A Scalable Solution to Deterministic Per-Flow Resource Booking

Pier Luca Montessoro Daniele De Caneva
Department of Electrical, Management and Mechanical Engineering
University of Udine, 33100 ITALY
{montessoro, decaneva}@uniud.it

Abstract— **This paper presents REBOOK, a resource reservation management algorithm for packet switching network. It provides deterministic, fast (real-time) dynamic resource allocation and release; it can be used as an engine supporting different network-oriented techniques for Quality of Service. Based on a stateful approach, it handles faults and network errors, and recovers from route changes and unexpected flows shutdown. The distributed scheme used to store flows information avoids the need of searching for entries within the routers' control memory when packets are received and guarantees constant complexity. REBOOK can be implemented in hardware and is compatible with any packet switching network. In the Internet, it can be integrated in TCP or used with UDP to make it network friendly. Moreover, a slightly extended RSVP implementation can be used as signaling and hosting protocol. A software implementation as standalone protocol has been developed to prove its effectiveness, robustness, and performances.**

Keywords- Quality of Service; network reliability; fast resource reservation; transmission rate control.

I. INTRODUCTION

The number of multimedia services on the Internet is rapidly growing, and the importance of Quality of Service (QoS) is increasing. A major problem in providing QoS guarantees in a packet switching network comes from the difficulty of handling, in routers, the state information belonging to active flows. In the core of the Internet, the conventional known techniques simply fail in keeping up-to-date the huge amount of flows information at a reasonable cost. This paper proposes a new technique that, involving end nodes' applications or edge routers or firewalls, provides constant-cost access to resource reservation information.

Quite often multimedia applications' designers prefer UDP as transport protocol because its efficiency, although those applications generally require high bandwidth and would benefit from congestion control mechanisms. Several methods have been introduced for adding QoS control mechanisms to packet switching networks, mainly based on adapting the sender's transmission rate in accordance with the network congestion state (see Section II), but typically with such approaches no QoS guarantees can be made effectively.

The proposed algorithm, that we call REBOOK [1], allows a control protocol to prevent congestion by reserving resources in advance. REBOOK is not dependant on TCP/IP protocols, as it can be used in any other packet switching network. Obviously, in the following the Internet and the TCP/IP protocol suite will be used as reference environment for its description.

REBOOK requires a hosting protocol to carry the algorithm's messages. They can be handled by a dedicated signaling protocol, like RSVP [3] or a new *ad hoc* protocol, or it can be embedded in a data transport protocol, like TCP using the options field.

In REBOOK, the node requested to send QoS-sensitive data (e.g., the sender of a multimedia stream) is responsible for resource reservation request, on the bases of the amount and type of data to be transmitted, application constraints and, possibly, SLA (Service Level Agreement) parameters. While the connection is active the amount of granted resources may be reduced to allow the activation of new flows in an almost-congested network or may be increased if switching nodes become less loaded. These events are acknowledged by the sender that will consequently adapt its transmission rate. RSVP is receiver-oriented mainly because it is designed to support singlecast and multicast flows as well. A possible implementation of REBOOK in RSVP for multicast support has been designed.

The proposed algorithm does not rely on any special network feature: it works even if only part of the network is REBOOK-aware; the resource reservation is effective even if part of a flow traverses unaware routers. There are no special requirements to routing, that can be asymmetrical (transmitting and receiving flows can follow different paths), except, obviously, its stability: in normal conditions data packets and control messages must follow the same route for the duration of the connection.

REBOOK does not rely on special hardware in routers either. Its status storage scheme allows direct access to table entries, without any hardware lookup feature, using conventional memory architecture. This makes its implementation faster and cheaper than today's typical solutions provided by hardware hashing.

Finally, REBOOK does not require any improvement in the switching fabric. No additional memory for queues and buffers nor different packet handling. On the contrary, the router architecture will drive the resource granting phase, depending on available resources left by previous reservations.

In the following, after a summary of related work in Section II, Section III discusses the scalability of the proposed stateful approach, whereas Section IV presents the algorithm itself. Section V and VI analyze implementation issues and experimental results, respectively. In Section VII, some conclusions are drawn and the future work is presented.

II. RELATED WORK

Congestion control in IP networks is a challenging issue, since it represents a critical factor for the robustness of the Internet [2]. Reservation of resources, admission control and traffic policing are among the most commonly used open-loop mechanisms to avoid congestion. On the other hand, closed-loop mechanisms rely on feedback to detect and prevent congestion [3].

A. Resource reservation and management

The IntServ architecture is an interesting implementation, because it uses RSVP [4] to reserve the resources required by the QoS-sensitive user's applications. Nevertheless, experience with real networks has revealed severe scalability problems of this architecture, due to the amount of routing and reservation information stored inside the routers.

Furthermore, RSVP implements a soft state model and uses periodic refresh control messages to manage its states, that introduces signaling overhead [5].

On the other side, the approach proposed by DiffServ is based on flow aggregates: it allows an efficient implementation inside the network. On the other hand, it conserves a statistical approach to resource provisioning and thus it does not provide any real service guarantee to any possible flow [6]. Enhancements to DiffServ may come from the Bandwidth Broker (BB) [7][8].

Cross-layer congestion control in IP networks has been addressed by XCP [9] that proposes a protocol-oriented model that puts the control state in the packets and not in the routers, with the objective of improving the scalability. Unfortunately, such schemes are hard to deploy in today's Internet [10].

Recent studies have also demonstrated that soft-state approaches coupled with explicit removal substantially improves the degree of state consistency while introducing little additional signaling message overhead [11]. This is the direction followed to design REBOOK.

B. Efficiency and path recovery

Resource reservation in packet networks is widely recognized as an essential requirement for applications, which require guaranteed minimum bandwidth, low delay, or both [12]; several fast resource reservation protocols are thus being studied and developed.

In order to increase RSVP efficiency, REDO RSVP [4] proposed a refresh mechanism made per aggregation instead of per flow, improving RSVP scalability by reducing the granularity of signaling information.

The YESSIR [13] and the LFS [14] protocols attempt to avoid complexity of RSVP by limiting its objectives and fulfilling bandwidth reservation for one-way, unicast flows.

MPLS with Traffic Engineering sets up label-switched paths (LSPs) along links with available resources: thus, ensuring that bandwidth is always available for a particular flow and avoiding congestion both in the steady state and in failure scenarios. The paradigm is that MPLS can easily address prioritized and/or guaranteed traffic along an arbitrary path, which can be independent from the underlying routing protocol. This would allow enhancing network utilization and fairness [15][16].

Interesting works on path selection algorithms are shown in literature [17][18][19], giving also some interesting solutions to failure recovery and QoS-aware fast rerouting procedures [19][20]. Nevertheless, the optimal resource allocation on all links and nodes along a reserved path when a topology change occurs is a common challenge, which relies on the trade-off between scalability and overall efficiency in resource management.

REBOOK is based on a Distributed Linked Data Structure (DLDS). DLDS are linked data structures that keep pointers to memory locations or indexes to table entries containing information stored in the routers that are very likely to be accessed in the future. When a packet whose handling requires access to that information is received, the DLDS pointer/index can be found in the packet itself and lookup procedures can be avoided. DLDS are *distributed* data structures since each pointer/index is not stored within the router it addresses, but in the end nodes, in the packets, and possibly in adjacent routers. DLDS are *dynamic*, since the collection of pointers/indexes is dynamically built and travels along the routes between the end nodes. To keep the pointers consistent in a dynamic environment, where route changes may send packets containing a pointer/index belonging to a router different by the one being traversed, a specific integrity check is adopted. Thanks to DLDS, REBOOK can improve many known techniques because it provides an efficient way to handle reservation information within the network nodes regardless the actual strategy to assign resources to the flows.

III. SCALABILITY

One widely accepted paradigm in networking is that packet switching is a scalable technique because it does not keep information for each connection (flow) traversing a node. Stateful approaches, in which some information are kept up to date for each connection in every router along the path, are not generally believed to be scalable enough to handle the increasing traffic on the Internet.

However, memory is no longer a limitation with today's technology. Provided that a tuple describing the status of a flow should contain network and transport layer addresses, some fields about the allocated resources and some control fields, we can roughly estimate a memory occupation of about 50 bytes per active flow, even for 128-bits IPv6 network addresses. In a conventional 4 GB memory the router could store information belonging to almost 86 million flows.

The open problem is the excessive computation time needed to handle the state information. As shown in the following, REBOOK solves this problem with a distributed status storage

scheme that keeps track of memory addresses in routers and allows direct access to the stored information, well avoiding the need of searching data continuously.

IV. THE ALGORITHM

The REBOOK algorithm allows resource booking/release on a packet network. It is based on several key concepts:

1) a distributed scheme for storing the resource reservation status
2) each router keeps a very limited amount of information for each flow requiring resources (as shown above, REBOOK can handle millions of flows in very high speed nodes)
3) a distributed scheme for keeping track of memory addresses (pointers) overcomes the need for searching the resource allocation tables
4) "keepalive" messages periodically signal the persistence of each flow along the path; the routers use this signaling to recover from route changes, uncommitted flow shutdowns, hosts or nodes faults, loss of REBOOK messages
5) the order of nodes traversed by a flow is kept in a distributed form and used to discover route changes.

REBOOK provides a unidirectional resource reservation, in the sense that to reserve resources in both direction of a flow two instances of the algorithm should be activated, even though some setup and shutdown messages can be glued together. REBOOK, in fact, can be easily integrated in existing transport or application layer protocols to merge end-to-end session setup and hop-by-hop resource reservation.

REBOOK works as follows: when a flow requires resource reservation, the host (or a border router that controls the QoS parameters for flows accessing the network) sends to the receiver (host or border router) a reservation request message that is normally routed along the path. Each node keeps track of the request and reserves the requested resource or the amount still available (if less than requested). Reduction in the allocated resources is written in the resource reservation request message while it traverses the router. The receiving end system sends back to the sender a resource reservation acknowledge notifying the current amount reserved. From that point on, periodical keepalive messages are sent to confirm the activity of the flow and to notify the routers along the path the current amount reserved. If the amount reserved by a router was subsequently reduced by the next ones along the path, from keepalive messages it will know how much can be released. Keepalive messages are also used to notify a booking reduction request from a overloaded router.

So far, REBOOK appears quite straightforward. The problem is that in the real world flows can suddenly disappear (due to host or router faults) or change way (dynamic routing). It is mandatory the quick identification and release of all the allocated resources that are no longer used. Even the smallest fraction of "lost" resources would produce catastrophic effects when cumulated over the days, months or even years long routers uptime. This requires continuous update of the resource allocation tables in routers, a task that so far has been believed to be too expensive, requiring special hardware architectures (e.g., Content Addressable Memories, hardware hash tables) or high computing power. The REBOOK's distributed status storage scheme overcomes these limitations with a pure software solution.

In order to make the resource reservation, the end node must communicate to the network the minimum amount of resources needed for the application to work and the maximum amount that can actually be used. How an application could know this information? And, more important, why an end node should not attempt to reserve much more than what it needs? There are several reasons, some general and others related to specific environments. The self-regulation of the end node is not new: the very basic TCP congestion control drastically reduces the transmission rate after a packet loss. Nodes share the resources trying not to overload the network. In the same way a multimedia application, on the basis of encoding and compression information, knows the peak transmission rate and the minimum bandwidth required to play the stream without interruptions. The reasons because a node should not overestimate its requirements depend on the environment. The most obvious case is a controlled environment where nodes are set-top box or computers with specific accounting and descrambling hardware for pay-per-view web TV distribution or similar multimedia services. A more general scenario is provided by ISPs that implement traffic shaping to limit high-speed peer-to-peer download; self-reducing bandwidth requests would avoid generalized slow-down of the user access and, on heavily loaded networks, connection refusal due to excessive resource reservation. In the future, speed-based accounting could become quite common; reducing the required bandwidth for non-real-time applications could help reducing access costs.

A. Rebook Messages and Data Structures

Figure 1 represents the REBOOK message fields. In order to compute the message size, IPv4 address format has been considered. Please note that for some message types ("RESET", for instance), not all the fields are needed, but in the following we will ignore this possible optimization.

A RESV message is used to start resource booking. The flow identification field is made of the tuple { *source IP, source PORT, destination IP, destination PORT* } and uniquely identifies a TCP or UDP flow. The resource reservation request is expressed by the *Resource* field, containing the minimum amount of resource needed by the flow to support the application and the maximum amount of resource that it can really use. R_{curr} is the actual resource available and reserved along the path; this value is reduced by the traversed router if the available resource is less than the maximum required. The remaining fields belong to the direct table access and fault recovery algorithms, and will be discussed later.

The reservation request carried by the RESV message is

Type	Flow ID	Resource	Path Length	Hop Counter / Reset	Resource Reservation Vector (RRV)
source to destination: RESV, KPALV UP_RESV, RL_RESV destination to source: RESV_ACK, PRL_ACK, RESET	IP_s, P_s, IP_d, P_d	R_{req}, R_{min}, R_{curr}	P_{LEN}	HOPS (N, -1 when reset)	[RAT index] $_{1..N}$ (RAT: resource allocation table)
1 byte	12 bytes	3 x 2 bytes	1 byte	1 byte	N x 3 or 4 bytes

Figure 1. REBOOK message format.

Flow ID	Resource	Path Position	Age	Local physical ports	Resource Release Request	List pointer
IP_s, P_s, IP_d, P_d	R_{req}, R_{min}, R_{curr}	P_{POS}	Age	IN, OUT	R_{rel}	list_ptr
12 bytes	3 x 2 bytes	1 byte	2 bytes	2 bytes	2 bytes	4 bytes

Figure 2. Resource Allocation Table format.

acknowledged by a RESV_ACK message, from receiver to sender. The KPALV type is used to identify the keepalive messages, from sender to receiver, while UP_RESV, RL_RESV, PRL_ACK are used to dynamically change the amount of allocated resources. The RESET message type signals that a flow's resource booking is no longer valid due to an unexpected event along the path.

Each time a new flow requires resource booking, the router creates a new *Resource Allocation Table* (RAT) entry. Its format is fixed, 29-bytes wide (IPv4, 53 bytes in IPv6), as shown in Figure 2.

The new entry is created from a free list using the *List pointer* field. The RAT entry stores the Flow ID field, the amount of requested and currently allocated resource and the system time (the *Age* field), used to identify reservations hanging due to faults or network errors. The two fields IN and OUT are router's implementation-dependent, and can be substituted by the actual internal information the router needs to handle the resources release when the flow terminates.

The field *Path Position* is the key that makes REBOOK robust at a very minimum cost and will be discussed in next Section.

The field R_{rel} allows a partial resource release when the router needs to free resources to make room for new flows.

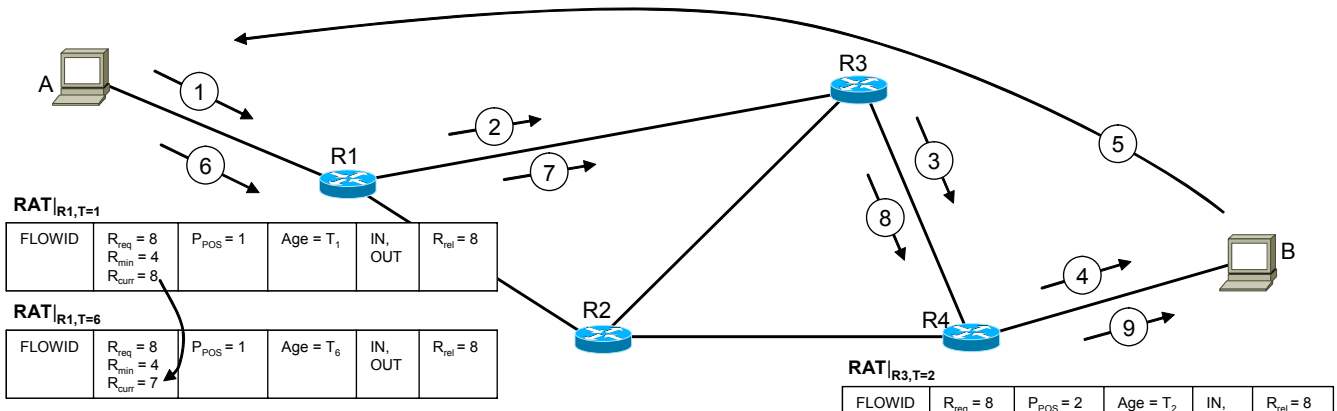
B. Resource Reservation Algorithm

Figure 3 represents a simple network in which host A must reserve a resource along the route to host B.

At the very beginning, host A sends a RESV message to the receiving host B requesting R_{req} resource (8 for instance) and stating that R_{min} (4 in the example) is the minimum acceptable. The R_{curr} field is initialized to R_{req}. The packet is travelling along the first hop, so the *Hop Counter* field is set to 1. The total length of the path (P_{len}) is still unknown and left to 0.

The router makes the resource reservation (if enough is available) and creates a RAT entry. The path position field (P_{pos}) is set to the current hop counter received in the RESV message and the *Age* field is set to the current system time. If resource availability is less than requested, the R_{curr} field in the outgoing message is set to the actual value (router R3 at step 3 in the example: $R_{curr} = 7$). Of course, previous routers in the path do not know yet that part of the reserved resources will not be used due to subsequent bottleneck: they will release them as soon as a keepalive message will be received.

Before increasing the hop counter and forwarding the RESV message to the next hop, a *Resource Reservation Vector* (RRV) entry is appended to the message, to save the index (or the memory address) of the newly created RAT entry. This index will return to the router in subsequent keepalive messages and will be used to update the RAT entry without the need of searching. Of course this approach makes the RESV messages increase their length from sender to receiver. This is limited to the RESV messages, only once for each flow, and the maximum message length for a 128 hops route (approximately the maximum length in IP, being half the maximum *Time To Live* value) would be less than 600 bytes. Anyhow, an alternative implementation for small fixed size messages is possible at the cost of a single backward message for each router traversed by the RESV message. This implementation is based on an address swapping mechanism.



1) *{"RESV", FLOWID, R_{req} = 8, R_{min} = 4, R_{curr} = 8, P_{LEN} = 0, HOPS = 1, RRV = NULL }*

2) *{"RESV", FLOWID, R_{req} = 8, R_{min} = 4, R_{curr} = 8, P_{LEN} = 0, HOPS = 2, RRV = [$ADDR_{R1}$] }*

3) *{"RESV", FLOWID, R_{req} = 8, R_{min} = 4, R_{curr} = 7, P_{LEN} = 0, HOPS = 3, RRV = [$ADDR_{R1}$, $ADDR_{R2}$]}*

4) *{"RESV", FLOWID, R_{req} = 8, R_{min} = 4, R_{curr} = 7, P_{LEN} = 0, HOPS = 4, RRV = [$ADDR_{R1}$, $ADDR_{R2}$, $ADDR_{R3}$]}*

5) *{"RESV_ACK", FLOWID, R_{req} = 8, R_{min} = 4, R_{curr} = 7, P_{LEN} = 4, HOPS = 0, RRV = [$ADDR_{R1}$, $ADDR_{R2}$, $ADDR_{R3}$, $ADDR_{R4}$]}*

6) *{"KPALV", FLOWID, R_{req} = 8, R_{min} = 4, R_{curr} = 7, P_{LEN} = 4, HOPS = 1, RRV = [$ADDR_{R1}$, $ADDR_{R2}$, $ADDR_{R3}$, $ADDR_{R4}$]}*

7) *{"KPALV", FLOWID, R_{req} = 8, R_{min} = 4, R_{curr} = 7, P_{LEN} = 4, HOPS = 2, RRV = [$ADDR_{R1}$, $ADDR_{R2}$, $ADDR_{R3}$, $ADDR_{R4}$]}*

8) *{"KPALV", FLOWID, R_{req} = 8, R_{min} = 4, R_{curr} = 7, P_{LEN} = 4, HOPS = 3, RRV = [$ADDR_{R1}$, $ADDR_{R2}$, $ADDR_{R3}$, $ADDR_{R4}$]}*

9) *{"KPALV", FLOWID, R_{req} = 8, R_{min} = 4, R_{curr} = 7, P_{LEN} = 4, HOPS = 4, RRV = [$ADDR_{R1}$, $ADDR_{R2}$, $ADDR_{R3}$, $ADDR_{R4}$]}*

Figure 3. Resource Reservation.

When the RESV message is received by host B it is sent back to host A in the form of a RESV_ACK message. At this point both hosts know the maximum amount of resource reserved along the path, the length of the route and the list of indexes/memory addresses where the status information are stored in the routers. Host A sends the first keepalive message and will keep sending them periodically until the flow is terminated.

C. Errors, faults and route changes

Handling a keepalive message in the routers consists in three tasks. Thanks to the distributed recording of RAT indexes, the complexity of keepalive messages handling is *constant* regardless the number of active flows.

At first, an integrity check is performed, for security reasons and to identify errors, faults or route changes. The current hop counter in the KPALV message is checked, then it is used to get the RRV entry containing the index or the memory address of the flow entry in the router's RAT. The stored *Flow ID* and *Path Position* fields are compared with the *Flow ID* and the hop counter in the message. If some changes or faults occurred along the route, these values no longer match and a new resource booking reset request is signaled by setting the *Hop Counter* field to the *reset* value (-1) in the message to be forwarded.

The second task while handling a keepalive message is to partially release the allocated resources if greater than the amount reached at the end of the path by the RESV message, value now stored in the R_{curr} field. This is the case of router R1 at time 6 in Figure 3. There is no need for an explicit booking confirmation to the routers since keepalive messages will always contain this information. Unnecessary booking will be released sooner or later even if some keepalive message is lost. Frequency of keepalive messages must be set according to the required reaction time to unexpected events.

When a route change happens or a flow unexpectedly dies, keepalive messages no longer update the *Age* field in the RAT entry. Booked resources will be released thanks to a low-priority process that scans the list of active RAT entries and removes the expired ones.

D. Dynamic Resource Allocation

Sometimes a router may be required to allocate some resources, but it might happen that not enough are left. Nevertheless, it is possible that some flows have been allocated more resources than the minimum requested. Here comes the third task when handling keepalive messages: the dynamic resource reallocation. When needed, some RAT entries may be marked for resource release by setting the R_{rel} field to a value less than R_{curr}. When a keepalive message is processed for those entries, the R_{rel} value is set in the R_{req} field of the message to be forwarded, so that the receiving host B will notify the request by sending a partial release message (*PRL_ACK*) to the transmitting host A. Host A will reduce the corresponding activity and will put in subsequent keepalive messages the new R_{curr} value.

Two other message types complete the algorithm: UP_RESV and RL_RESV. UP_RESV is sent from the transmitting host to attempt a resource allocation upgrade for a flow currently active. Their handling is similar to the one for RESV messages. RL_RESV is used to release the allocated

resources under normal circumstances, that is, when the transmitting host terminates the connection and explicitly requires the resource release along the path.

E. Multicast

In order to support multicast flows, reservation and keepalive messages must be replicated at the multicast tree forks. Slightly different procedures in respect of the ones described above shall be called when a received REBOOK message is encapsulated in a network layer packet containing a multicast destination address. Obviously, the REBOOK engine must access (or receive) the multicast routing information, but this happens only during the resource reservation setup phase, as the output ports and other useful information will be stored in the *Multicast Resource Allocation Table* (MRAT) table. Unlike the RAT, in the MRAT a multicast flow is represented by a linked list of entries, each one related to a branch toward the destinations. Some fields are common to all the linked entries and an optimized implementation may collapse them into a single record: *Flow Id,* R_{req}, R_{min}, *Path Position, Age, IN*. The remaining fields are branch-dependent; in particular, keeping apart the R_{min} information allows each branch to reserve a different amount of resource without affecting the other branches. This is useful for multimedia flows featuring progressive or scalable encoding, provided that the router is able to forward the multicast packets according to the contents and to the required speed on each branch. To keep up to date the MRAT entries, in each router the keepalive messages are replicated, partially rewritten and sent along each branch.

The easiest way to setup resource reservation in a multicast environment is making each router at a multicast tree fork act as both a source (toward the subsequent nodes) and a destination (toward the preceding nodes) for REBOOK setup messages. The reservation should be driven by a signaling protocol, like RSVP [3] (next section will discuss the relations between REBOOK and RSVP). This way, the reservation request is receiver-initiated, as usual for multicast services, and REBOOK may anticipate the pointers collection using a backward pointers collection. When the first REBOOK packet sent by the receiver of the data flow reaches the sender, it contains the pointers referring to the traversed routers in reverse order. Like in RSVP, the actual reservation starts in the next step, when resource reservation messages are sent by the sender of the data flow to the receiver(s).

Dynamic joining and removal of hosts to and from the multicast services will be handled by the REBOOK agent active in each router.

V. IMPLEMENTATION ISSUES

Designing a REBOOK implementation integrated in industry-level routers requires choices depending on economical and technical constraints coming from hardware manufacturers and is therefore beyond the purpose of this paper. However, some general considerations can be drawn.

A. Impact on switching architectures

REBOOK does not require any dedicated hardware solution, even though it can be partially or fully implemented in hardware. The REBOOK management engine is required to handle reservation request/release and keepalive messages only, whose rate is orders of magnitude slower than the data traffic. The only mandatory constraint is the presence of an ingress filter to identify REBOOK messages: they must be delivered to the management process and are sent back to the switching fabric for forwarding. Depending on the architecture, this may require an additional internal buffer.

The switching architecture should not be affected by REBOOK at all. The REBOOK engine will contain parameters and rules stating the switch capabilities (port-to-port bandwidth, buffers length, etc.) and will continuously keep track of the amount of resources still available, in order to acknowledge or not the reservation requests. If all the data traffic was REBOOK compliant, resource would never be overloaded (e.g., buffer overflow) and no data packet would be lost, simply because the sender would not be allowed to transmit (or would be allowed to talk slower). In the more realistic scenario where only part of the traffic could be REBOOK compliant, priority tags or Type of Service fields could be used to identify REBOOK flows; packets belonging to such flows can be handled by separated queues and buffers to isolate them from the non-REBOOK traffic and to fulfil the resource reservation for REBOOK traffic. It is worthwhile to notice that REBOOK aims at reporting to the sender the maximum transfer rate allowed along the path to the receiver; as long as the sender respects this boundary, best-effort routers provide a QoS-like service.

B. RSVP and other hosting protocols

REBOOK can be implemented as a standalone control protocol and/or can be integrated in existing protocols, e.g., RSVP or TCP.

REBOOK can be used to improve RSVP by efficiently handling its resource reservation requests and by providing a deterministic tracking of the amount of reserved and available resources. As discussed above, the main adjustment required in respect of the algorithm described in Section IV is that the pointers collection may start during the RSVP receiver-initiated reservation request, instead of being activated by the sender. However, reservation setup and final confirmation of pre-allocated resource amounts are sender-driven: the sender knows the data flow bandwidth constraints, and the reservation messages are always processed in the same direction, thus working with symmetrical and asymmetrical routing as well.

To support multicast flows, REBOOK can work within RSVP provided that in intermediate nodes of the distribution tree the same RSVP process that merges reservation requests for multicast flows manage the entries in the MRAT described in Section IV.

Many multimedia streaming applications use TCP connections to control UDP data flows. REBOOK can be

integrated in TCP packets thus drastically reducing the need of additional packets for keepalive messages.

Anyhow, the easiest implementation is designing an *ad hoc* protocol around the REBOOK messages. This is the way followed for the experiments presented in the next section.

C. Deployment

We foresee several, non-exclusive ways to make REBOOK available at the application layer. The first and the most obvious one consists in including the REBOOK algorithm within clients and servers software for QoS-critical applications. It is quite easy to implement REBOOK as an add-on for already existing applications and let it negotiate the bandwidth required for the specific application. REBOOK could be implemented as a browser plug-in embedded within web pages of multimedia services. In this trail we are following the development of a plug-in module for the widespread multimedia application VLC.

An alternative that reduces the implementation effort is the deployment of REBOOK-aware agents at the boundary between each network and the Internet (Figure 4).

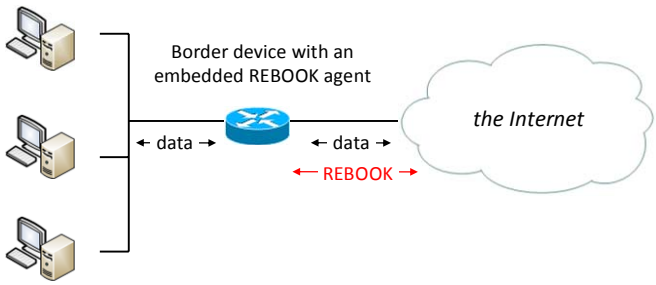


Figure 4. REBOOK agent at network edge.

Such agents will autonomously negotiate the required bandwidth with outer REBOOK aware routers on the basis of the traversed traffic, managing the QoS needs of the whole network. An agent of this kind could be easily integrated within firewalls that perform stateful packet inspection (some examples can be found in the web site of the major producers [22] [23] [24]). For a stateful firewall already keeping track of each traversed flow it is straightforward to handle REBOOK messages. Moreover, agents could be installed within boundary routers too, especially if they are software routers. A natural extension of this approach is the integration of REBOOK agents within border routers and traffic shaping appliances. In fact, flows that require resource reservation are typically characterized by an almost constant bit rate and thus they are easily identifiable and manageable by automatic resource reservation. This can be applicable even for networks invested by billions of flows per second thanks to the many algorithms for elephant flow identification present in literature [25] [26] [27]. This way the traffic shaping router will became an integrated bandwidth manager for the network; it is responsible of classifying flows, automatically issuing reservation requests only for flows that really need it, assuring fairness in bandwidth sharing and finally assuring that flows will not exceed the bandwidth reserved to each of them. Moreover, it is worthwhile to notice that REBOOK agents would prevent possible Denial-of-Service attacks based on REBOOK messages, as no reservation request could be accepted if coming directly from end nodes.

A key feature of the REBOOK algorithm is that REBOOK-aware devices and hosts may be deployed progressively. Obviously, it is impossible to deploy REBOOK or any other new protocol at the same moment throughout the entire Internet. Indeed, REBOOK might never be deployed everywhere. However, since REBOOK does not interfere with routing, unaware routers are transparently traversed by REBOOK messages. Only REBOOK-aware nodes handle the messages as described and guarantee the resource reservation. Nevertheless, as will be discussed below, REBOOK may improve network performances even in partially deployed networks.

Intermediate clouds that do not support REBOOK are not capable of performing resource reservations, so strict service guarantee cannot be made. However, if such clouds have sufficient excess capacity, they can provide acceptable and useful real-time services. The problem is now shifted to estimation of the service provided by that cloud. Depending on the real framework, this problem has different solutions (Figure 5).

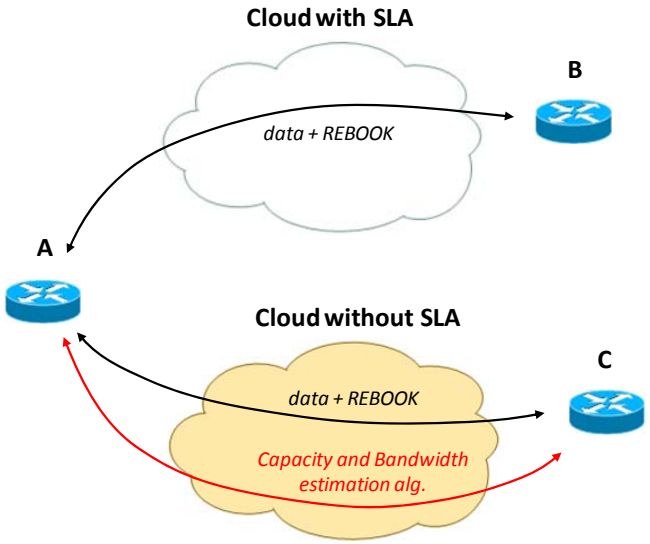


Figure 5. REBOOK and SLA support.

In the first case (A-B) the owner of the REBOOK-aware network uses the cloud to perform tunneling and has some kind of SLA with the owner of the cloud. The straightforward solution is to assign to each flow routed through the cloud a reservation compatible with that SLA. If there is no agreement between the owners (case A-C), the resource reservation control may be driven by the end systems (that monitor the data flow) instead of intermediate systems (that should communicate the available resource amounts). Non-REBOOK-aware nodes traversed by a data flow may drop packets if congested; the packet loss or keepalive messages missing rate may be monitored by the receiver; when these

events overcome a predefined threshold, PRL_ACK REBOOK messages may be generated by the receiver itself in order to reduce the reserved bandwidth and, therefore, the maximum sender's transmission rate. In extremely congested situations, when entire sequences of keepalive messages are missing, the reservation (and the flow) may be dropped. However, more complex strategies can be adopted: REBOOK-aware routers that communicate through a non-REBOOK cloud could monitor the state of the route traversing the cloud by means of one of the many bandwidth or capacity estimation algorithms (e.g., [28][29][30]) and consequently make an estimation of the resource amount available for reservation.

Additionally, it is worth noting that even if a complete Internet coverage is not possible, whole REBOOK-aware networks are absolutely plausible. In fact, multi-content providers that supply a wide range of services like IP television, Internet access and telephony are becoming popular. These providers usually manage entire networks with the need for QoS guarantees for their services. Such closed, single-owner networks could easily deploy REBOOK-aware devices.

Another situation that is gaining importance nowadays is the QoS management within overlay-networks. There is an increasing interest in their use to deliver multimedia content like video on demand, video telephony and so on. Integrating REBOOK in them is easier than interfacing a real router since these architectures are software-based and run on general-purpose servers.

D. Security

Cooperation between intermediate systems and end systems requires trust. Just like the TCP congestion-control mechanism, REBOOK works as soon as all the participating nodes behave as expected. As described at the beginning of Section IV, there is no advantage for an unfaithful node that stores or communicates invalid pointers. However, a security issue may come from DoS (Denial of Service) attacks, in two directions: invalid pointers and over-reservation requests.

REBOOK provide an intrinsic robust solution to the invalid pointers problem. The consistency check prevents the use of invalid information, but a key feature of REBOOK is that each pointer is never used by agents other than the router that stored it in a previous phase of the algorithm: pointers are not *communicated* to others, but only *stored* in a distributed data structure. Therefore, they can be encrypted and signed with symmetrical cryptography, without the need of key exchange.

The over-reservation problem may come from tampered software in end nodes. But the sender-driven reservation model makes the server and the service provider, and not the end user, responsible for correct reservation. Moreover, several possible applications are related to specific highly controlled environment such as video-on-demand distribution, where the end nodes are proprietary set-top boxes or web browsers plug-in. Lastly, nodes and REBOOK applications may be required to authenticate before starting the reservation, using some of the existing protocols and data encryption mechanisms already available.

More detailed studies will be required after the applications have been designed, but this is beyond the scope of this paper.

E. Software architecture

The REBOOK engine has been fully implemented. For the experiments, a standalone connectionless UDP-based signaling protocol has been designed and used to exchange REBOOK messages.

REBOOK implementation consists in three portable modules written in C language: router, sender and receiver. Each module, or a combination of two or all of them, can be attached to software router kernels, server programs and client applications. Thanks to several preprocessor directives a Dynamic Link Library or a linkable object code can be produced; moreover, a pure C single agent or multiple instances of C++ classes can be generated, allowing the same to be included in real routers and in simulators as well.

Figures 6, 7 and 8 show the interactions between the REBOOK modules and the host and router software.

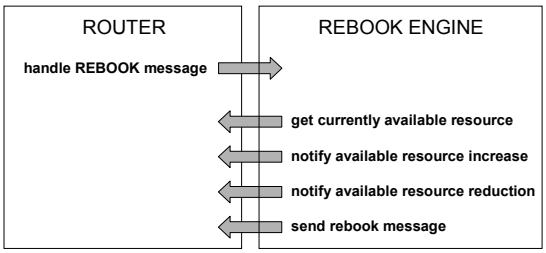


Figure 6. Router module software interface.

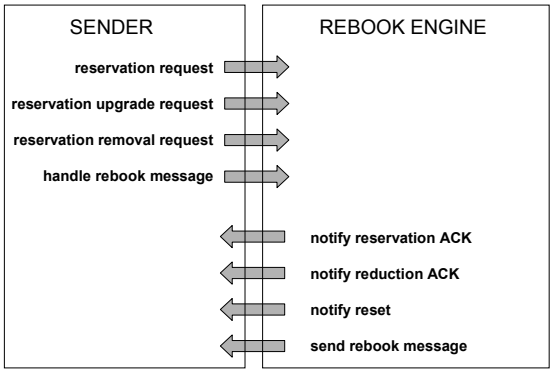


Figure 7. Sender module software interface.

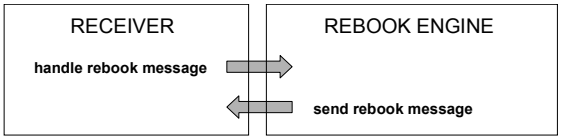


Figure 8. Receiver module software interface.

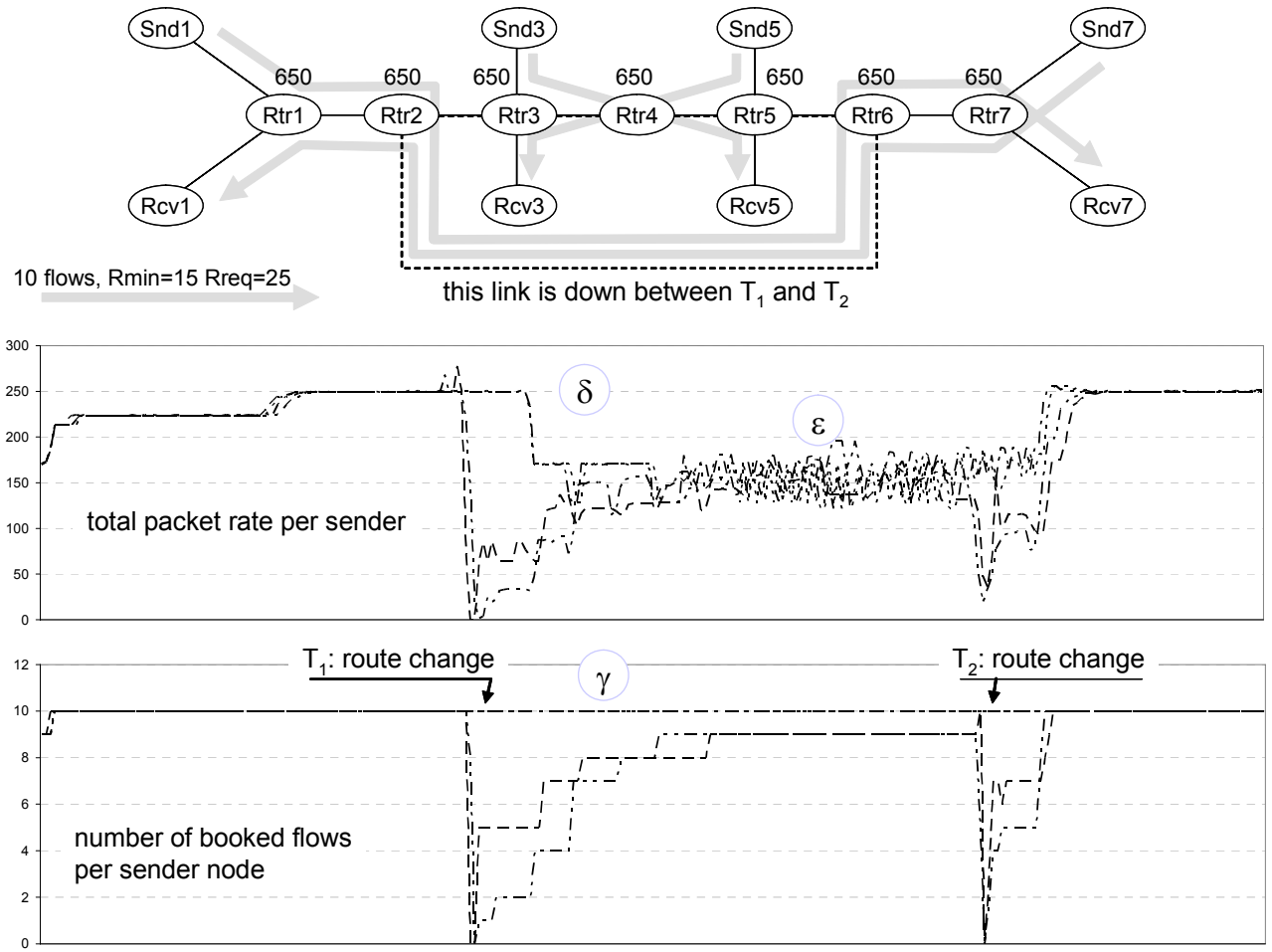


Figure 9. Congestion prevention during route change

VI. EXPERIMENTAL RESULTS

Several experiments have been designed and run to demonstrate the REBOOK robustness, performance and scalability in a real distributed environment. In order to make the experimental framework as close as possible to real, complex and possibly overloaded networks, a router emulation environment has been developed, allowing us to use computer lab Personal Computers as routers. This way, we could emulate a network with several routers actually running asynchronously, where packets can be really dropped and route changes happen asynchronously as well.

Five kinds of nodes have been developed.

Sender host: it can transmit several UDP flows towards one or more destinations; for each flow REBOOK control packets are handled to book resources along the path, and the flow transmission packet rate is modulated according with the granted bandwidth.

Receiver host: it receives and handle REBOOK messages as described in previous sections. Data packets contain a counter used by the receiver for signaling when packets are lost.

REBOOK-aware router: it is the key module, running the resource booking procedures. A periodically activated thread performs the RAT "cleanup" procedure, i.e., removes the entries expired due to repeated missing keepalive messages. Each router is assigned a maximum capacity (total number of packets per second that can be forwarded). If the traffic exceeds this value in a given time window, packets are dropped (both data packets and REBOOK messages, of course).

REBOOK-unaware router: REBOOK has been tested even in mixed environments where only some routers are REBOOK-aware. This kind of router treats REBOOK messages as normal data packets and drops packets exceeding its capacity.

Routing Control Center: this is the module that sends the routing tables to the routers. Each routing table update is acknowledge by the router and becomes immediately operative. Therefore, during route changes rules can become temporarily incoherent and packet routing errors are possible.

Figure 9 shows a 7-routers network where 40 data flows are exchanged between 4 sender-receiver pairs. Before and after the route change the network capacity is large enough to accept all the flows at full speed. When the link between routers 2 and 6 is dropped, flows from senders 1 and 7 start

competing for bandwidth in routers 3, 4 and 5. It is interesting to notice that the flows *Snd3-Rcv5* and *Snd5-Rcv3* are not affected by the route change and their booking is maintained (γ). Instead, flows *Snd1-Rcv7* and *Snd7-Rcv1* are dropped and when the senders start sending new reservation requests the routers signal to the already active senders the need of partial resource release. As soon as this happens (δ), new flows can be accepted and the system finds a new stability (ε). When the link between routers 2 and 6 is restored new reservations are made for flows *Snd1-Rcv7* and *Snd7-Rcv1* obtaining permission to send at full speed again.

Figure 10 shows the result of an experiment that demonstrates how REBOOK can be useful even in partially REBOOK-aware networks to limit the packet loss by controlling the sender's transmission rate on the basis of the packet loss rate measured at the receiver side.

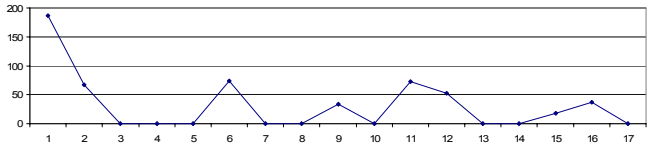


Figure 10. Packets drop control in partially REBOOK-unaware network

In the experiment, one router along the path does not support REBOOK and, in addition, it is a bottleneck due to insufficient capacity in respect of the number of active flows. The REBOOK-unaware router is transparent to REBOOK messages (the hop counter field in keepalive messages is updated by REBOOK-aware routers only). Keepalive messages and data packets are monitored by the receiver; when the number of lost packets exceeds a given threshold, the receiver autonomously generates a PR_ACK message to the sender, just like when a resource release request comes from a router. The graph of Figure 9 reports the number of dropped packet in the REBOOK-unaware router. Periodically, the sender attempts to increase the resource reservation, REBOOK-aware routers acknowledge the requests, the REBOOK-unaware router restarts dropping packets and the receiver asks the sender to reduce the transmission rate again.

Since REBOOK requires control messages in addition to data packets, a possible issue regards the traffic overhead. The experiments showed an increase in traffic load less than 1.8% with neglectable keepalive messages processing time. More precisely, on the congested network of Figure 9 we measured an increase of 5% for the seven routers total CPU time in the REBOOK-enabled run. However, since about 15% of the packets have been lost during the run without REBOOK, the average CPU time *per delivered packet* has been indeed reduced by 9%.

Routers must periodically remove obsolete entries in the RAT to free resources belonging to rerouted or dead flows. In our implementation the RAT is an array whose used and unused elements are linked in two list. To measure the actual management cost the RAT has been populated with 10.000.000 entries representing data flows with expiration times randomly distributed over 100 cleanup process scheduled activations. On a Pentium 4, 2.80 GHz, 512 MB RAM computer running Windows XP each cleanup run required, in average, approximately 100 ms CPU time.

VII. CONCLUSION AND FUTURE WORK

This paper presented an innovative algorithm for robust and deterministic resource reservation, based on a Distributed Linked Data Structure that provides direct access to flow information within the routers. This makes the algoritm computational cost *constant*, regardless the number of active flows.

Several options are available to implement the algorithm. REBOOK has been fully implemented as a standalone protocol in a software-based router emulator and has been extensively tested on heavily loaded networks with dynamically changing topologies. It demonstrated to be scalable and robust.

Many research directions can bee foreseen starting from REBOOK: investigating the integration in existing protocols, with special focus on multicast-oriented protocols and applications; REBOOK engine hardware implementation for high performance routers; REBOOK-aware firewalls, proxy servers and traffic shaping routers design; fair resource release request strategy within REBOOK-aware routers; extension to other fields like High Performance Computing (HPC) and Wireless Sensor Networks (WSN).

REFERENCES

[1] P. L. Montessoro, D. De Caneva, "A Distributed Algorithm for Efficient and Scalable Resource Booking Management," Proceedings of CTRQ 2010 Third International Conference on Communication Theory, Reliability, and Quality of Service, June 13-19, 2010 - Athens/Glyfada (Greece), pp. 128-134

[2] S. Floyd and K. Fall, "Promoting the Use of End-to-End Congestion Control in the Internet," IEEE/ACM Trans. on Networking, vol. 7, no. 4, pp. 458-472, August 1999.

[3] Cui-Qing Yang and Alapati V. S. Reddy, "A Taxonomy for Congestion Control Algorithms in Packet Switching Networks," IEEE Network Magazine, Vol. 9, Number 5, July/August 1995.

[4] L. Zhang, S. Deering, D. Estrin, S. Shenker and D. Zappala, "RSVP: A new resource ReSerVation Protocol," IEEE Network, vol. 7, no. 5, pp. 8-18, September 1993.

[5] L. Mathy, D. Hutchison, S. Schmid and G. Coulson, "Improving RSVP for Better Support of Internet Multimedia Communications," Proceedings of ICMS'99, Florence, Italy, June 9-11, 1999. IEEE press, pp 102-106.

[6] W. Almesberger, S. Giordano, R. Mameli, S. Salsano and F. Salvatore, "Combining IntServ and DiffServ under Linux," Public file.

[7] S. Sohail and S. Jha, "The Survey of Bandwidth Broker," Technical Report UNSW CSE TR 0206, School of Computer Science and Engineering, University of New South Wales, Sydney 2052, Australia, May 2002.

[8] Z. Zhang, Z. Duan and Y. Hou, "On Scalable Design of Bandwidth Brokers," IEICE Trans. Communications, Vol. E84-B, No.8 August 2001.

[9] D. Katabi, M. Handley and C. Rohrs, "Congestion Control for High Bandwidth Delay Product Networks," SIGCOMM'02. Pittsburgh, Pennsylvania, USA. August 19-23, 2002.

[10] Yong Xia, L. Subramanian and S. Kalynaraman, "One more bit is enough," SIGCOMM'05. Philadelphia, Pennsylvania, USA. August 22-26, 2005.

[11] Ping Ji; Zihui Ge, J. Kurose and D. Towsley, "A Comparison of Hard-State and Soft-State Signaling Protocols," Networking, IEEE/ACM Transactions on , vol.15, no.2, pp.281-294, April 2007
[12] F. Kuhns, J. Turner and S. Norden, "Lightweight Flow Setup for Wirespeed Resource Reservatio," Proceedings of the Allerton Conference on Communication, Control and Computing, 2003.
[13] Ping Pan and H. Schulzrinne, "YESSIR: A Simple Reservation Mechanism for the Internet," Communication review, vol. 29, no. 2, April 1999.
[14] F. Kuhns, J. Turner and S. Norden, "Lightweight Flow Setup for Wirespeed Resource Reservation," Proceeding of the Allerton Conference Communication, Control and Computing, 2003.
[15] I. Minei, "MPLS DiffServ-aware Traffic Engineering," Juniper Networks, 2004, White Paper
[16] V. Sharma et al., "Framework for Multi-Protocol Label Switching (MPLS)-based recovery," RFC 3469, 2003
[17] F. Rafique Dogar, Z. Uzmi and S. Baqai, "CAIP: A Restoration Routing Architecture for DiffServ Aware MPLS Traffic Engineering," 5th Workshop on Design of Reliable Communication Networks (DRCN), pp 55-60, 2005.
[18] T. Anjali, C. Scoglio, J. de Oliveira, I. Akyildiz and G. Uhl, "Optimal Policy for LSP Setup in MPLS Networks," Computer Networks, vol. 39, no. 2, pp. 165–183, June 2002.
[19] B.A. Movsichoff, C.M. Lagoa and Hao Che, "End-to-End Optimal Algorithms for Integrated QoS, Traffic Engineering, and Failure Recovery," IEEE/ACM Transactions on Networking, vol.15, no.4, pp.813-823, August 2007
[20] A. Kvalbein, A.F. Hansen, T. Cicic, S. Gjessing and O. Lysne, "Fast IP Network Recovery Using Multiple Routing Configurations," INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings , vol., no., pp.1-11, April 2006
[21] R.S. Bhatia, M. Kodialam, T.V. Lakshman and S. Sengupta, "Bandwidth Guaranteed Routing With Fast Restoration Against Link and Node Failures," Networking, IEEE/ACM Transactions on , vol.16, no.6, pp.1321-1330, Dec. 2008
[22] Juniper Networks web site, www.juniper.net, 2010.
[23] Cisco Systems web site, www.cisco.com, 2010.
[24] HP networking products and solutions web site, http://h17007.www1.hp.com/us/en/, 2010.
[25] K. Psounis, A. Ghosh, B. Prabhakar, and G. Wang. „SIFT: a Simple Algorithm for Trucking Elephant Flows and Taking Advantage of Power Laws," Proceedings of the 43rd Allerton Conference on Communication, Control, and Computing, Urbana-Champain, Illinois, USA, September 2005
[26] C. Guang, G. Jian, "Online identifying elephant flows through a scalable non-uniform sampling algorithm", Proceedings of ICCT 2008. 11th IEEE International Conference on Communication Technology, 10-12 Nov. 2008.
[27] M. Zadnik, M. Canini, A. W. Moore, D. J. Miller, W. Li, "Tracking Elephant Flows in Internet Backbone Traffic with an FPGA-based Cache," Proceedings of the 19th International Conference on Field Programmable Logic and Applications, Prague 2009.
[1] [28] V J. Ribeiro, R. H. Riedi, R. G. Baraniuk, J. Navratil, L. Cottrell "pathChirp: Efficient Available Bandwidth Estimation for Network Paths," Proc. Passive and Active Measurement Conference, La Jolla, CA, Apr. 2003
[29] S. Suthaharan, S. Kumar, "Measuring Available Bandwidth: pathChirp's Chirp Train Structure Remodeled," Telecommunication Networks and Applications Conference, 2008. ATNAC 2008. Australasian, 7-10 Dec. 2008.
[30] R. Prasad, C. Dovrolis, M. Murray, K. Claffy, "Bandwidth estimation: metrics, measurement techniques, and tools," Network, IEEE , vol.17, no.6, pp. 27-35, Nov.-Dec. 2003.