By,

Harsh Kininge – G00543747

Naveen Paidipalli – G00456650

Raghavendra Parsi – G00455771

Abstract

ReSource Reservation Protocol (RSVP) indicates that evry node needs to send a sporadic control messages to maintain active RSVP sessions. These sporadic messages increase lineraly with number of RSVP sessions. To reduce this overhead, we can make use of longer refresh intervals. But these lead to larger delay in re-synchronizing RSVP state. In our paper, we introduce a new state compression approach where one can address the above mentioned issue. This is by introducing a periodic digest message to each neighbour node which has a compressed version of entire RSVP state. We also enhance the RSVP with an aknowledgement mechanism through which we can the avoid message losses

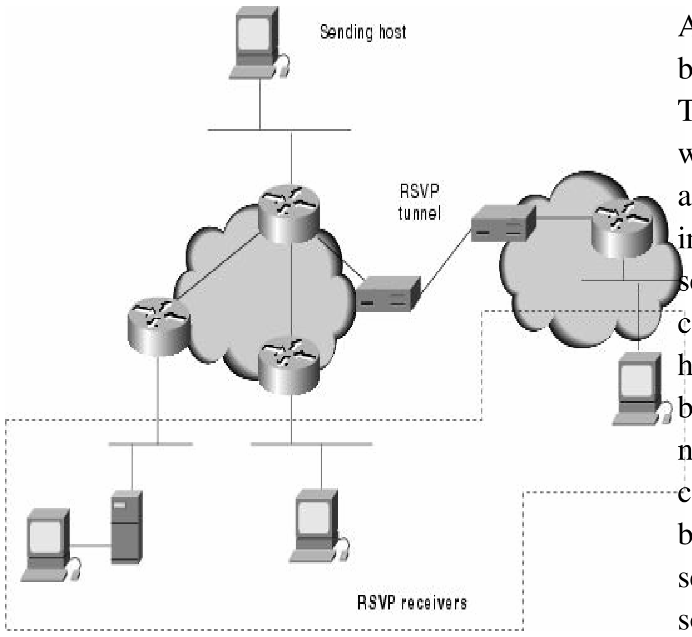
Introduction to RSVP

Network-control protocols that allow internet applications to acquire different qualities of services for their crucial data flow. One such protocol is Resource Reservation Protocol. Each application in a network has various network performance requirements. RSVP is basically developed to provide IP networks with capacity to support varied performance requirements of different applications. It is crucial to know that RSVP is not an routing protocol but RSVP works in compliance with routing protocols and installs the equivalent of dynamic access lists along the routes that routing protocols calculate.

RSVP is used by hosts or routers to request or deliver specific levels of quality of service (QoS) for the application data streams. RSVP states the protocol for how applications place reservations and how they can handover the reserved resources once the requirement for them has ended. RSVP functionality will generally result in resources being reserved in each node along a path.

RSVP is designed to make use of the robustness of current Internet routing algorithms. RSVP does not perform routing on its own, instead it uses other routing protocols to determine where it should carry reservations requests. As routing changes paths to adapt to topology changes, RSVP adapts its reservation to the new paths wherever reservations are in place. This modularity does not rule out it from using other routing services. Current research within the RSVP project is focusing on RSVP to use routing services that provide alternate paths and fixed paths.

Diagrammatic Representation



Quality-of-Service (QoS) offered by RSVP:[4]

All Internet Protocol (IP)- based wireless networks, Base stations(BSs) connect radio system to an IP radio access network(RAN). These Base stations use the IP protocols for data transport and signaling in either Wireless Local Area Networks (WLANs) or fourth-generation networks (4G). But we see that their coverage areas may be arranged in any arbitrary topology. There has been tremendous growth in the accessing of Internet through mobile hosts through wireless networks over the past decade. These mobile hosts basically include mobile phones, laptops, PDA's etc…Due to this substantial growth it's been a big challenge for the servers to deliver the same Quality-Of-Service (QoS) to these mobile users as done in the case of the fixed hosts.

A communication network forms the backbone of any successful organization. These networks transport many applications which include various magnitudes of data and its respective classification. These also include some high quality video and delay sensitive data such as in the case of real time conversation between two mobile or fixed hosts which require Quality-of-Service. The bandwidth intensive applications stretch network capabilities and resources, but also complement, add value, and enhance every business process. Networks must provide secure, predictable, measurable, and sometimes guaranteed services. Achieving this required Quality-of Service by properly managing the delay factor, delay variation, bandwidth, and packet loss parameters on a network becomes the secret to successful end-to-end business solution. In order to achieve such important tasks the Quality of Service plays a vital role in the Communication. In a way, the Quality-of-Service in exact words can be referred to provide different priority to different applications, users, or data flows or to guarantee a certain level of performance which involves a flow of a data.

Today Internet has revolutionized in the fields of business, entertainment, education and many other aspects. Specifically, when we consider the commercial world which uses the Internet and Web related technologies to establish Intranets and Extranets that help streamline business processes and new business models. The increasing popularity of IP and have shifted the paradigm from "IP over everything," to "everything over IP." But if we take look all the way, the internet did not guarantee any kind of service that is considered to be the "best effort service". There are many reasons why this was not merely possible as there are several factors which affect the packet transfer from one host to another. Some of the factors include propagation delay, queuing delay, dropping of some packets when routers reach the maximum size. All these have considerable effect on the quality of the communication like for instance a real-time telephonic conversation over a network. Other factors include corruption due to bit errors and the noise in the transport channel. In order to avoid all these error and provide a quality communication the Quality of Service (QoS) has been introduced which we discuss further below.

Factors affecting Quality-of-Service (QoS):[5]

The Quality of Service plays a very crucial role in the delay sensitive voice communication and bandwidth intense video communication, especially when the network resources are limited and the best effort service fails to work. A network that agrees to provide a certain level of QoS is in contract or agreement in providing that level of performance known as SLA (Service Level Agreement). When we particularly discuss the various crucial factors that affect the QoS, they can be classified into two major categories:

- Human Factors: Stability of Service, Availability of Service, Various kinds of delays and User information.
- Technical Factors: Reliability, Scalability, Effectiveness, Maintainability, Grade of Service, etc.

Below we describe some of the main problems which affect the QoS, considering two hosts where we suppose one host to be a Sender and the other being the receiver. So, there is a considerable scope leading to these problems when a packet of information travels from Sender to the receiver:

- *Dropped Packets:* The routers sometimes might fail to deliver (drop) some packets if they arrive when the buffers are already full or buffers reaches the maximum size. Some, none or all of the packets night be dropped due to this reason, depending on the state of the network and sometimes it's hardly possible to determine this in advance. The receiving application may ask for retransmitting this information possibly causing severe delays in overall transmission.
- *Delay:* This basically occurs when the packet holds up in long queues or when the packet takes a less direct route to avoid congestion. The affect of this would be the delay in packet reaching the destination. Also, in some cases, excessive delay can render an application, such as VoIP, unusable.

- *Jitter:* Packets from the source will reach the destination with different delays. A packet's delay varies with its position in the queues of the routers along the path between the source and the destination and this position can vary unpredictably. This variation in delay is called Jitter, which seriously affects the quality of streaming (Audio/Video).
- *Out-of-Order delivery:* When a collection of related packets is routed through the Internet, different packets tend to take different directions or routes, each resulting in a different delay. The result is that the packets arrive in a different order than they were sent. This problem necessitates special additional protocols responsible for re-arranging out-of-order packets to an isochronous state once they reach their destination. This is especially important for video and VoIP streams where the Quality is severely affected by both latency and lack of isochronicity.

When a collection of related packets is routed through the Internet, different packets may take different routes, each resulting in a different delay. The result is that the packets arrive in a different order than they were sent. This problem necessitates special additional protocols responsible for rearranging out-of-order packets to an isochronous state once they reach their destination. This is especially important for video and VoIP streams where quality is dramatically affected by both latency and lack of isochronicity.

- *Error:* Sometimes packets are misdirected, or combined together, or corrupted in the process of their travel from the source to the destination. This should be detected by the receiver and ask the sender to resend the information.

Functionality of RSVP [4,5]

RSVP is a soft-state protocol, RSVP reserves with a limited lifetime for the reservation state. The end points of RSVP data flows maintain reservation by sending sporadic refresh messages along the data paths, once the life time expire, the session state is automatically deleted. Hence the network has no more orphaned reservations. The sporadic refresh messages help in assuring the correct protocol operations. Following are the important functions of refresh messages:

- Modifying routes: Change in routes cause data flows to switch to different paths. By method RSVP refresh messages follow the data paths, thus the first RSVP messages along the new paths will establish requested reservations, while the state along the old paths is either explicitly cut down or timed out.

- Continuous refresh messages repairs the state inconsistencies: Since RSVP messages are sent as IP datagrams which may be lost on the way. RSVP state can change due to unexpected causes such as undetected bit errors etc. These factors can cause momentary inconsistency in RSVP state along the data paths. Sporadic refreshes serve as a simple repair technique that would verify all state inconsistencies in RSVP state for active sessions.

- Simple to change value in refresh message: When sender or receiver wants to modify its reservation parameters during the session, it simply changes to the modified parameter values in refresh messages.

Related Performance Issues:

- *Augmented Protocol Overhead*: As the number of active RSVP sessions increases the protocol overhead grows linearly. Even in the absence of latest control information produced by source and destination as RSVP node sends to its neighbor one message per active sender session pair per refresh period.

- *Teardown Delay:* This delay is caused losses in RSVP control messages. Although sporadic RSVP refreshes eventually recover any previous loss, the recovery delay which is directly proptional to the refresh period can be unacceptable in a number of circumstances.

Operations and Terminologies used in RSVP:

As we know RSVP is a receiver driven protocol, hence to provide receiver driven functionality, a data source sends PATH messages towards the receivers, leaving a trace of path state at each router that was visited. Receivers that request for the reservation send RESV messages that follow the path state traces upstream towards the data source reserving resources at each intermediate node in the path. The states that are set up by PATH and RESV messages is called path and reservation states. These states are deleted if no matching refresh message occur before expiration. The state may also be deleted by using PathTear or ResvTear messages. PATH and RESV messages do not change, when a route changes, the next path message will initialize the PATH state on a new route and RESV messages will establish reservation states there, the states on the now unused segment of the route will time out. Thus whether a message is new or a refresh is determined at each node depending on existence of state at that node. Before moving on with the topic let us discuss with some of the basic definitions used later in our paper:

- *RSVP state*: A RSVP reservation or path state.
- *Regular/RAW RSVP messages*: Consisting of RESV, PATH, PathTear and ResvTear messages.
- *MD5 Signatures*: The result of computation of MD5 algorithm
- *Digest*: Set of MD5 signatures that represent a compressed version of RSVP.

As we know RSVP is a receiver driven protocol, hence to provide receiver driven functionality, a data source sends PATH messages towards the receivers, leaving a trace of path state at each router that was visited. Receivers that request for the reservation send RESV messages that follow the path state traces upstream towards the data source reserving resources at each intermediate node in the path. The states that are set up by PATH and RESV messages are called path and reservation states. These states are deleted if no matching refresh message occurs before expiration. The state may also be deleted by using PathTear or ResvTear messages. PATH and RESV messages do not change, when a route changes, the next path message will initialize the PATH state on a new route and RESV messages will establish reservation states there, the states on the now unused segement of the route will time out. Thus whether a message is new or a refresh is determined at each node depending on existence of state at that node.

RSVP: Data Flow

RSVP is designed to manage flows of data rather than make decisions for each individual datagram. Data flows consist of discrete sessions between specific source and destination machines. A session is more specifically defined as a simplex flow of datagrams to a particular destination and transport layer protocol. Thus, sessions are identified by the following data: destination address, protocol ID, and destination port. RSVP supports both unicast and multicast simplex sessions.

RSVP: Soft State Implementation[1]

In the context of an RSVP-enabled network, a soft state refers to a state in routers and end nodes that can be updated by certain RSVP messages. The soft state characteristic permits an RSVP network to support dynamic group membership changes and adapt to changes in routing. In general, the soft state is maintained by an RSVP-based network to enable the network to change states without consultation with end points. This contrast with a circuit-switch architecture, in which an endpoint places a call and, in the event of a failure, places a new call.

RSVP protocol mechanisms provide a general facility for creating and maintaining a distributed reservation state across a mesh of multicast and uncast delivery paths.

To maintain a reservation state, RSVP tracks a soft state in router and host nodes. The RSVP soft state is created and must be periodically refreshed by path and reservation-request messages. If no matching refresh messages arrive before the expiration of a cleanup timeout interval, the state is deleted. The soft state also can be deleted as the result of an explicit teardown message. RSVP periodically scans the soft state to build and forward path and reservation-request refresh messages to succeeding hops. When a route changes, the next path message initializes the path state on the new route. Future reservation-request messages establish a reservation state. The state on the now-unused segment is timed out. (The RSVP specification requires initiation of new reservations through the network 2 seconds after a topology change.) When state changes occur, RSVP propagates those changes from end to end within an RSVP network without delay. If the received state differs from the stored state, the stored state is updated. If the result modifies the refresh messages to be generated, refresh messages are generated and forwarded immediately.

Solution to the above defined problem:

To resolve the problem between protocol overhead and responsiveness we present a new approach to RSVP overhead reduction. The importance of our scheme is to replace all the refresh messages sent between two neighboring nodes for each of the RSVP sessions with the digest message that contains a compressed snap-shot of all the shared RSVP sessions between two neighbors. When RSVP nodes for each of the RSVP node receive a digest from a neighbor node, it compares the value carried in the digest message with the value computed from local RSVP state. If 2 values agree node refreshes all the corresponding local state otherwise node starts a state re-synchronization process to discover and repair the inconsistency. To assure quick state synchronization in face of packet losses we also enhance RSVP with an acknowledgement option so that instead of waiting for next refresh, any lost RSVP message can be quickly transmitted. The goal of our proposal is to improve RSVP's scalability allowing efficient operation with large number of sessions. More specifically, we aim at reducing the number of refresh messages while still preserving the soft-state paradigm of RSVP. As in current method we send a refresh message per sender-session pair to neighbor, our approach is to allow each RSVP node send a digest message which is a compact way of representing all the RSVP session state that is shared between two neighboring nodes. In this way, number of sessions are directly proportional to the number of neighboring nodes. These RSVP messages are sent either when triggered by state changes that are detected to re-synchronize the state shared between two nodes. As we infer, these benefits can come only with the overhead. Generally the protocol overhead can be categorized into bandwidth overhead for message transmission and computation overhead for processing these messages.

As we have to compress RSVP state into a digest, we have to concatenate the state of all the RSVP sessions into a long byte stream and compute digest. However this technique suffers from a high overhead of re-computing the whole digest again whenever any change happens. To scale the digest computation we compute the digest in a structured way. First, we hash all the RSVP sessions into a table of fixed size. We then compute the signature of each session as an outcome, and for each slot in the hash table we compute the slot signature from the signatures of all the sessions hashed to that slot. On top of this set of signatures, we build an N-ary tree to compute the final digest

We used tree structure to compute the digest because:

- Whenever the digests computed at two neighboring nodes differ, the two nodes can efficiently locate the portion of inconsistent state by walking down the digest tree.
- When an RSVP session state is added/deleted/modified, an RSVP node only needs to update the signatures along one specific branch of the digest tree, i.e. the branch with the leaf node where the changed session resides.

MD5 Algorithm:[2]

MD5 is a Message Digest Algorithm developed by Ron Rivest at MIT. MD5 is secure version, though little slower, compared to his previous algorithm MD4. The MD5 has been widely used secure hash algorithm particularly in Internet-Standard message authentication. The algorithm takes as input a message of arbitrary length and produces as output a 128-bit message digest of the input. This is mainly intended in the applications where digital signatures are involved. This involves compressing, where a large file is compressed in a secure manner before being encrypted with a private key and a public key. In our design, we use the MD5 algorithm to compute state signatures. We will briefly describe about MD5 in this particular section. MD5 (Message-Digest algorithm 5) is used widely, as a partially insecure cryptographic hash function with a 128-bit hash value. As an Internet standard , MD5 has been employed in a wide variety of security applications, and is also commonly used to check the integrity of files. An MD5 hash is typically expressed as a 32 digit hexadecimal number. MD5 digests have been widely used in the software world to provide some assurance that a transferred file has arrived intact. MD5 is widely used to store passwords. To mitigate against the vulnerabilities mentioned above, one can add a salt to the passwords before hashing them. Some implementations may apply the hashing function more than once—see key strengthening. MD5 processes a variable-length message into a fixed-length output of 128 bits. The input message is broken up into chunks of 512-bit blocks (sixteen 32-bit little endian integers); the message is padded so that its length is divisible by 512. The padding works as follows: first a single bit, 1, is appended to the end of the message. This is followed by as many zeros as are required to bring the length of the message up to 64 bits fewer than a multiple of 512. The remaining bits are filled up with a 64-bit integer representing the length of the original message, in bits.

The main MD5 algorithm operates on a 128-bit state, divided into four 32-bit words, denoted A, B, C and D. These are initialized to certain fixed constants. The main algorithm then operates on each 512-bit message block in turn, each block modifying the state. The processing of a message block consists of four similar stages, termed rounds; each round is composed of 16 similar operations based on a non-linear function F, modular addition, and left rotation. There are four possible functions F; a different one is used in each round:

$F(X,Y,Z) = (X \wedge Y) \vee (\sim X \wedge Z)$

$G(X,Y,Z) = (X \wedge Z) \vee (Y \wedge \sim Z)$

$H(X,Y,Z) = X \oplus Y \oplus Z$

$I(X,Y,Z) = Y \oplus (X \vee \sim Z)$

Where $\oplus$, ^, V, ~ denote, AND, OR and NOT operations respectively

We can say with a high level of assurance that no two sets of different RSVP states will result in the same signatures. However it should be noted that our state compression scheme can work well with any hash function that has a low collision probability such as CRC-32 as long as two neighbor nodes agree upon their choice of the hash function. As another optimization, we also add an acknowledgment option (ACK) to the RSVP protocol. The ACK is used to minimize the re-synchronization delay after an explicit state change request. A node can request an ACK for each RSVP message that carries state-change information, and promptly retransmit the message until an acknowledgment is received. It is important to note the difference between a soft-state protocol with ACKs and a hard-state protocol. A hard-state protocol relies solely on reliable message transmission to assume synchronized state between entities. A soft-state protocol, on the other hand, uses ACKs simply to assure quick delivery of messages; it relies on periodic refreshes to correct any potential state inconsistency that may occur even when messages are reliably delivered, for example state inconsistency due to undetected bit errors, or due to undetected state changes.

State Organization

Refresh efficiency comes with a price. Due to the need of storing the neighbor states because separate digests which need to be sent to different neighbor. Consequently computation costs are inflated since we have to compute the per neighbor digests and we have to operate on per neighbor data structure.

Neighbor Data Structure Design

Current RSVP structure the state inside a node as a common pool of sessions, without considering their destinations. Whereas the digest messages sent towards a particular neighbor contain a compressed version of the state shared with the neighbor. Hence the need arises to further organize RSVP state inside a node according to the neighbors each session that comes from or goes to. To counter this session what we do is introduce a new data structure design called neighbor data structure design. It will hold all the information needed to calculate, receive and send digests to and from a specific node. It will be a combination of all RSVP sessions that the current node sends to or receives from a particular neighbor. To increase efficiency neighbor data structures may not actually store the sessions but contain pointers to the common pool of sessions. In this method a session shared with many neighbors is not copied many times to the neighbor structures. The neighbor structure contains the digest computed from the sessions shared with neighbor and even with auxiliary information such as retransmissions and clean up timers. A node needs to compute two digests for each neighbor, one for the state refreshed by messages received from that neighbor and one for the state the local node is responsible for refreshing towards that neighbor. We call these two digests InDigest and OutDigest is sent in place of raw refreshes while InDigest is used for comparison when the receiving a Digest message from that neighbor.

Session State to Signature:

To compress a session state into a signature, we first need to identify which session parameters need to be constantly synchronized between neighbors. The state representation in this implementation is based on the design is significantly modified and broken down into a more fine-grained layout. All state is stored as objects containing relationships to other objects. The main entry point into the state representation is given by Session objects, which are stored in a global, hash-based container. Starting from a Session object, the full state for each session can be traversed to access specific state blocks. Some of the RSVP message type fields are provided below:

- Path
- Reservation-request
- Path-error
- Reservation-request error
- Path-teardown
- Reservation-teardown
- Reservation-request acknowledgment

A session is uniquely identified by a session object which contains the IP destination address, protocol ID and optionally a destination port number of the associated data packets. A Path State Block (PSB) is comprised of a sender template (i.e. IP address and port number of the sender), and a Tspec that describes the sender's traffic characteristics and possibly objects for policy control and advertisements. A Reservation State Block (RSB) contains filterspecs (i.e. sender templates) of the senders for which the reservation is intended, the reservation style and a flowspec that quantifies the reservation. It may also contain objects for policy control and confirmation. Although PSBs and RSBs contain some other fields such as incoming interface and outgoing interfaces, these fields have only local meaning to a specific node and therefore should be excluded from the digest computation. As to RSVP objects defined in the future, the digest computation can also be applied to them if necessary.

A brief Description of Mechanism: Normal Operation

Neighboring nodes start by exchanging regular RSVP messages as usual. Once a node discovers a compression capable neighbor, it creates a digest for the part of its RSVP state that it shares with each of this neighbors. Subsequently, the node sends Digest messages instead of raw RSVP refreshes at regular refresh intervals. When an event that changes the RSVP state (e.g. a sender changes its traffic characteristics (Tspec)) occurs, raw RSVP messages are sent immediately to propagate this change.

Raw RSVP messages are sent as before, with the added option of asking for an ACK. A sender requesting an acknowledgment, includes in the message a timestamp object with the ACK Requested flag turned on. The sender also sets a retransmission timer for the packet sent. Processing at the receiver side includes updating the digest of the session that the message belongs to as well as updating the digest tree. If during processing a condition occurs that requires sending back an errormessage back to the sender (e.g a ResvErr) then the receiver sends back to the sender that error message. This error message will cancel any pending retransmissions of the original message.

If no ACK is received before the retransmission timer expires, the sender retransmits the message up to a configured number of times. Each of the retransmissions carries the same timestamp contained in the original message. If an updated message (i.e. a PATH message from the same sender but with different Tspec) is sent before an ACK is received, the original message becomes obsolete and no longer needs to be retransmitted. If no ACK arrives even after the message has been retransmitted for the maximum number of times, the message is purged from the node’s list of pending messages. Any inconsistencies created by the possible loss of this message will be later resolved by digests.

Digest messages are always sent with the ACK Requested flag turned on. Digest messages are also retransmitted for a maximum number of times in the absence of ACK messages. However, following the original RSVP design where an RSVP node never stops sending refresh messages for each active session, a node should not stop sending digest refreshes even if it fails to receive an acknowledgment in the previous refresh interval. If the neighbor node crashed and becomes alive again, it will find the digest value different from its own and the two routers will start the re-synchronization process. When the digest value is changed, the node needs to cancel any pending retransmission of the obsolete Digest message and promptly send a Digest message with the new digest value. When a node receives a Digest message, it checks to see if the state reported by the Digest message is consistent with the corresponding state stored locally. To do so the node does a binary comparison between each of the MD5 signatures contained in the digest object and the corresponding MD5 signatures in the InDigest. If all of them agree then the state is consistent and an ACK is sent back. Otherwise the receiver returns a DigestErr message containing its

InDigest and the process described in the next section begins.

Procedure for Recovery

There are many reasons for two RSVP neighbors going out of sync. For instance, a state-changing RSVP message got lost and the sender did not ask for acknowledgement. It may also happen that a node crashed and lost part or all of its state. This process can be made easier if the inconsistencies are detected when they arise and immediately does the recovery process or does the repair. As we mentioned earlier, a node sends a ‘DigestErr’ message if the received digest value disagrees with the local digest. The timestamp and digest value of it help the two neighbors localize the problem. If the timestamp acknowledged is smaller than the timestamp of the last Digest message sent, this error message is for an obsolete message. This message should be ignored since it may not represent the current state of the neighbor. If they are equal, the node starts a depth-first search of the mismatching signatures from the root of the digest tree. When a node receives a ‘DigestErr’ message it compares the digest value with its own to find the states that are inconsistent. When it finds the first mismatching signature (call it S1), it sends a Digest message containing the signatures used to compute S1. A DigestErr is expected for this Digest message since at least one of the children signatures should not match. The node again looks for the first mismatching signature (S2) in the second DigestErr message and sends the children of S2 in a Digest message. This procedure is repeated until the leaf signature (Sh) causing the problem is found. Now, the node knows that one or more of the sessions in that hash table slot (represented by Sh) must be inconsistent with those in the neighbor. It can then locate these sessions by further exchanging the session signatures with the receiver. It is observed that there is a tradeoff between the latency of the recovery procedure and the transmission efficiency. For example, if the tree has many levels, many RTTs are needed to exchange the digests at all the tree levels in order to find the leaf-level sessions that contribute to the inconsistency. However, if speed of convergence is more important than efficiency, one can stop at an intermediate tree level and refresh all the states represented by the mismatching signature at that level.

Time Arguments involved:

In this part we deal with the two parameters of time, associated with digest messages, mentioned below:

- The refresh period between successive digest refreshes (R).
- The retransmission timeout (T).

The implementation part starts with a node sending digests at intervals of r (r, randomly chosen in the range [0:5*R; 1:5*R]). This is done so to avoid the synchronization of the digest messages. The retransmission time (T) is basically used when an acknowledgement is not received after a time interval of T, and node will be held up in the work of retransmitting that digest message. To be consistent, the digest refreshes are also sent every thirty seconds by default and this interval should be made configurable. The digest messages are explicitly acknowledged and there would be no necessity to decrease the value of 'R' to avoid lost digest messages, but smaller values of 'R' would be efficient, as it affects frequency, in the environments where prolonged periods of inconsistency are undesirable. Also, the retransmission time (T) should be proportional to the round-trip-time (RTT) between any two connected neighbors.

Summary:

Our focus in this paper was to mainly bring two new changes to the implementation of the RSVP. We have let each node compress aggregate RSVP state to a digest that can be efficiently carried in a single packet, which can further be exchanged between neighbor RSVP nodes. The digest computation using the MD5 algorithm is done in a structured way so that, any inconsistency between two neighbors can be easily identified and located for repair. In doing so, this state compression drastically reduces the message overhead of the RSVP. Secondly, we focus on the enhancement of RSVP providing it with an acknowledgement option. In doing so, the signal can be quickly re-transmitted if necessary, when the sender receives an acknowledgement from the receiver. Although, reliable delivery of control messages cannot be used to replace soft-state refreshes, use of acknowledgement definitely speeds up the state synchronization in the case of message losses. We are also planning to explore the feasibility of this approach to some other soft-state protocols, in achieving the same kind of efficiency by using a soft-state approach with state compression instead of a hard-state protocol such as Transmission Control Protocol (TCP).Lastly, one of our future objectives is to calculate the computational costs in determining how expensive the operation of inserting or deleting nodes would be, applying the MD5 algorithmic analysis. This would require us some more time.

References:

[1] Refreshing Mechanism in RSVP, An adaptive and dynamic Timer Design to maintain soft state in RSVP, http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=934897

[2] Information about MD5 algorithm, FPGA implementation of MD5 Hash Algorithm, http://www.engr.mun.ca/~howard/PAPERS/ccece_2001.pdf

[4] RSVP and QoS, Extending RSVP for Quality of Secure Service, http://portal.acm.org/citation.cfm?id=1128677

[5] QoS mechanisms in all IP Wireless Access Networks,

http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1303757