past decade, the focus of Internet QoS has been moving further and further away from complex stateful solutions; this paper will reinforce this shift by introducing a simple architecture and signaling protocol which provides the most stringent QoS needs to sensitive applications. The remainder of this paper is as follows. Section 2 provides a discussion of related work in this area. Section 3 presents the architectural design of the SCAR model, the signaling protocol SCAR-SP, and scheduling in SCAR. Section 4 discusses experimental results and comparative analyses with other approaches. Section 5 concludes this paper with a summary and discussion of future work.

2. Related Work

As mentioned earlier, the two most widely recognized models to Internet QoS are IntServ and DiffServ. The amount

of router state required for the IntServ model, particularly in core routers, increases with the number of flows. Since router performance is linked with its ability to maintain these flows, and the Internet is still growing at a phenomenal rate, this model has serious scalability and complexity concerns. Since DiffServ treats packets in the same class identically, it is difficult to provide quantitative service to individual flows. The model is strong on simplicity, but weak on guarantees. The drawbacks of both models have resulted in numerous attempts to solve the issues plaguing this research area.

2.1. Architectures

The Scalable Resource Reservation Protocol (SRP) [2] attempts to remove the need for both signaling protocols and state from the routers. Sources mark data sent at an initial rate, intermediate nodes remark the data based on an acceptable rate that can be supported, and the destination reports this rate back to the source. The source then throttles back the rate, and only sends at the observed peak rate. While fairly simple, this architecture relies on the cooperation and trust of end hosts, and cannot provide delay bounds.

Stateless Core (SCORE) [24] was designed to move state from the routers into the packets themselves using a technique called Dynamic Packet State (DPS). It can support both DiffServ and IntServ models, and unlike other proposals can provide guarantees on delay. While one of the better solutions proposed in recent years, it has yet to gain attention.

More recently, Simplified Guaranteed Service (SGS) [21] has emerged. It requires no per-flow state to be maintained in core routers, and defines a new signaling protocol called Sender Oriented Signaling (SOS). It is based on aggregate reservations, and can provide bounds on bandwidth. This scheme has yet to be tested extensively, and uses burdensome garbage collection.

Other approaches proposed in the literature include the Flow Initiation and Reservation Tree Protocol (FIRST) [11], IntServ over DiffServ [4], Edge Assisted Quality of Service (EQOS) [5], Endpoint Admission Control (EAC) [9], and Aggregate RSVP [3]. Unfortunately, these approaches have various problems limiting their effectiveness, including resource management and deployment issues, additional complexity, or required mass participation and cooperation that is simply not feasible in practice.

2.2. Signaling Protocols

A major component of most QoS architectures providing integrated services is the signaling protocol. The Resource Reservation Protocol (RSVP) [14] is currently the only IETF standardized resource reservation signaling protocol. While RSVP provides many useful functions, it comes at the cost of complexity and scalability.

Several lightweight protocols have been suggested to alleviate these concerns. These include Yet Another Sender Session Internet Reservation Protocol (YESSIR) [22], which runs only over RTP sessions, and the Boomerang Protocol which can only signal using ICMP. The Ticket Signaling Protocol (TSP) [13] also provides a connectionless approach resulting in high scalability. Lightweight protocols are typically intended and designed for specific purposes. They do not however provide a general solution to the problems facing QoS delivery today.

Other non-lightweight protocols such as the Dynamic Reservation Protocol [25] can provide in-line reservations which essentially integrates signaling into the data packets themselves. DRP provides added functionality, but generally speaking it has the same problems, such as overall scalability concerns and unattractive unicast delivery as RSVP.

Since 2001, the Next Steps in Signaling Working Group (NSIS) has been working on standardizing a next generation multipurpose signaling protocol [18]. There have been numerous requirements proposed so far, including increased simplicity, the ability to allow flow and signaling aggregation, and both unidirectional and bidirectional flows. There has been no specific requirement for multicast due to the enormous complexity needed for a complete solution. It should also provide either sender or receiver initiations.

3. The SCAR Approach

In this section, we present our approach, SCAR. We discuss its design and signaling protocol SCAR-SP, and conclude with a discussion of scheduler integration.

3.1. Architectural Design

SCAR has been designed to meet the following goals:

Scalability – Traditionally, the amount of router state increased proportionately to the number of individual flows with the IntServ model, leading to serious scalability concerns. SCAR aggregates individual flows into classes, requiring no per-flow state in the intermediate nodes. Additionally, techniques to insert state into the packets themselves, as in SCORE, are not required.

Simplicity – To maintain simplicity and reduce overhead, SCAR requires only minimal state in nodes to keep track of aggregate classes only, and does not support multicasting, in line with recent thoughts on the subject [10]. The need for resource "garbage collection" as in SGS or synchronization, as some other solutions require, is also unnecessary.

Robustness – Mechanisms are in place to account for reservation failure, routing changes, and message loss. The onus has been placed on the end-host to ensure messages and reservations are timed properly to account for these failures. Intermediate nodes are able to verify traffic specifications, and police those who are not adhering to their contract.

Flexibility – Less state typically means less functionality and flexibility. SCAR can provide bounds on both delay and bandwidth and enable optimal resource allocation without the need for per-flow state.

We consider in this paper a three class architecture including the guaranteed service class which can provide the most stringent of QoS guarantees, the controlled-delay class for delay-insensitive applications, and the best-effort class. Other predefined classes are quite possible to incorporate into SCAR. To provide these classes within an existing best-effort network, we conceptualize a logical partition. By

separating link speed between each class, and with proper management, a guaranteed service network can be placed atop the existing best-effort network. This abstraction can be seen in Figure 1. To distinguish classes, data packets are classified within the headers through marking.

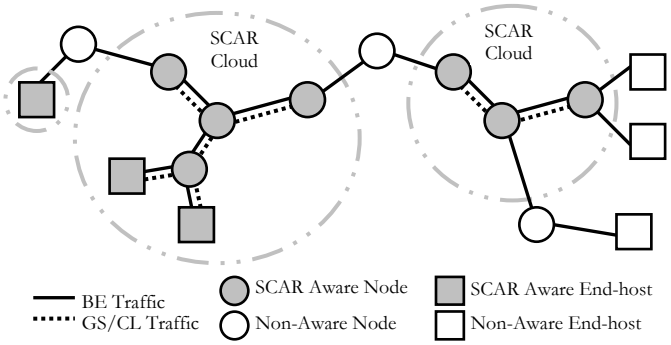

Figure 1. The logical separation of a network

In Figure 1, a logical separation can be seen between all SCAR aware nodes and end-hosts. Operation of SCAR can still take place transparently within non-aware areas of the network, although QoS cannot be guaranteed in this case when heavy congestion occurs at non-aware nodes. There are several mechanisms which need to be in place in an SCAR network. Traffic must be classified, shaped, and policed before the egress link with non-conforming packets being isolated to ensure fairness to other flows. This can potentially be done through either re-marking or dropping. Proper scheduling must additionally be in place to provide accurate bandwidth and delay bounds. SCAR does not specifically dictate the techniques used to provide guarantees in nodes, as there are already numerous approaches to do so. This will be discussed further later in this paper.

Using the additive properties of aggregation, bandwidth guarantees can be made without the need for per-flow maintenance. This is shown in Figure 2, where end-host A creates a 50KB/s reservation in (a), followed by a 20KB/s reservation by host B in (b). Aggregate state is then maintained on all incoming and outgoing links. This simplicity, coupled with proper policing, enables powerful network services.

We now begin a detailed look at elements from the SCAR architecture in Figure 3 in the remainder of this section, except for the optional security and billing modules. Details on these items can be found in [23].

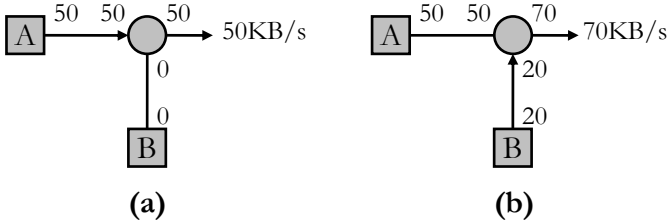

Figure 2. Additive property of aggregates

3.1.1. Marking Data Flows

End-hosts are responsible for shaping, marking, and filtering their traffic. Filterspecs [7], to determine which packets receive QoS, are created and enforced by the end hosts through marking of their own packets, thus alleviating intermediate network nodes from this burden.

A flowspec [7] describing the requested level of QoS is still transmitted to intermediate nodes. This contains an Rspec defining the desired QoS, and a Tspec describing the data flow. SCAR does not dictate which particular Tspec should be used. For example, the (σ, ρ) model [12] which describes peak and burst rates is traditionally used in RSVP, and conforms to leaky bucket parameters for policing and shaping. End-hosts wishing to additionally specify long term average rates can opt to use a dual leaky bucket traffic model instead. Furthermore, added to the flowspec when specifying a guaranteed service reservation are per-node delay values. These simply specify the largest upper bound on delay the end-host can tolerate at the node.

End-hosts are responsible for marking their own flows into the classes they belong in such a way they do not exceed their traffic specifications. The marking of flows is done by overloading the Type of Service field in the IP header. An end-host must mark two elements in this field; the *Service type* and the *Tspec type*. As mentioned above, the service type describes the level of QoS the packet will receive, while the Tspec type describes the traffic specification the packet is adhering to. While the only service types considered in this paper are guaranteed services, controlled-delay, and best-

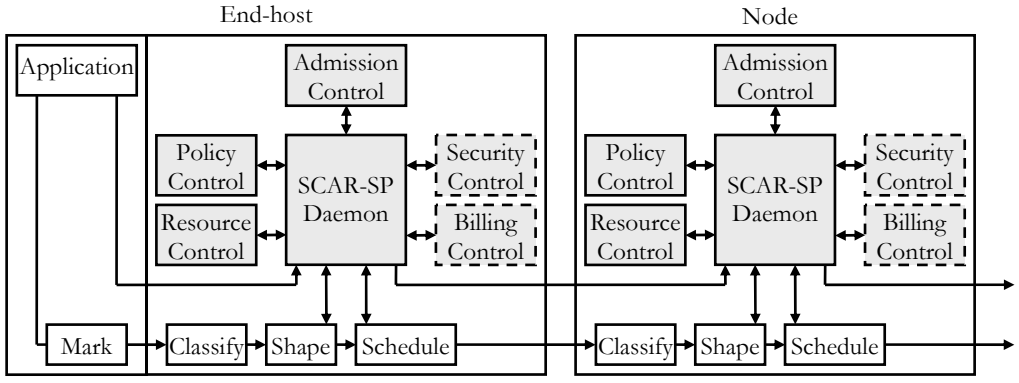

Figure 3. SCAR architecture

effort, the controlled-load class is defined here as well, with space reserved for future expansions.

Since intermediate nodes will most likely employ different policing and shaping techniques for each traffic specification, end-hosts must mark their packets with the particular traffic specification their packets belong to. If a reservation is made with one specification, and end-hosts mark their packets in another, they will be policed appropriately at intermediate nodes. Our current work supports Leaky Bucket, Dual Leaky Bucket, and other models, as discussed further in [23].

3.1.2. Shaping and Policing Misbehaving Flows

Either unintentionally or maliciously, the possibility exists that flows will not adhere to SCAR conventions. A misbehaving flow within an aggregate can not be directly policed, as no state is maintained to identify it. Instead, misbehaving aggregates are policed by isolating them from the network. Thus, the further the architectural edges are pushed towards end hosts, the greater the isolation granularity. In Figure 4, a network is shown in which the gateway, depicted as node D, and end-hosts A and B are SCAR aware. Node C located between the gateway and end-hosts is not SCAR aware. In this particular case, hosts are still able to make reservations transparently through node C, and, as long as there is no congestion at this node, QoS should not be compromised. In this case, both A and B have 50KB/s reservations to different destinations. A problem exists, however, if host A or B decides to transmit at a higher rate than reserved, or maliciously floods the network with packets marked as reserved. As there is no state maintenance at node D, it is therefore not aware who the perpetrator is, except that it is coming from the direction of node C. It must then police this aggregate, resulting in service degradation to both hosts A and B. If C was SCAR aware, however, it would only need to police the non-conforming transmitter, leaving the other at its reserved rate.

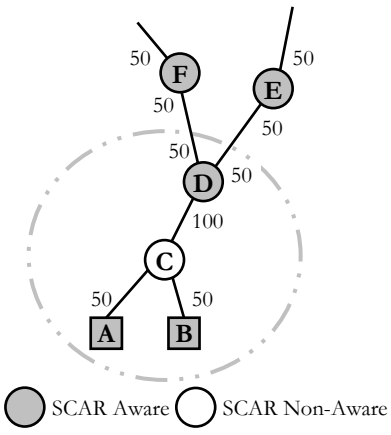

Figure 4. A network with transparent operation

Policing entails that each node will simply prevent any packets exceeding the reserved aggregate from obtaining QoS, with the onus placed on end-hosts to never exceed their reservation. Nodes may allow some tolerance to over-utilization, remark packets into a best effort class, or strict packet dropping to insure conformance. Misbehaving flows would normally be policed at the first node, and thus policing policies would typically be enforced within ones own network, or immediate upstream provider. Shaping and policing occurs on the ingress link at each node for proper isolation. For example, flows adhering to a (σ, ρ) model would be shaped and policed with a leaky token bucket.

3.1.3. Policy and Admission Control

Admission control is used to determine if enough resources are available to admit a reservation, while policy control determines if the end-host is permitted to do so. In an aggregated architecture, admission control is rather simple; only the current and maximum allowed aggregate reservation need be known. The maintenance of this aggregate, however, is far more difficult, and will be discussed further later in this paper No particular policy control is presented in this paper, as policies spanning networks are normally heterogeneous being controlled by their respective network managers.

3.1.4. Resource Allocation

Optimization of resource allocation, which is concerned with minimizing network costs when establishing QoS, has been slow to gain footing in recent years. Even resource reservation protocols such as RSVP rarely address allocation policy. It has been shown in [19] that loosening local QoS requirements at bottleneck nodes where resources are in short supply and compensating by tightening the local requirements at other nodes, in most cases, more optimized end-to-end flows will result. Generally, most current allocation policies assign equal QoS proportions at each node. Therefore, when providing some end-to-end delay guarantee *d*, each node would be assigned *d/n* delay, where *n* is equal to the number of intermediate nodes. As each node's traffic load is determined by this delay value, so is the entire path. When imbalances occur within a given network path, equally distributing QoS will naturally create bottlenecks. It has been shown that the QoS metric being served plays the most significant role in determining performance. The amount of resources, the number of hops, and position of bottlenecks are also influencing factors. An allocation policy is said to perform better than others when it can support greater network load [19].

The two metrics by which SCAR measures its guaranteed services are bandwidth and delay. SCAR aims at providing a resource allocation scheme in which informed decisions can be made at each node to provide service to the maximum number of flows. Thus, our new signaling protocol, SCAR-SP, has been designed to enable such a scheme. SCAR-SP enables a discovery procedure similar to RSVP's One Pass With Advertising (OPWA) which carries *path fingerprints* within the signaling messages that contain resource usage information along the end-to-end path.

SCAR-SP enables end-hosts to request differing delay values, but does not dictate how these decisions should be made. For example, higher delay values could be set at those

links with less bandwidth available, which has been shown in [19] to yield improvements in traffic flow. However, mechanisms must be in place to measure these gains. A straight forward and uncomplicated implementation called the Simple Allocation Scheme (SAS) is presented in the next section, which takes a similar approach. Performance measurement is conducted locally, with intermediate nodes identifying themselves as bottlenecks. We leave the investigation of more complex mechanisms for future work.

In most cases, applications are only concerned with their end-to-end performance. Thus, in different scenarios the same application will often require different performance guarantees at each intermediate node. Neglecting propagation delay, a teleconferencing application that can tolerate up to 50ms per flow would require on average 10ms of delay at each node over a short 5 hop path. This is in contrast to a 10 hop path where 5ms would be needed.

In a situation where each node can only provide one local delay guarantee, difficulties arise when multiple flows share a node. A shared node would therefore need to provide the lowest delay requirement, leading to an overall lower utilization. (If utilization was allowed to be high, it would be very difficult to maintain a lower delay.) In our architecture, end-hosts can negotiate differing delay values. In this scenario, shared core nodes would typically offer larger delay bounds, providing better utilization. This would be naturally be compensated by edge nodes where resources are more readily available, and smaller delay bounds could be offered. Further details are discussed later in this paper.

3.2. The SCAR-SP Signaling Protocol

SCAR-SP has been designed to provide efficient call mechanisms when creating, refreshing, or removing reservations. Unlike most signaling protocols, which are restricted to either a sender or receiver-oriented design, SCAR-SP enables both approaches. Either the receiver or sender can be given the responsibility to provide the flow specifications for the reservation. In many cases, receiver control is favoured over sender controlled, as senders are often not aware of what resources are available to the receiver, including bandwidth or even computational power. On the other hand, receiver control can also be a hindrance, especially in the case where senders are billed for QoS, or the sender wants more control over policy.

Another advantage of SCAR-SP's design is that senders are first responsible for characterizing the traffic it will be transmitting. By knowing the traffic model, receivers establishing a reservation can make more informed decisions. For example, if a receiver is not aware that the sender will be transmitting at a variable rate, it may unknowingly make a constant rate reservation, in effect causing underutilization. Additionally, if the sender can characterize its traffic in more than one way, the receiver can then be given the option of which specification it wishes to receive.

SCAR-SP messages require state typically not associated with signaling messages to be inserted within them. This state is minimal, and cannot be avoided in soft-state architectures. It should be reiterated that this type of state is significantly different from the state associated with IntServ scalability concerns, and is shown later in this paper to cause no significant impact on performance. The largest advantage to keeping this state in the signaling protocol is the fact that no lookup is required on behalf of intermediate nodes. SCAR's architectural specification still holds, and imposes that in its simplest form, no per-flow state shall be inserted into intermediate nodes or the data packets themselves.

SCAR-SP is a soft-state protocol which provides better dynamic adaptability and robustness than a hard-state protocol, and allows adaptation of routing changes to take place fairly seamlessly with end-host cooperation. Unlike RSVP where refresh messages are generated by intermediate nodes hop-by-hop, SCAR-SP shifts this responsibility to the end-hosts in an end-to-end fashion. They are then required to initiate periodical refreshes so that their soft-state reservations are kept active. This further reduces processing overhead at intermediate nodes.

If a reservation expires, resources are automatically de-allocated. To maintain simplicity, SCAR-SP is simplex and can only offer unidirectional reservations. To establish a bidirectional flow, two separate reservations must be made. Its design can also handle lost messages, and requires minimal processing power at the nodes.

The operation of SCAR-SP is similar to most other signaling protocols. The SCAR-SP daemon, shown earlier in Figure 3 facilitates the creation and removal of reservations in conjunction with policy and admission control modules. Admission control determines if enough resources are available to admit a reservation, while policy control determines if the end-host is permitted to do so. The classifier determines which class the traffic belongs to. Policing and shaping of the guaranteed service class ensures traffic is behaving properly, sending at proper peak and average rates. The scheduler can then guarantee the reserved bandwidth and delay to this traffic.

In an aggregated architecture, admission control is rather simple; only the current and maximum allowed aggregate reservation need be known. The maintenance of this aggregate, however, is far more difficult as signaling loss and partial reservations are introduced. Part of the difficulty arises due to the fact that admission control is based on an end-to-end transaction. That is, if all intermediate nodes along the end-to-end path do not accept the reservation, then all nodes which have since accepted it locally must roll-back to a previous state. Likewise, lost signaling messages can create confusion as partial reservations occur. Normally this would imply that nodes must maintain internal state. However, SCAR-SP can deal with these problems effectively. In the sections we below, we provide a brief overview of SCAR-SP operations and behaviour. Further details on SCAR-SP, including its message formats and handling of failures, route changes, lost messages, and other conditions can be found in [23].

3.2.1. SCAR-SP Operation

SCAR-SP defines four phases of operation; discovery, reserving, refreshing, and tearing down. The discovery phase

entails determining the current status of the end-to-end reservation path. The reservation phase involves the merging of flows in to a service class aggregate, while the refresh phase keeps these reservations active. In the teardown phase, a reservation is explicitly de-allocated. Three messages are defined; discover (DISC), reserve (RESV), and refresh (REFR). DISC messages are used in the discovery phase, while RESV messages are used during the reserving phase. REFR messages are used during the refreshing and teardown phase. For the remainder of this section, it is assumed that traffic is characterized by a simple dual leaky bucket scheme. As mentioned previously, either senders or receivers can make the reservation. As each of these methods vary slightly, the next two sections will detail each separately.

3.2.1.1. Receiver Oriented Reservations

To establish a guaranteed service receiver-oriented reservation, a DISC message is first sent from the sender to the receiver. This message is passive and is intended solely to determine the current status of all nodes participating on the reservation path. It is additionally used as a container for other information which must be reported to the receiver. As the message is traveling from the sender to receiver, each intermediate node updates the following information: the maximum end-to-end refresh interval, the maximum average rate, the maximum peak rate, the maximum peak rate duration, and the maximum burst. These are end-to-end variables which are only updated if the nodes variables are less than the current value. The receiver is therefore provided with the maximum refresh interval and flow descriptor that each intermediate node on the end-to-end path can currently provide. Intermediate nodes must also append the following per-node information: the minimum queuing delay available and the current queuing delay being served.

Thus, these values represent a variable length field in the discover message. Path state is inserted into the message as well to ensure return messages follow the same path. Policy control is also checked at each node during the discovery phase, to individualize the results for the sender-receiver pair if the policy dictates. Depending on nodal policy, some networks or hosts may be given preferential treatment in their flows in the form of greater bandwidth, or possibly be denied resources all together.

By collecting this information, the receiver can then make a well informed reservation request. Quite often, protocols not providing some form of reporting function can consequently produce many unsuccessful reservation requests, essentially flooding the network with hosts seeking reservations that cannot be made. As long as resources do not change dramatically after the discovery information has been received, and before a reservation attempt has been made, very few unsuccessful reservation requests will result.

After receiving a discover message, a RESV message is then sent back containing a flow descriptor detailing the QoS level being sought, and the path information obtained from the DISC message. The flow descriptor in this case contains all metrics described during the discovery phase; a requested refresh interval, the average rate, peak rate, peak rate duration, and burstiness of the flow. Also included is a variable sized list of sequentially ordered delay values; one for each node in the path. These delay values are chosen by the receiver such that the sum of all delay values requested is equal to or less than its end-to-end delay requirement. RESV messages require that each delay value must be greater than or equal to the corresponding minimum delay value at each node, and represent the largest delay value in which it is willing to accept. In most cases, the delay currently being provided at any given node will be less than that requested, meaning flows will often see better than expected QoS. Additionally, a node is guaranteed to never raise its delay value if it would compromise any current reservations. It should be noted that these delay values represent queuing delay at the local node, and do not include propagation delay.

Using the path state, the RESV message traverses back through the same reservation path. As it does so, admission and policy control is conducted at each node. If both are a success, the resources are allocated by updating appropriate local node aggregate variables. When a successful RESV message is received by the sender, it may begin transmitting its data at the reserved rate conforming to the traffic specifications contained in the RESV message.

Since nodes need not have their clocks synched, individual *expiration identifiers* must be appended to the RESV message at each node. This information is independent at each node, and will typically refer to the local time in which the reservation expires. These values are then used in subsequent merge messages used to refresh the resource reservation.

To preserve the reservation, REFR messages must occasionally be sent. Without such messages, a reservation will timeout and resources at each intermediate node will be de-allocated. The sender initiates this reservation refresh by sending a DISC message to the receiver. Attached to this message, and not examined by nodes, is the list of expiration identifiers obtained from the last REFR or RESV message. This list is then subsequently used by the receiver during the actual refresh. The DISC message once again appends node information and path state, and acts as a reporting feature to the receiver. A receiver can use this information if it wishes to change its resource reservation. The new REFR message issued by the receiver can contain the same flow descriptor used previously to maintain the same level or QoS, or can attempt to raise or lower reserved resources by issuing a new flow descriptor. All new REFR messages are once again subject to admission control; however it is guaranteed to pass if the descriptor has been left unchanged or lowered.

There is no explicit teardown message, as end-hosts can indirectly de-allocate their resources immediately by sending a REFR message with a refresh interval of zero. This will cause a reservation to timeout instantly.

3.2.1.2 Sender Oriented Reservations

A sender oriented reservation requires one extra message when establishing a reservation and that each message during the session is marked with an orientation flag indicating a sender oriented reservation. The sender first transmits a DISC that is used to collect path state and inform the receiver

of one or more traffic patterns it is willing to send. Upon receipt of this message, the receiver can choose which traffic pattern it wishes to accept, and generates a fresh DISC message. This message follows the reverse direction using the path state received, and contains blank traffic pattern information for intermediate nodes to append or update.

The sender, after obtaining the network information, can send its RESV message. Path state is now recorded in RESV and REFR messages, and is subsequently used in DISC messages. Similarly, expiration information is appended to RESV and REFR messages, and reported back. When collecting path information and when making reservations, the orientation flag must be checked as it indicates the incoming and outgoing links for the reservation.

3.2.2. Expiration Identifiers

When a reservation is established, a timing mechanism must be in place to de-allocate a flow's resources if a refresh is not received. Complications arise when there is no per-flow state kept in the intermediate nodes. The way in which a node wishes to implement this soft-state is not tied to one method. We present a potential mechanism in this section, by first examining how to de-allocate per-flow resources.

To maintain a scalable solution, a node should not need to maintain per-flow expiration information. Thus, we introduce a framing strategy and calendar queue that can collect this information as an aggregate. Shown in Figure 5, three reservation requests arrive on a node at local times 0.6, 2.2, and 2.8. The first requests a 6 second refresh interval, while the others only request 2 seconds. Assuming the resource requests were successful, the resources reserved in the first request are set to de-allocate in expiration slot g. Both the second and third requests will de-allocate resources in slot e. Using the additive properties of an aggregate, we can combine both de-allocations into one value. A ceiling function is used when de-allocating resources in each expiration slot. Thus, at local time 5.0, all resources within expiration slot e expire, even though resources were reserved at different times and used the same refresh interval.

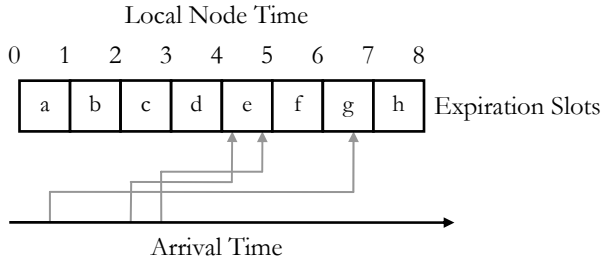

Figure 5. The use of expiration slots to de-allocate resources

End-hosts should not assume, however, any extra leniency when refreshing their resources since it is unknown to what extent intermediate nodes will potentially extend a reservation. The simplicity of such a scheme allows fine granularity in selecting the width and number of expiration slots. In networks with any form of latency, end-hosts will not even likely be aware.

The framing strategy introduced above allows resources to timeout and de-allocate effectively and efficiently. However, a second complication arises; how do we prevent such a de-allocation when a refresh is encountered and no state is maintained at the node? As previously mentioned, expiration identifiers (EIDs) are contained within the signaling messages for reference. Since node clocks are not synched, individual EIDs are needed. It should be noted that resources are allocated immediately when requesting resources, but are subject to time-framing when being refreshed.

As an example, suppose that a RESV attempt of 10KB/s is made with a refresh interval of two seconds. Aggregate resources are allocated to 10KB/s and an aggregate of -10KB/s is set at expiration. The expiration slot value is then returned as the EID for later reference. To refresh, a REFR is sent containing the same rate, same requested interval, and the EID received. In this case, a new expiration timeout is scheduled in the same manner as above. The refresh rate is then added to the expiration slot pointed to by the EID. This will prevent any pending de-allocation for this flow. By not preventing this de-allocation through adding the same refresh rate as originally specified, end-hosts can raise or lower reservation requirements when refreshing reservations.

3.2.3. Simple Optimization of Resource Allocations

As mentioned previously, SCAR-SP enables optimization of resource allocations through its discovery process. While more elaborate schemes are possible, and are the focus of continuing research, a simple scheme is presented here.

The Simple Allocation Scheme (SAS) is designed from two perspectives; the intermediate nodes, and the end-hosts. Intermediate nodes are responsible for bottleneck prevention; that is, raising or lowering local resource allocations to maximize the end-to-end flows participating on it. Since bottlenecks form when a node has exhausted its allocated resources for an aggregated class, detection is trivial. The node can then increase resources allocated by giving a greater link share, by offering larger delay bounds, or both. Appropriate upper and lower thresholds must be placed on delay and bandwidth respectively to keep new flows from obtaining undesirable QoS. Resource management control is done independently at each node, with domains controlling their allocation policies. Conversely, when there is a drop in demand, nodes can lower delay bounds or link share.

On the other side of the coin, participation of end-hosts is needed to shift resource reservations when bottlenecks form. Since intermediate nodes can not change resource allocation if it threatens any flows within the current aggregate, cooperation is needed. When adjusting delay requirements within nodes, the minimum delay field is raised or lowered appropriately. New flows are now restricted to this delay bound during admission control; however, the current delay the node is servicing cannot change until all current reservations re-adjust. To do so, nodes must wait for a period equal to its maximum refresh interval, examining refresh messages. If reservations are no longer reserving at the current delay value, it can then be adjusted appropriately to the new minimum delay value. Now, when end-hosts

examine network fingerprint information and notice that the QoS levels they are requesting are higher than currently being accepted from new flows, they must shift resource usage if possible. Resource usage from bottleneck nodes can be given up, and obtained at other non-bottleneck nodes.

3.3. Scheduling

By controlling link bandwidth and their corresponding buffers at each node, packet loss, delay, and throughput can be managed. This can be accomplished through admissions procedures allowing access to the resources, and scheduling disciplines to limit the competition of flows.

To provide node-local QoS guarantees to a flow, a packet scheduling discipline is used to guarantee bandwidth and place an upper bound on delay. The discipline may additionally provide bounds on jitter or loss. This is done by choosing which packet to transmit when its respective outgoing link becomes idle. While mechanisms within a node must be in place to address local QoS guarantees, there must also be mechanisms to ensure the end-to-end QoS is met. Due to the uncertainness of packet switching networks in a multiplexing environment, traffic patterns can become distorted through differing areas of load on the network. These distortions can result in bursts of traffic at differing points, regardless of how the traffic had entered the network, and can compromise QoS. Consequently, appropriate mechanisms must be employed to handle these situations.

The SCAR approach, as discussed earlier, is flexible and can support a wide variety of scheduling disciplines and algorithms that provide guarantees on bandwidth and delay. Since extensive work has already been dedicated to the development of approaches to scheduling network resources, this allows implementations of SCAR to leverage this existing work, and make use of an appropriate approach.

4. Experimental Results and Analysis of SCAR

In this section, we will analyze SCAR and its signaling protocol, SCAR-SP. Where possible, we also compare our approach to current RSVP implementations, and other signaling protocols and architectures discussed in Section 2, including Boomerang, SGS, and YESSIR. With no implementation available for other systems such as SCORE, we were unable to assess and compare with their performance. For brevity, we report on highlights of the most important and interesting experimental results; for complete results, the reader is urged to consult [23] for details.

4.1. Experiment Design and Setup

Unfortunately, there is a lack of consistency in proof of concept and implementation of work in this area. RSVP has multiple competing implementations available, while other approaches have implementations for only specific configurations of Linux or BSD Unix. Some of the newer approaches have been implemented for simulation packages like the Network Simulator ns-2 [15]. As a result, analyses of SCAR involved a mixture of live network tests and simulations, with baseline testing to ensure that results were reasonably consistent and comparable.

For primary testing, a 400MHz Celeron workstation with 256MB of RAM was used, as this most closely matched the configurations of test machines in related work (again to ease comparisons). As an operating system, we used the Knoppix Linux Live CD 3.6 distribution with kernel 2.4.27. By doing so, there was no need for swap space or additional overhead as the lightweight kernel is loaded directly into RAM. When traffic generation was required, a collection of Athlon XP 1800 machines with 512MB of RAM were used, with the same Linux operating system, connected by 100MB Ethernet.

For simulations, we used version 2.27 of ns-2. While there are some limitations when measuring nodal performance metrics, ns-2 is quite useful in generating very large topological simulations for study. Nodal performance was measured through software instrumentation, as ns-2 is event driven, and runs independent from processing power. The Celeron was also our primary simulation system.

SCAR and SCAR-SP were implemented and tested in this simulation environment. To implement these in ns-2, sender and receiver agents were first created to generate and handle reservation requests and responses. The sender agent was additionally responsible for generating and marking flows, and initiating periodic refreshes by sending DISC messages.

Intermediate nodes and end-hosts had a SCAR-SP daemon inserted which provided the logic required to handle incoming SCAR-SP signaling messages. Further modules were defined for policy, resource, and admission control. As our architecture does not dictate which policies to use, for simplicity we use a generic ADMIT_ALL policy which allows any end-host to reserve any arbitrary amount of the available resources. For scheduling purposes, we implemented rate-controlled servers at each node. This involved placing per-input shapers at each link. In ns-2, this was done by overloading the recv() function, and shaping the data packets before any internal switching. For simplicity, tests were restricted to leaky bucket regulated traffic.

4.2. Validation of SCAR and SCAR-SP

Before conducting performance and scalability testing, we first carried out a series of simulations to validate SCAR and SCAR-SP and ensure that they were functioning correctly. This involved the creation, maintenance, and teardown of flows, the proper scheduling of flows, and tests of flow protection, admission control, and the policing of misbehaving flows. In all cases, SCAR and SCAR-SP performed as they should and were deemed to be delivering their respective functionality correctly.

4.3. Performance and Scalability Experimentation

Many signaling protocols and architectural designs have been slow to gain footing due to scalability concerns, which is defined as the capacity for the network to expand the amount of flows, nodes, and traffic. These concerns are a direct result of the overhead required when signaling tens, if not hundreds, of thousands of flows simultaneously within

core nodes. To measure scalability, there are numerous types of overhead which are analyzed when studying signaling protocols and their respective frameworks; these include processing, memory, and network requirements.

4.3.1. Processing Overhead

Generally, the total processing overhead generated at an intermediate node can be divided into two parts; the signaling load, and session load. Signaling load is determined by the number of signaling messages arrive at the node in some time frame. Session load, on the other hand, deals with the management of existing reservations and their state.

Overhead was calculated as the percentage of time spent by the test system in processing messages and maintaining and scheduling flows. This data was collected using tools such as *top* and *tcpdump* when live tests were possible, and through instrumentation when simulations were required.

For test purposes, the reservation refresh interval was set to 30 seconds in all cases as suggested in [8], and all flows are set to live for 62.5 seconds before being regenerated, a value selected for comparison purposes with previous work in this area. While flows in the real world will most likely live for longer periods, we choose a shorter interval to address scalability concerns during periods with many short-lived reservations. In most cases, a reservation session within this interval would include a setup, two refreshes, and a teardown.

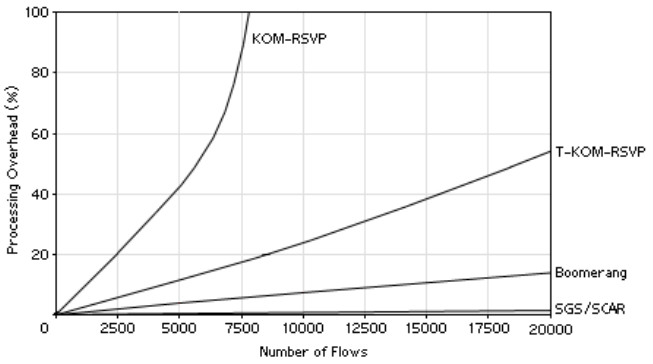

Figure 6. Processing Overhead

We summarize the overhead results in Figure 6. As can be seen, implementations of RSVP (KOM-RSVP, and the more optimized T-KOM-RSVP) had the most overhead, while SCAR and SGS producing excellent results, with SCAR just edging out SGS. Both SCAR and SGS could handle 100,000 simultaneous flows with less than a 50% processing load (with SCAR at 47.0% and SGS at 48.7%) in our test system. Boomerang performed reasonably well, but we could not observe YESSIR in this case due to issues in generating compatible traffic and flows.

4.3.2. Memory Overhead

Memory overhead at intermediate nodes is largely a function of the number of flows reserved and maintained at the node. Approaches that require more memory run the risk of exhausting resources on nodes, particularly as the number of flows increases. This ultimately places a limitation on the number of flows that can be effectively supported.

Memory requirement measurements were obtained from monitoring the */proc* virtual file system in testing live implementations, through instrumentation of simulations, and also through the examination and analysis of source code.

Memory overhead requirements for various approaches are shown in Figure 7. RSVP implementations had the highest memory overhead; the ISI RSVP implementation was found to be quite limited in the number of flows that could be supported. SCAR and SGS both had low memory needs, while Boomerang and YESSIR had moderate needs. (YESSIR results were derived from code analysis without the generation of flows.) SCAR is quite scalable from this perspective, as the memory it requires is independent of the number of flows due its stateless aggregate philosophy.

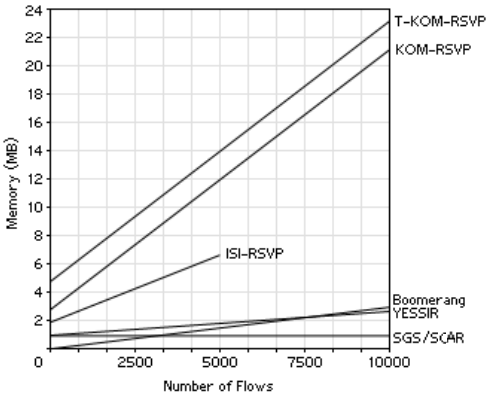

Figure 7. Memory Overhead

4.3.3. Network Overhead

Network overhead refers to the amount of additional network traffic introduced to support the creation, maintenance, and teardown of flows. Since SCAR requires state in signaling messages, it incurs a higher overhead than other approaches as shown in Figure 8.

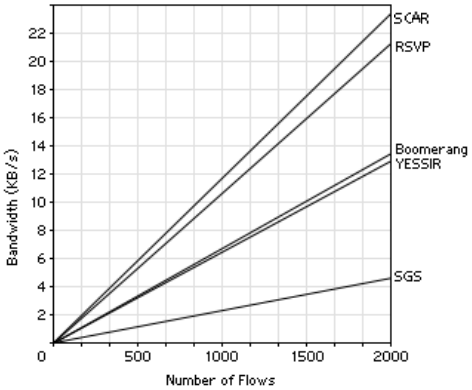

Figure 8. Network Overhead

Network overhead was computed based on an analysis of the protocol messages required to manage the number of flows required, assuming 30 seconds between refreshes and 20 hops end-to-end, shown in [1] to be an accurate average

for the Internet. It is important to note that since YESSIR is an in-band protocol, it adds little network overhead itself; the overhead reported is primarily RTCP used to carry it.

The higher network overhead for SCAR does not cause a significant concern. As shown earlier, this network overhead does not impose a significant processing or memory burden on intermediate nodes. Furthermore, the signaling overhead for SCAR is minimal in comparison to the network requirements of the flows themselves. For example, typical rates on Sprint backbones are 2488Mbps (OC-48) with the number of flows for any given minute in the area of several hundred thousand [16]. Signaling overhead for SCAR-SP on such a link serving a generous 100,000 reserved flows requires only 0.3% of its link capacity. In fact, network overhead remains a linear function, and it is generally not thought of as a problem when scaling to large numbers of flows in a call admission organization. When one considers that processing and memory considerations would prevent many approaches, including RSVP, to be completely unable to handle this number of flows in the first place, SCAR is performing reasonably well.

5. Conclusions and Future Work

This paper presented SCAR, a stateless QoS architecture for the Internet that can provide end-to-end guarantees to multiple flows on demand. It is highly scalable in its memory requirements, network requirements, and most importantly, processing requirements. It provides a greater set of functionality than previous lightweight protocols, and mechanisms to assist in optimizing resource allocations. It is not dependant on any particular scheduling algorithm, and can handle multiple traffic classifications.

A new soft-state signaling protocol, SCAR-SP, was presented to enable signaling within this new architecture. Its design allows either the sender or receiver to make reservations, and allows senders to specify their unique traffic characteristics to the network. It can additionally operate through non-aware intermediate nodes in a transparent operation. Experimentation has shown that nodes can handle hundreds of thousands of flows simultaneously with little impact on local node performance.

In the future, we plan to continue experimentation and port our ns-2 implementation for use in live testing. We are also currently investigating a variety of competing techniques to reduce the network overhead in signaling in SCAR-SP. As mentioned earlier, more work needs to be done in developing billing and security modules for SCAR, continuing the work in [23]. Additionally, optimization of resource allocations, which has rarely been addressed in the past, needs to be researched and tested more thoroughly.

References

[1] Albert R. et al. Diameter of the World Wide Web. *Nature, 401, 9*. Sept. 1999.

[2] Almesberger W. et al. SRP: A Scalable Resource Reservation Protocol for the Internet. *Computer Communications, Vol 21, Number 14*. Nov. 1998.

[3] Baker F. et al. Aggregation of RSVP for IPv4 and IPv6 Reservations. *RFC 3175*, Sept. 2001.

[4] Bernet Y. et al. A Framework for Integrated Services Operation over DiffServ Networks. *RFC 2998*, Nov. 2000.

[5] Bhatnagar S., Vickers B., Providing Quality of Service Guarantees Using Only Edge Routers. *In the Proceedings of IEEE Globecom,* San Antonio, Texas, Nov. 2001.

[6] Blake S. et al. An Architecture for Differentiated Services. *RFC 2475*, Dec. 1998.

[7] Braden R. et al. Integrated Services in the Internet Architecture: an Overview. *RFC 1633*, Jun. 1994.

[8] Braden R. et al. Resource ReSerVation Protocol (RSVP) -- Version 1 Functional Specification. *RFC 2205*, Sept. 1997.

[9] Breslau L. et al. Endpoint Admission Control: Architectural Issues and Performance. *In the Proceedings of ACM SIGCOMM 2000*, Stockholm, Sweden, Sept. 2000.

[10] Brown I. et al. Internet Multicast Tomorrow. *The Internet Protocol Journal, Volume 5*, December 2002.

[11] Chung T. et al. Flow Initiation and ReServation Tree (FIRST): A New Internet Resource Reservation Protocol. In Proc. IEEE 1999 Pacific Rim Conf. on Communications, Computers and Signal Processing, Victoria, Canada, 1999.

[12] Cruz R. A Calculus for Network Delay, Part I: Network Elements in Isolation. *IEEE Transactions on Information Theory*, Jan. 1991.

[13] Eriksson A., Gehrmann C. Robust and Secure Lightweight Resource Reservation for Unicast IP Traffic. *In the Proc. of International Workshop on QoS*. May 1998.

[14] Estrin D. et al, RSVP: A New Resource ReSerVation Protocol. *In IEEE Network.* Sept. 1993.

[15] Fall K., Varadhan K.. The ns Manual. *Reference documentation*. Mar. 2005.

[16] Fraleigh C. et al. Packet-Level Traffic Measurements from a Tier-1 IP Backbone. *Sprint ATL Technical Report TR01-ATL-110101*. Nov. 2001.

[17] Fabbi M. Bandwidth Throwing—Yesterday's LAN Design Method. *Gartner Group Research Note*. Dec. 1997.

[18] Hancock R. et al. Next Steps in Signaling: Framework. *Internet Draft*, October 2003.

[19] Nagarajan R. et al. Local Allocation of End-to-end Quality of Service in High Speed Networks. in *Proceedings of the IFIP Workshop on Performance Analysis of ATM Systems*, Jan. 1993.

[20] Nortel Networks. Introduction to Quality of Service. *Nortel Networks White Paper 56058.25_022403*. Mar. 2003.

[21] Ossipov E., Karlsson G.. Sender Oriented Signaling for a Simplified Guaranteed Service. *Proc. of Third International Workshop on Quality of Future Internet Services*, 2002.

[22] Pan P., Schulzrinne H. YESSIR: A Simple Reservation Mechanism for the Internet. *Proc. International NOSSDAV Workshop*. Cambridge, United Kingdom, July 1998.

[23] Reid D. SCAR: A Stateless Architecture for Achieving Scalable QoS. *Masters Thesis. Department of Computer Science, The University of Western Ontario*. May 2005.

[24] Stoica I. Stateless Core: A Scalable Approach for Quality of Service in the Internet. *PhD Thesis, CMU*, 2000.

[25] White P., Crowcroft J. A Dynamic Sender-Initiated Reservation Protocol for the Internet. *8th IFIP Conference on High Performance Networking*, Vienna, September 1998.