Processor Scheduling in Routers for RSVP Message Processing

Martin May† T. V. Lakshman† Anwar Elwalid*

† Bell Labs
Lucent Technologies
101 Crawfords Corner Road
Holmdel NJ 07733 USA

* Bell Labs
Lucent Technologies
600 Mountain Avenue
Murray Hill, NJ USA

Abstract

We propose CPU-scheduling schemes for processing RSVP messages in routers. We consider scenarios where there can be both bandwidth and CPU contention, i.e., the router processors are sufficiently loaded that RSVP messages may be dropped at the processors. When this happens, it is possible that link capacity is available but yet reservations do not succeed, i.e., reservation blocking is possible even if link-bandwidth is available. We show that the common first-in-first-out (FIFO) processing of RSVP messages leads to poor performance when the processor utilization is high. Link bandwidth can be wasted due to suboptimal control message processing and link utilization can oscillate considerably causing poor performance. We propose simple alternate adaptive scheduling mechansims that keep the link utilization high and hence reduce the request blocking rate. The adaptation changes the relative rates of processing so that PATH and RESERVE messages are processed at a higher rate when the current reserved link utilization is low, and the relative rate of processing of TEAR-DOWNs is increased when the link utilization is high. Apart from link-state, the adaptation also takes into account recent resource request sizes, and the relative link speeds when the same processor controls multiple links of differing speeds. We show by simulations that the proposed simple adaptive scheme has a lower request blocking rate than both FIFO and weighted round robin with non-adaptive weights.

I. Introduction

Providing differentiated services in an IP network has become an area of strong research and commercial interest. One possible approach is the one where type-of-service (TOS) bits defined in the IP packet header are interpreted in an agreed upon manner [8] so as to provide differentiated services. Another possibility is the reservation based approach which uses signaling to reserve resources for the duration of a flow or flow aggregrates. For this case, the current protocol of choice for signaling resource reservations is the Reservation Setup Protocol (RSVP) [1]. RSVP, as an example, can be used in conjunction with service models such as guaranteed rate and controlled load [9] to request quality of service for certain flows.

RSVP facilitates exchange of resource reservation information among routers in the Internet and it is a soft state protocol which relies upon periodic refreshes to maintain router state information. Refreshes that are not sent or processed within a timer expiration interval cause the established flow state to be torn down. The periodic refreshes and consequent soft state information in routers permit the protocol to operate robustly in the presence of route changes, lost signaling messages, and without always requiring explicit tear-down of state. By using soft state protocols, IP networks can offer services comparable to those in virtual circuit networks with explicit connection set-up, tear-down and connection specific routing.

However, the periodic refreshes cause the processing load in routers to increase with the number of RSVP flows that have been set-up even if these flows are not actively sending packets. There are two types of scalability issues that can potentially arise when using RSVP. One is due to the need to perform packet classification to determine which RSVP flow (if any) that an arriving packet belongs to. This has to be done for every arriving packet and can become a potential bottleneck when the number the number of established RSVP flows is large (in the thousands) and link rates are very high. We do not consider this issue in this paper. Instead, we assume that packet classification and forwarding is done (by specialized hardware or by processors) in a subsystem different from the processor processing RSVP signaling messages [6]. The second scalability issue is that caused by the control processor, which processes RSVP (and possibly other) messages, becoming bottlenecked due to high message processing load.

The RSVP load offered to the control processor consists of i) traffic due to reservation setups and teardowns and ii) traffic due to refreshes generated by RSVP-flows that have already been established. Even though refreshes require very little processing, the offered traffic due to refreshes increases with the number of RSVP flows in-progress. Also, delayed processing of refreshes can cause timer expiration in downstream routers and cause costly spurious tear-downs of flows. Hence, refreshes need to be processed at a higher priority to minimize tear-downs of flows in progress. We consider both the case where there is a control processor per interface and the case where a control processor is used for a group of interfaces (links). Even in the case where there is a control processor per interface, the processing load due to refreshes can be very high as for instance when the link speed is very high and the average request size is small (resulting in a very large number of established flows). Recent studies have shown that core Internet routers, which have to handle and maintain about 45000 flows per minute [7], have saturated CPU capacities. In this environment, simplistic RSVP message handling strategies are inadequate and can lead to poor performance.

It may be argued that the choice of processor should be dictated by appropriate traffic engineering considerations and so the processor should never be highly utilized. Even in this case, there can be temporary overloads of the control processor. This can be due to a mass call-in like scenario which generates a large number of new reservation requests or tear-downs. Link failures causing a large number of route changes can cause a temporary overload which if dealt with using FIFO scheduling may cause unnecessary tear-downs.

Also, the control processor, most likely, is not going to be processing RSVP messages alone. Instead, it may also handle route updates, multicast protocols, and perhaps slow-path forwarding of IP packets which need options processing. Recent publications have shown that routing table sizes are increasing rapidly and will probably increase processing loads for updates. Degermark, in [4], measured the time that a router CPU needs to build up the routing table as being between 50ms and 100ms. In conjunction with the observations of Vern Paxson about current routing instability, one should expect large transient loads on router CPUs. With a burst of routing instabilities, and re-calculations of OSPF routes, there can be high CPU usage strictly due to routing tasks. These are factors that may cause RSVP message processing to experience processing bottlenecks.

When the control processor becomes a bottleneck, it is possible that link capacity is available but yet reservation requests do not succeed, i.e., reservation blocking is possible even when link capacity is available. Clearly, appropriate scheduling of the control processor can be used to maximally utilize the link capacity within the constraints of the available processing capacity. In this paper, we first show that FIFO processing of messages results in several problems such as oscillating link utilization, wastage of available link bandwidth, and a higher request blocking rate. We then propose adaptive processor scheduling schemes that adjust the relative priorities with which different RSVP messages are processed based on link utilization and set-up, tear-down request sizes seen in the recent past. We evaluate the performance and efficacy of these schemes using simulations. These schemes are for the case where there is a control processor per interface. Next we consider the case where a control processor controls multiple interfaces. Here, there is the additional constraint of fairness. The processor scheduling should not result in some interfaces being highly utilized while requests on other interfaces are persistently blocked even when there is available capacity. We generalize the schemes proposed for the processor per interface case to this case of processor per group of interfaces.

The paper is organized as follows: The next section describes the basic RSVP message processing model that we use. Section 3 describes our adaptive scheduling scheme. Section 4 points out problems with existing RSVP implementations and their impact on the overall protocol performance (such as oscillating link utilization) and also has a performance study of the schemes that we propose. Concluding remarks are in Section 5.

II. RSVP Message Processing

In this section, we briefly summarize aspects of the RSVP protocol that are relevant to the study of processor scheduling issues. RSVP is a receiver-oriented resource reservation protocol. Reservations are initiated when a sender requests a resource reservation (we only consider bandwidth reservations in this paper) such as a request for a certain amount of bandwidth. This request is signaled to the network using a PATH message. The PATH message is routed along the network to its destination (or set of destinations) through a series of routers just as any other IP packet. The router establishes a flow-state for the flow (or aggregated flows) indicated by this PATH request and then the PATH message is propagated[1]. Also, it starts a timer which tears-down the flow state if no RESV or UPDATE messages are received for the flow before timer expiration.

When a PATH message reaches its destination, the receiver sends back a RESV message with a bandwidth request which is possibly different from that requested in the PATH message. When a router receives a RESV message for which there is an established flow-state, it

[1]To multiple next-hops in the case of multicasts. However, we restrict ourselves to the unicast case in this paper

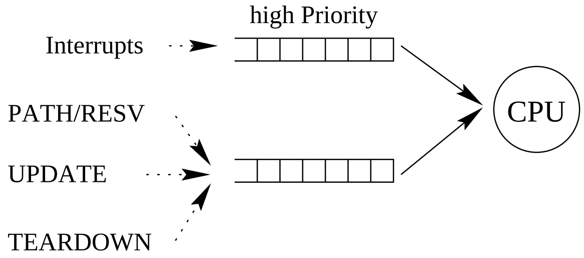


Fig. 1. FIFO scheduling for RSVP Messages with high-priority timer expiration processing

commits the requested bandwidth to the flow if sufficient bandwidth is available. Otherwise, the request is blocked. From the point-of-view of this router, the flow is now in progress. Each message receipt, resets the flow-state tear-down (refresh) timer. The flow in progress is kept alive by periodic UPDATE messages which are generated by the sender. Each router, upon processing of UPDATE messages, resets its tear-down timer and propagates the UPDATE message. A flow is torn-down and resources released, when either an explicit TEAR-DOWN message is received or when there is a refresh timer expiration.

In the absence of any agreed upon timer adaptation mechanisms, we assume that the refresh timers are set to fixed values that are at least an order of magnitude larger than typical round-trip times.

Our initial processing model is as follows: There is a control processor per high-speed link. RSVP signaling messages that arrive over the link are sent to the control processor where they are put in a queue for further processing. The simplest service discipline for the queue is FIFO with no distinctions made between the different message types. To ensure that interrupts due to timer expirations are not lost, these are processed with high priority. Therefore, as shown in Figure 1, the processor scheduling is done using priority queueing with two queues. The high priority queue processes timer expirations in a FIFO manner, and the low priority queue processes all RSVP messages in a FIFO manner. If the control CPU is not a bottleneck, this discpline is adequate. FIFO processing is not adequate when the control processor is overloaded or has high utilization and so has large processing queues.

As an example, consider the case when the link-utilization is very high. In this case, processing of PATH or RESV ahead of TEAR-DOWN messages is not fruitful since bandwidth is unlikely to be available to satisfy new requests and so requests will be blocked. Under high-link utilization, it would be beneficial to process TEAR-DOWN messages in the queue before processing PATH messages. However, with FIFO processing we cannot adopt such link-state dependent preferential behavior. Similarly, when the link utilization is low, we can defer processing of TEAR-DOWNs and instead use the bottlenecked CPU resources to process new PATH and RESV messages. Again, this is not permitted by FIFO scheduling. Furthermore, we would always like to give priority to UPDATE messages since defering UPDATE message processing can cause spurious loss of state in the same router or in downstream routers.

In Section 4, we show by simulation that FIFO scheduling results in undesirable behavior and though simple is not a good candidate discipline for scheduling the processing of RSVP messages in scenarios where the processor utilization is high. Therefore, we propose adaptive processor scheduling schemes that try to reduce blocking and increase link utilization.

III. Adaptive Processor Scheduling for RSVP Message Processing

Instead of FIFO processing of RSVP messages, an immediate scheduling change is to use weighted round-robin scheduling. The general hierarchical weighted fair queueing scheme we use is as shown in Figure 2. The rest of this section describes the scheme that we use in detail.

Suppose we classify signaling messages into just three categories: PATH and RESV, UPDATE, and TEAR-DOWN. We can then set the weights for processing each of these message types based on some a priori traffic statistics. However, there is an immediate disadvantage to using fixed weights because it is not clear what weight should be given to UPDATEs. The UPDATE traffic increases in proportion to the number of RSVP flows that have already been established. Moreover, if UPDATE messages are lost due to insufficient weight being assigned to it, then flows already set-up are unnecessarily torn down. A possibility is to give priority to UPDATES and use weighted round-robin amongst the other classes. This has the problem that if the UPDATE traffic is very high due to a very large number of flows having been set-up, then TEAR-DOWNs are not processed adequately. Hence, flows which could have been torn down may last longer and prevent new flows

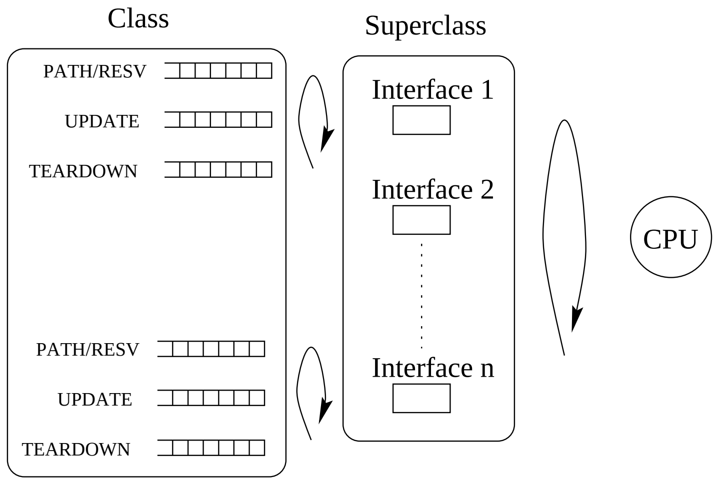


Fig. 2. Hierarchical Adaptive Weighted Round Robin Scheduling with a processor controlling multiple interfaces

from having their reservations satisified. Furthermore, a fixed weight round-robin mechanism does not take into account link utilization and knowledge of link utilization can be used to increase the probability of requests being satisfied.

The scheduling scheme we propose also uses weighted round-robin but with adaptive weights. The scheme is based on the following: (1) Updates always need a fixed fraction of the bandwidth. This fixed fraction can be made an inceasing function of the number of flows in progress but with an upper bound so that there is always a minimum bandwidth available for processing the other message types. (2) When link utilization is low, PATH and RESV messages must be given a higher weight since delaying the processing of TEAR-DOWNs does not adversely impact request blocking. Similarly, when link utilization is high, TEAR-DOWNs must be given a higher weight since processing PATH or RESV messages before TEAR-DOWNs will probably result in the bandwidth requests in those messages being denied. (3) The weights used have to be adjusted based on the average sizes of requests and tear-downs in the recent past. As an example, if reservations are small, and link utilization is low then the weight given to RESV messages have to be increased since processing of each message has a much smaller impact on the link utilization. Similarly if request sizes are large, then the weights for reservations can be scaled down. (4) Processing times for each message type, if known, should be taken into account in assigning weights. (5) Instantaneous queue-lengths by message class can be used to make weights dependent on short-term arrival rates.

Adaptation for processor per link:

For simplicity of exposition, let us consider only three classes of service. PATH and RESV messages are assigned one class, UPDATE messages are assigned to a second class, and TEAR-DOWN messages are assigned to a third class. Let the corresponding weights be denoted by w_{PR}, w_u and w_t. When a timer expires, it generates an interrupt to tear-down state. We accord high priority to these interrupts and try to process these interrupts immediately. This is because these interrupts are most likely to be generated when RSVP messages have been dropped somewhere in the network. If the processing corresponding to these interrupts is not done (such as due to them being accorded priority lower than that accorded to other tasks which can potentially saturate the processor), then the flows corresponding to these interrupts are orphaned (with no timers set to eliminate state) and they will hold on to their reserved bandwidths indefinitely causing bandwidth wastage.

The weight for updates is set to a fixed fraction of the link capacity. The only adaptation is in making this fraction proportional to the number of flows in progress.

Let PR_{ave} be the average request size expressed as a fraction of the link capacity. It is updated at every PATH or RESERVE message arrival instant in the following manner. Let *current_request_size* be the size (expressed as a fraction of the link capacity) of the bandwidth request in the message being currently processed. Then, $PR_{ave}(new)$, the updated value of the average request size is computed from the current average, $PR_{ave}(old)$, using exponential smoothing, i.e.,

$$\begin{aligned} PR_{ave}(new) &= \alpha_{PR} * PR_{ave}(old) \\ &+ (1 - \alpha_{PR}) * current\_request\_size \end{aligned}$$

We choose α_{PR} so that it tracks request sizes over time-scales of the order of a few hundred inter-arrival times. Let TD_{ave} be the average tear-down size computed, with smoothing factor α_{TD}, in a manner similar to that for PR_{ave}. Let U be the fraction of link capacity that has currently been reserved. Define $n_{PR} = (1-U)/PR_{ave}$ and $n_{TD} = U/TD_{ave}$. Then, n_{PR} denotes the number of average sized reservation requests that need to be processed to fill up the available link capacity, and n_{TD} denotes the number of average sized tear-down requests that would have to be processed to free up the utilized link capacity. (Note that the average request size and average tear-down size need not be the same since these are measured as averages over intervals typically much shorter than call holding times.)

The weight for processing PATH and RESERVEs, w_{PR}, is now chosen as $n_{PR}/(n_{PR}+n_{TD})$. Similarly, the weight for processing TEAR DOWNS, w_{TD}, is chosen as $n_{TD}/(n_{PR}+n_{TD})$. Here, there is an implicit assumption that the processing times for all the three message types are the same. Since that is not the case in practice, we scale the weights in proportion to the processing times (also these weights must be rescaled to account for the capacity allocated to UPDATEs). This basic scheme that we use has the property that the weight accorded to PATH and RESERVE message processing is increased when the request sizes are small and when the link utilization is low, i.e., $1-U$ is large. Therefore, at low utilization the system tries to increase the rate at which PATH and RESERVEs are processed since delaying TEAR-DOWNs does not affect link utilization. Similarly, when the link utilization is very high, the system processes TEAR-DOWNs at a faster rate and thereby increase the chance that the next PATH or RESERVE which is processed has a higher likelyhood of not being blocked.

One drawback of this scheme is that we do not take arrival rates into account. If we want to take arrival rates into account, an implicit way of doing this is to use average queue lengths as well in computing weights. An simple scheme is to add to n_{PR} the average queue length for this class (computed over the same time scale) and to add to n_{TD} the average queue length for TEAR DOWNs. The weights are now computed in the same manner as above. Simulation results for these cases are presented in the next section.

Processor scheduling for multiple links per processor:

The extention of the above schemes for multiple links per processor can be done by using a two-level hierarchical weighted round-robin scheme instead of a single level weighted round-robin scheme. At the higher level, there a super-class is associated with each link that the processor controls. At the lower level, for each link super-class we have the same service classes as in the processor per link scheme (see Figure 2). We first compute the weights for the lower class associated with each super-class in the same manner as described above. Let $1, 2, \ldots n$ be the links associated with a processor and let the weights for the level 1 class of link i be denoted by w^i_{PR} and w^i_{TD}. For each super-class i, we then compute the super-class weight W_i as

$$W_i = \frac{w^i_{PR} + w^i_{TD}}{\sum_{j=1}^{n} w^j_{PR} + \sum_{j=1}^{n} w^j_{TD}} \quad (1)$$

With this scheme, if any link is highly utilized (hence experiencing a high blocking) or lightly utilized (hence wasting bandwidth which could be used if the processor were not a bottleneck) then one of its lower class weights is high. If other links are not in these extreme situations, then their weights are not as high. Hence, the super-class weights tends to give a higher share of the processor to links which are at the extremes of utilization and have a backlog of messages to be processed. This enforces a certain amount of fairness amongst the links. Note that if one link has a large number of small requests while another link has a small number of large requests, then the former link gets a higher super-class weight. Also, note that link speeds can be widely differing since during weight computation, the link speeds and request sizes are normalized.

The update weight for each super-class can be obtained by calculating the update weight for each link, w^i_u, as before and then computing the UPDATE super-class weight for link i as

$$W^i_u = \frac{w^i_u}{\sum_{j=1}^{n} w^j_u} \quad (2)$$

IV. Performance Studies

The network configuration simulated is either a single router or two routers in tandem attached to a large number of sources and receivers that generate RSVP messages. The characteristics of reservations request vary in the different simulations. We assumed that the greatest number of reservation requests are small requests e.g. for audio conferences (about 64 kb/sec). But some sources generate requests for much larger bandwidth. These requests may be thought of as coming from video servers or video conferencing systems. To simulate other tasks that the processor may do, we simulate extra tasks such as routing table recalculations. The relative service times for RSVP message processing that we use in the simualtions were obtained using

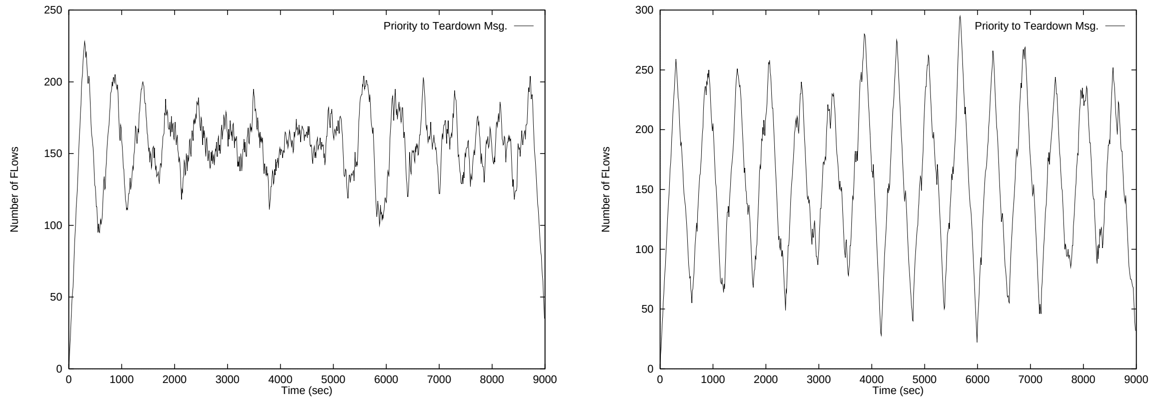


Fig. 3. Number of flows low/high CPU load without router timeouts

measurements on an existing distribution of RSVP software.

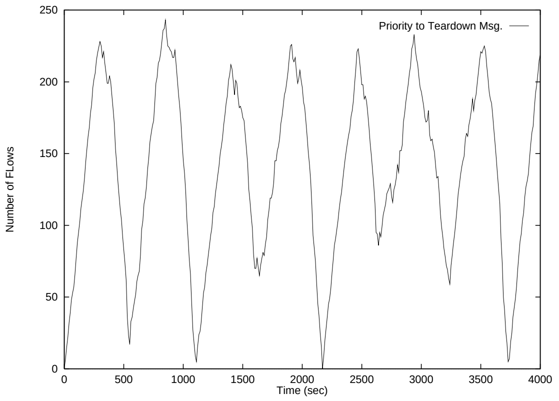
A. FIFO and Priority Scheduling

We first simulated what a priori looks like a simple and useful scheduling scheme. We use FIFO processing for PATH, RESV, TEAR-DOWN and UPDATE messages. We process state tear-downs due to timer expiration with high priority (see Figure 1). This is because if timer expirations share the same FIFO processing with other messages, some timer expirations may never get processed when the system is overloaded. This leads to orphaned flows and reserved bandwidth which is never released leading to loss of link capacity. For simplicity, we made the request size to be the same for all reservation messages (i.e., models a scenario with many flows of same type).

Figure 3 shows the link utilization (expressed in number of flows since requests are of constant size) as a function of time for two levels of processor utilization denoted by low and high. For "low" utilization, the load offered is below capacity without tear-downs. Adding the tear-down load causes processor saturation. For high utilization, the processor is always saturated. This models periods of CPU overload such as those caused by routing transients. During these overloaded periods, good scheduling should make best use of CPU resources so as to make the system behave robustly and prevent undesirable oscillations in link usage.

For the plots in Figure 3, the refresh timer is set to very large values so that it does not expire. Nevertheless, we see that the pattern of link utilization is very unsatisfcatory. Initially, when the number of flows is not very large, the processor load is low enough that many flows are succesfully established. Note that to establish a flow, both its PATH and RESV message must be succesfully processed. As the number of flows increases, the UPDATE traffic increases proportionally and hence increases processor load. Still the number of established flows keeps increasing. Next, some of the flows which have been established start generating TEAR-DOWN messages since their holding times have elapsed. Since we give priority to TEAR-DOWNs, the number of flows in progress decreases. When the number of flows has decreased sufficiently, the arrival rate of TEAR-DOWNs decreases sufficiently that new flows get established faster than the rate at which flows are being torn down. Hence, the number of flows increases again. After a delay, equal to the holding time TEAR-DOWNs are generated again and the number of flows (and hence link utilization) goes down. This oscillation is more pronounced at high offered loads. However, with the refresh timer virtually turned off, only certain choice of traffic and service time parameters produces these strong oscillations. If the refresh timer is set to a reasonable value, then oscillations happen very often as explained below.

Figure 4 shows results for a scenario similar to that in Figure 3 except that the refresh timer is not disabled. We see extreme oscillation in both cases with the number of flows in progress periodically becoming nearly zero. This can be explained as follows. As more and more flows are set up, the processor load keeps increasing since the UPDATE traffic increases in proportion to the number of flows in progress. The number of flows in progress increases at the rate of RESERVE message processing since the processing of each RESERVE message sets up a new flow unless the link saturates first and causes blocking (which does not happen in the example shown). This increasing load eventually causes the processor to saturate and the FIFO queue builds up with PATH, RESERVE and UPDATE messages waiting to be processed. Since the processor is saturated and the load increases with each processed RESERVE mes-



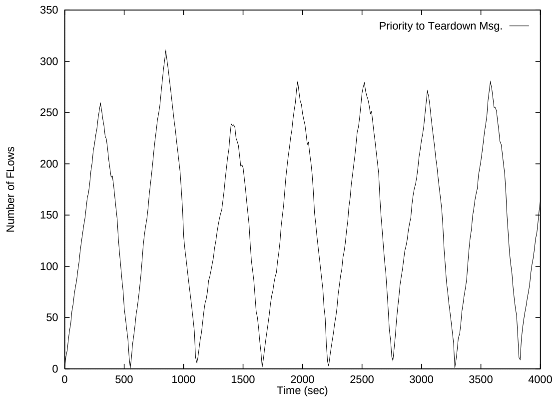


Fig. 4. Number of flows low/high CPU load and router timeouts

sage, the queue eventually overflows. Newly arriving UPDATE messages are dropped and eventually timer expirations happen for flows whose UPDATE messages have been dropped. These cause the termination of all flows for which UPDATEs have been lost causing the link utilization to continually decrease (since timer expirations which have priority are processed faster than new flow set-up). During the time that these timer exprirations are being processed, the low priority FIFO queues does not receive any service. Hence, more UPDATEs (as well as other arriving messages) are lost or delayed. This causes even more timer expirations which further starve the low priority queue of access to the CPU. Flows are constantly torn down with very little chance of new flows being succesfully setup. When sufficient flows have been torn-down, the processing load decreases and new flows can be setup causing link utilization to increase. This process repeats causing oscillations in link usage. The oscillatory behavior is caused due to FIFO processing of UPDATEs, priority processing of timer expirations (to prevent bandwidth wastage due to orphaned flows), and due to processor saturation happening before link saturation (as can happen if request sizes are small in comparision with link capacity and the processor is shared amongst multiple links or has to perform multiple functions). The three combine to cause a postive feedback effect leading to oscillations.

B. Performance of Adaptive Scheduling

Figure 5 shows simulation results when the processor is not completely overloaded. The results shown are for FIFO processing of all messages, fixed weighted round-robin (with weights chosen to be proportional to service times and labeled FWRR in the plots), and adaptive weighted round-robin schemes (labeled simply as WRR in the plots). All requests have sizes between 1 and 5 kbits/s. The interarrival times of the requests are exponentially distributed.

In Figure 5, the number of flows in progress when each of these schemes is used is plotted as a function of time. Since the processor load is moderate, the effect of scheduling is not very pronounced and both FIFO and adaptive weighted round robin have similar behavior with the number of flows being slightly larger for the latter. However, for FIFO there are more spurious teardowns due to UPDATE messages not being processed in time. These spurious tear-downs are smaller for fixed round-robin and lowest for adaptive round-robin showing the usefulness of adaptive processor scheduling. The fixed weighted round robin has the worst behavior in terms of the number of flows because the static weights are not optimized for the offered load.

Figure 6 shows the same setup as that for Figure 5 but with a high fraction of the CPU usage being due to route processing. Hence the CPU is saturated even though the offered load due to RSVP messages is the same. When the CPU load increases the advantage of our mechanisms are clearly evident, in terms of number of flows in progress and in terms of reserved bandwidth. The advantage of the adaptive weight scheme is obvious from the figure. Adaptive WRR is very effective in the two extreme cases where reserved bandwidth is very low (we accept more reservations) and when the reserved bandwidth reaches the maximum link bandwidth. In this case, we maximize the number of accepted calls since adaptive WRR frees the bandwidth of closed connections faster than FWRR (Figure 6).

In the extreme cases, adaptive WRR may lead to a higher number of router timeouts than FWRR. Suppose the link is utilized very little. Then TEAR-DOWN weight decreases to a very small value and most of the processor capacity will be used to setup new flow reservations. The low rate of TEAR-DOWN processing can cause the refresh timer to expire and tear-down the

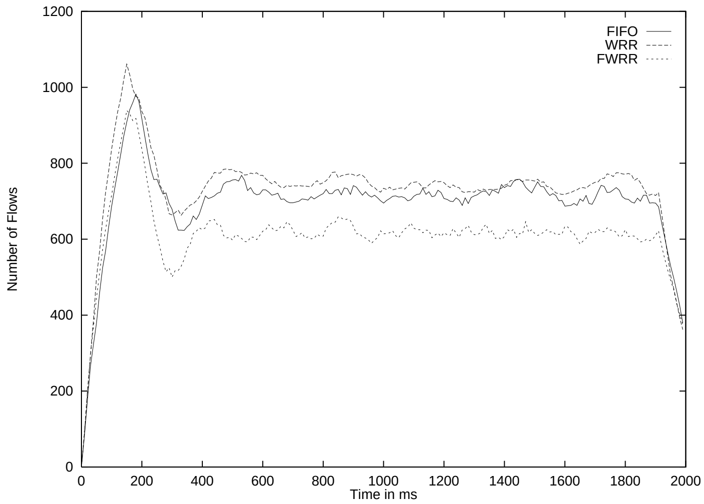


Fig. 5. Number of flows with a medium CPU Load

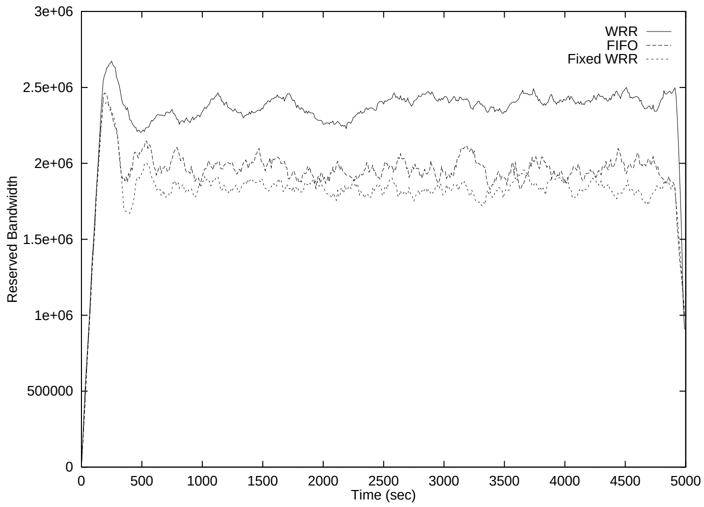


Fig. 6. Reserved bandwidth with a high CPU Load

flow. This is by choice. Since the processor is bottlenecked, we must decide whether to allocate processor resources to handle explicit TEAR-DOWNs or to allocate it to handle PATH or RESERVEs. Processing a TEAR-DOWN only makes more bandwidth available. Since the utilization is low, the system is not bandwidth constrained and so it is better to use scarce processor resources to process PATH or RESERVEs than for processing TEAR-DOWNs. Hence, TEAR-DOWN processing can be delayed as much as is allowed. By doing so, we just increase the holding time by the timeout period for the flow for which the TEAR-DOWN was not processed in time. This does not adversely affect performance since this happens only when link utilization is low.

Figure 7 shows the number of flows in the first router for a configuration where RSVP sources are connected to the first router which is connected by a single link to a second router to which are connected RSVP receivers. The processors in both routers are saturated. Both routers have a larger number of flows in progress (though only the number of flows in one of the routers is shown in the figure) under the adaptive weighted round robin scheme than with the FIFO scheme. Clearly, proper processor scheduling can improve utilization of available link bandwidth. In the example in Figure 7, there is approximately a 40% increase in the number of flows in progress.

Figure 8 shows that adaptive weighted round robin can prevent oscillations in link utilization (shown in the figure in terms of the number of flows in progress). This is because UPDATEs are always processed at a rate pro-

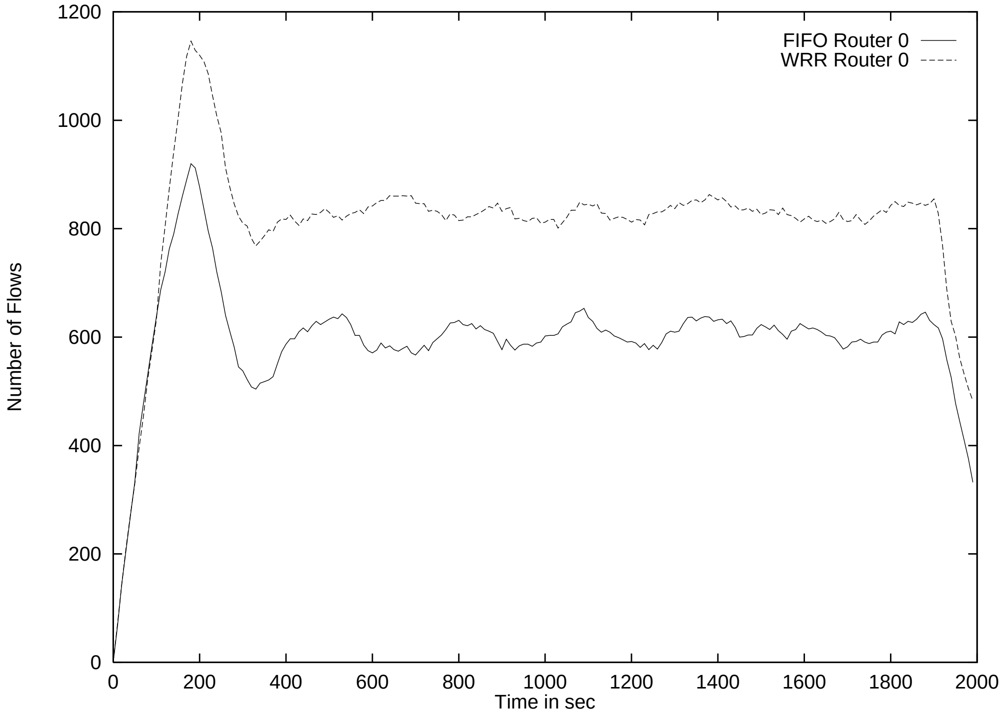


Fig. 7. Number of flows in router 1 with a high CPU Load and 2 routers in tandem configuration

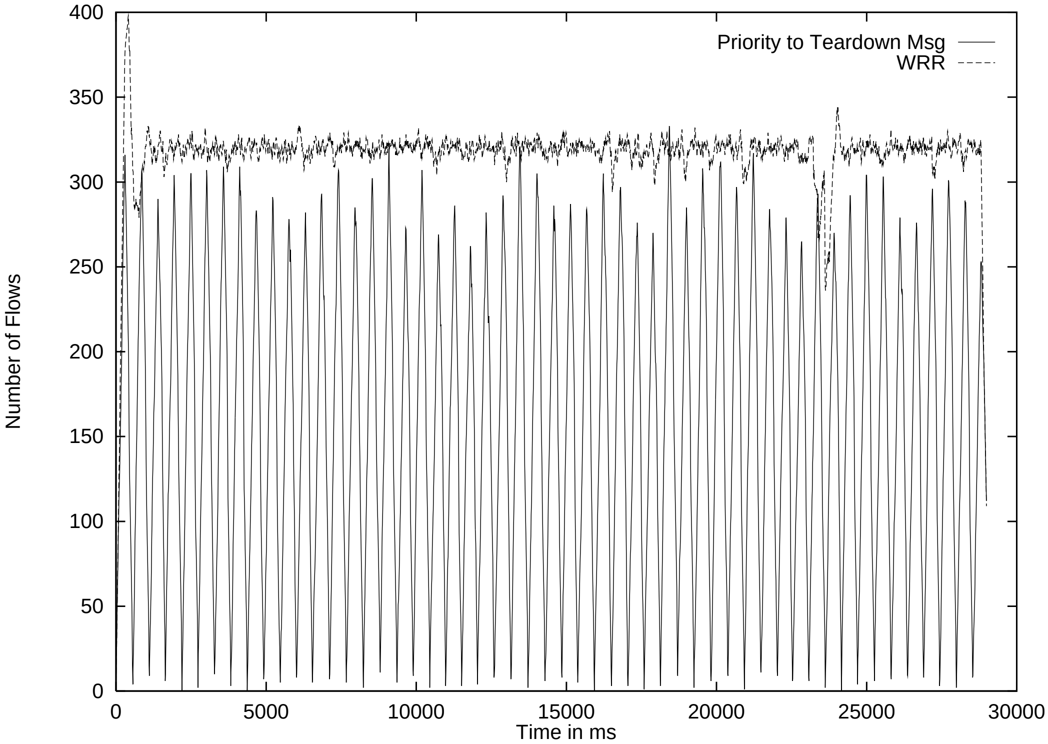


Fig. 8. Comparison of link utilization for FIFO combined with priority to TEAR-DOWNS with adaptive weighted round robin

portional to the number of flows in progress and are not queued behind PATH and RESERVE messages wait-

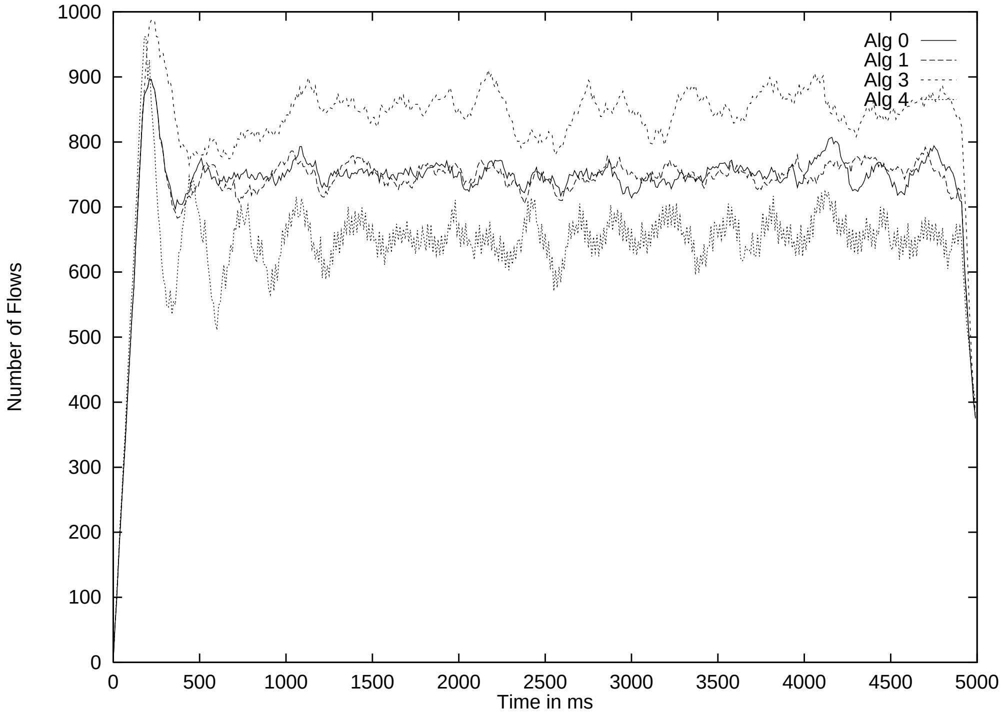


Fig. 9. Number of Flows for the algorithms with a high CPU Load

ing to be processed. Unless the UPDATE messages by themselves can saturate the processor (or if the UPDATE weights are set too low), UPDATE messages are not dropped or delayed long enough to cause timer expiration. The positive feedback effect, evident in FIFO processing, due to delayed UPDATEs causing timer expirations which further delay other updates and cause more timer expirations is not likely to happen since UPDATEs are seldom delayed long enough to cause timer expirations. Hence, oscillations in link utilization are reduced drastically in comparison to FIFO scheduling where the oscillations are so large that the number of flows in progress periodically becomes almost zero.

C. Variations to the basic adaptation scheme

We studied the impact of 4 different adaptive weight calculations.

- In the basic scheme, we calculated the weights as described in the simplest version in the previous sections.
- The second scheme, includes the queue content to the above calculation and so biases the basic scheme to message classes with longer queues.
- The third scheme improves on the second by multiplying the weights by the normalized work accumulated in the corresponding message queues.
- The fourth scheme determines the weights only according to the queue-length of each message class.

Figure 9 shows the resulting behaviour of the four algorithms (they are marked alg0, alg1, alg3, and alg4 respectively in the figure). The plots shows the number of flows in progress when the different algorithms are used and when the processor is saturated due to background task and RSVP message processing. We do not plot results for FIFO processing since we have already seen that FIFO performs poorly in comparison with basic adaptive scheme.

Since the basic adaptive scheme does not take queue lengths into account, it is not very responsive, when computing weights, to bursts of PATH, RESERVE or TEAR-DOWN requests that may arrive. Instead, the weights are set purely based on link-state, and average request sizes. If the link is not in the extreme ends of utilization, the weights change slowly (as they adjust to track the average request sizes which may change). Making the weights to be a function of the queue length permits the weight to change in response to increased or decreased arrivals of a particular message class. For the second scheme, we calculated the weights w_{PR} and w_t as follows: To determine w_{PR}, we first calculate n_{PR} as before. Then, calculate a new n^q_{PR} as a weighted sum of n_{PR} and the queue length of PATH and RESERVES. We varied the weight given to the queue length in the range 0.5 to 1. Then w_{PR} is calculated as in the basic scheme except that we use n^q_{PR} instead of n_{PR}. The TEAR-DOWN weight w_t is calculated in a similar manner.

Since using queue lengths does not account for service times, the third scheme instead of using queue lengths uses accumulated work in each queue. We calculate the weights w_{PR} and w_t as in the basic scheme. We then multiply these weights by the normalized work in each queue. This should account for differing arrival patterns much better than the second scheme and this can be seen from Figure 9. The plots show that a combination of link utilization, request sizes (as in the basic scheme) combined with work in queue informaton gives the bests results. The fourth scheme which only uses queue-length information is unable to adapt to link-state and hence has the poorest performance. The basic scheme and the basic scheme with queue lengths added have approximately the same performance.

V. Concluding Remarks

Control processor scheduling is an important issue that can impact both the performance and robustness of networks. Yet processor scheduling in routers has not been studied as intensely as link scheduling has been. Considering the case of processor scheduling for RSVP message processing in routers, we showed that simplistic processor scheduling can cause poor link usage and cause blocking even when link capacity is available. We considered both the case of a processor per link and processor per group of links. We showed that FIFO, or FIFO combined with simple priorities are not adequate for satisfactory link utilization in situations where the control processor is somewhat highly loaded. We then proposed the use of simple adaptive hierarchical weighted round-robin scheduling to make best use of the links when the processor is loaded. We proposed adaptation schemes using link state, request size, and queue informaton. We demonstrated the advantages of adapation by simulation of different variants of the adaptive scheduling scheme. An area of future work is to include into the scheduling schemes the processing of other protocols and tasks that the control processor may perform (such as processing multicast joins/leaves) so as to improve performance and robustness under transient processor overload. Also, analytical explanation of the oscillations in link utilization, and study of network wide effects remain areas for future study.

References

[1] R. Braden et. al., "RSVP: A New Resource ReSerVation Protocol", *IEEE Network* September 1993. See also http://www.ietf.org/html.charters/rsvp-charter.html.

[2] A. Campbell, G. Coulson, "QoS Adaptive Transports: Delivering Scalable Media to the Deskto", *IEEE Multimedia*, March 1997.

[3] D. Clark, D. L. Tennenhouse, "Architectural Considerations for a New Generation of Protocols", *Proceedings of ACM SIGCOMM '90*.

[4] M. Degermark et. al., "Small Forwarding Tables for Fast Routing Lookups", *Proceedings of ACM SIGCOMM '97*.

[5] Integrated Services Working Group, http://www.ietf.org/html.charters/intserv-charter.html

[6] V. P. Kumar, T. V. Lakshman, D. Stiliadis, "Beyond Best Effort: Router Architectures for Tomorrow's Internet", *IEEE Communications Magazine*, pp. 152-164, May 1998.

[7] S. Lin, "A Simulation Study of IP Switching", *Proceedings of ACM SIGCOMM '97*.

[8] K. Nichols et. al., "Differentiated Services Operational Model and Definitions", *Internet Draft*, February 1998.

[9] J. Wroclaswski, S. Shenker, "Specification of the Controlled Load Network Element Service", *Internet Draft*, November 1996.