Baltzer Journals

A Routing Architecture for Mobile Integrated Services Networks

Shree Murthy
Sun Microsystems, Inc.
901 San Antonio Rd.
Palo Alto, CA 94303
Email: shree@eng.sun.com
Phone: (650) 688-9449

J.J. Garcia-Luna-Aceves
University of California
Computer Engineering
Santa Cruz, CA 95064
jj@cse.ucsc.edu
(408) 459-4153

A drawback of the conventional Internet routing architecture is that its route computation and packet forwarding mechanisms are poorly integrated with congestion control mechanisms. Any datagram offered to the network is accepted; routers forward packets on a best-effort basis and react to congestion only after the network resources have already been wasted. A number of proposals improve on this to support multimedia applications; a promising example is the Integrated Services Packet Network (ISPN) architecture. However, these proposals are oriented to networks with fairly static topologies and rely on the same conventional Internet routing protocols to operate. This paper presents a routing architecture for mobile integrated services networks in which network nodes (routers) can move constantly while providing end-to-end performance guarantees. In the proposed connectionless routing architecture, packets are individually routed towards their destinations on a hop by hop basis. A packet intended for a given destination is allowed to enter the network if and only if there is at least one path of routers with enough resources to ensure its delivery within a finite time. Once a packet is accepted into the network, it is delivered to its destination, unless resource failures prevent it. Each router reserves resources for each active destination, rather than for each source-destination session, and forwards a received packet along one of multiple loop-free paths towards the destination. The resources and available paths for each destination are updated to adapt to congestion and topology changes. This mechanism *could* be extended to aggregate dissimilar flows as well.

Keywords: multipath routing, congestion control, quality of service, performance guarantee

This work was supported in part by the Defense Advanced Research Projects Agency (DARPA) under contracts DAAB07-95-C-D157 and F19628-96-C-0038.

Report Documentation Page	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE **1998**	2. REPORT TYPE	3. DATES COVERED **00-00-1998 to 00-00-1998**
4. TITLE AND SUBTITLE **A Routing Architecture for Mobile Integrated Services Networks**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of California at Santa Cruz,Department of Computer Engineering,Santa Cruz,CA,95064**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT **Approved for public release; distribution unlimited**		
13. SUPPLEMENTARY NOTES		
14. ABSTRACT		
15. SUBJECT TERMS		

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**32**	

1 – Introduction

Supporting multimedia applications requires the provision of performance guarantees or quality of service (QoS) which are commonly expressed in terms of bounds to end-to-end delay, throughput, and delay variations or jitter. All of today's popular approaches for supporting quality of service in IP or ATM internetworks are based on establishing connections [7, 4, 3]. However, supporting real-time multimedia applications in a wireless mobile infrastructure in which all nodes can be mobile and where nodal connectivity changes constantly is difficult to accomplish, because the constituency of the paths supporting connections from sources to destinations cannot be guaranteed. Furthermore, current proposals for the provision of QoS in the Internet rely on conventional Internet routing protocols, which are not well suited for the provision of QoS. First, they react to congestion *after* the network resources have been wasted, i.e., a source reacts to congestion only after it receives a congestion indication from the congested resource in the data path before the source can regulate its input rate into the network. Second, they are based on single-path routing algorithms. Even in theory, a routing protocol based on single-path routing is ill-suited to cope with congestion. This is because the only thing a single path routing protocol can do to react to congestion is to change the route used to reach a destination. As has been documented by Bertsekas [1], allowing a single-path routing algorithm to react to congestion can lead to unstable oscillatory behavior.

This paper introduces a new routing architecture for mobile integrated services networks that provide performance guarantees while the network topology can change continuously. This work was motivated by the conjecture that architectural elements similar to those used in a connection-oriented architecture (e.g., the Integrated Services Packet Network (ISPN) architecture [4]) to allow the network to enforce performance guarantees can be used in a *connectionless* routing architecture to integrate routing and congestion control and to provide performance guarantees for the delivery of those datagrams accepted in the network.

In our connectionless routing architecture, packets are individually routed towards their destination on a hop-by-hop basis. A packet intended for a destination is allowed to enter the network if and only if there is at least one path with enough resources available to ensure its delivery within a finite time. In contrast to the existing connectionless routing schemes, once a packet is accepted into the network, it is delivered to its destination, unless a resource failure prevents it. Packet routing is done on a per-destination basis rather than a per-session basis as in conventional connection-oriented networks. The resources available at each router are updated to adapt to congestion and the dynamic state of the network.

Although OSPF can provide multiple paths, they have to be of equal length.

This approach is based on the following three architectural elements:

- Traffic shaping by means of destination-oriented permit buckets
- Traffic separation and scheduling on a per destination basis
- Maintenance of dynamic multiple loop-free paths to reduce the delay from source to destination.

Generalized processor sharing (GPS) is an ideal scheduling mechanism which ensures fairness among all the competing flows and also provides traffic isolation among these flows. i.e., one misbehaved flow will not affect the characteristics of other well-behaved flows. Packet-by-packet generalized processor sharing (PGPS) [14] is the packetized version of GPS which achieves the abovementioned properties. PGPS provides fairness among competing flows and also provides flow isolation. In this paper, we assume a PGPS server at each router for scheduling of packet transmissions; more practical scheduling approaches can be integrated in the future. Routing is done on a per destination basis over multiple paths. To establish multiple loop-free paths, we extend prior results on loop-free, single-path routing algorithms introduced in [11, 16, 10]. This results in a congestion-oriented multipath routing architecture that uses a short-term metric based on hop-by-hop credits to reduce congestion over a given link, and a long-term metric based on end-to-end path delay to reduce delay from source to destination.

Section 2 describes the new connectionless routing architecture and introduces the concept of *flow multiplexing* to support various QoS parameters. Section 3 describes the operation of the routing protocol for the new connectionless architecture in detail. Section 4 gives the formal proof of correctness of both scheduling and credit-based mechanisms. Section 5 derives a worst-case delay bound as a function of the available credits. Section 6 discusses the extension of the protocol to hierarchical topologies. Finally, Section 7 presents our conclusions.

2 – Two-Tier Routing Architecture

The network is modeled as an undirected finite graph represented by $G : (N, E)$, where N is the set of nodes and E is the set of edges or links connecting nodes. A bidirectional link connecting nodes i and j is represented as (i, j) and is assigned a positive weight in each direction. A link is assumed to exist in both directions at the same time. All routing messages that are received (transmitted) by a node are put in the input (output) queue on a *FCFS* basis and are processed in that order. Each node is represented by a unique identifier and the link costs can vary in time but are always positive. A *path* from node i to node j is a sequence of nodes, $i, n_1, ..., n_r, j$, where (i, n_1), (n_x, n_{x+1}), (n_r, j) are links in the path. A *simple path* from i to j is a sequence of nodes in which no node is visited more than once. A *multipath* from i to j is a set of simple paths from i to j. The paths between any pair of nodes and their corresponding distances change over time in a dynamic network. At any point in time, node i is connected to node j if a physical path exists from i to j at that time. The network is said to be connected if every pair of operational nodes are connected at a given time.

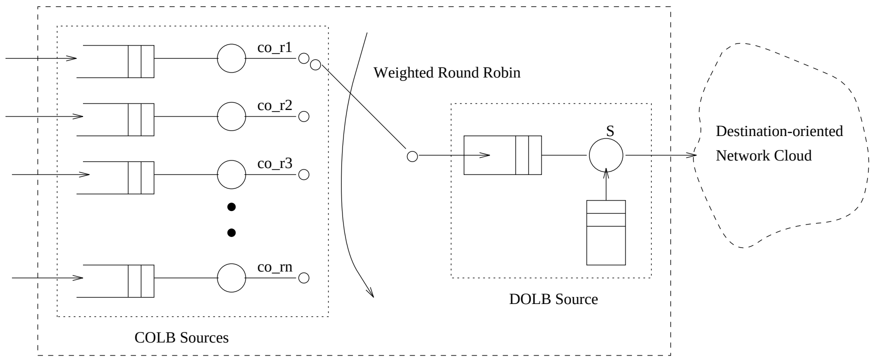


Figure 1: Source Model

In a destination-oriented network, all nodes in the network can be potential sources. Each source is capable of supporting connection-oriented sessions. Figure 1 shows the model at each source. Each connection-oriented sessions are regulated by a leaky-bucket filter, referred to as *connection oriented leaky bucket* (COLB). A connection-oriented session generates traffic at a rate co_r_i, where $i = 1, ..., N_s$. At any time, there can exist at most N_s sessions in parallel. Each session is identified by a source-destination pair and the type of service that is being requested.

Each connection-oriented session is mapped into a destination-oriented flow. In destination-oriented flows, packets are serviced on a per-destination basis instead of a per-session basis as in a connection-oriented architecture. A *destination-oriented leaky bucket* (DOLB) is maintained depending on the type of service and destination. All COLBs addressed to the same destination and having the same service requirements are grouped together. The way in which COLB flows are fed to the DOLB depends on the relative rates of the associated COLB flows.

To ensure fairness among different connections, a *weighted round robin* scheduling mechanism among all COLB flows belonging to the same group is used at each of the DOLB inputs. Our protocol guarantees that all packets which enter the connectionless network (destination-oriented network) will be delivered to their respective destinations, unless a resource failure prevents it. Depending on the available resources at each hop, the traffic entering the network is regulated at the COLB stage itself, before the packets actually enter the connectionless network.

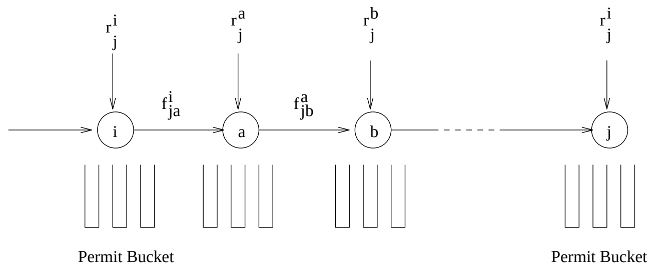


Figure 2: Node Model

In order to regulate the traffic flow from COLBs to DOLB, a feedback mechanism is required from DOLB to COLB, so that the leaky bucket parameters of COLB can be regulated.

Let N_s be the number of connection-oriented sessions associated with a DOLB source s with traffic generation rates of co_r_1, co_r_2,..., $co\_r_{N_s}$, respectively. Each of these sessions is controlled by leaky-bucket parameters (σ_i, ρ_i), where σ_i is the buffer size and ρ_i is the token generation rate at source i.

Let CR_j^S be the total available credits (or tokens) at source S for destination j in the destination-oriented network. The token generation at each of the COLBs is a function of the credits available at source S and this determines how many packets can be sent through the destination-oriented network. When there are N active sources the token generation rate at each COLB is given by

$$\rho_i = \frac{co\_r_i}{\sum_{k=1}^{N_s} co\_r_k} \, CR_j^S \tag{1}$$

When a new COLB source becomes active for the first time, a request is sent to the destination asking for more credit allocation for that source. Such a request is associated with a sequence number. When a response is received for that credit request, the obtained credits are distributed to the requesting source. If the destination is not able to handle any more requests from the sources due to resource limitations, credits are not sent back for such explicit requests and such sources will not be allowed into the destination-oriented network cloud.

This mechanism ensures that sources are well behaved and DOLB does not accept more data than it can handle, and thus can guarantee delivery of packets once the packets enter the connectionless network.

A connection-oriented flow is recognized by a source, destination and a type which specifies the QoS associated with that flow. Flows having the same specification ({source, destination, type}) are grouped together. i.e., all COLB sources having the same flow specification are grouped together and mapped to a single DOLB source. Destination based credits are allocated to these DOLB sources and the packets are routed to destination using a credit-based multipath routing algorithm. This ensures that all sources with the same flow characteristics are guaranteed similar service. At the destination, packets are demultiplexed to their respective COLB destinations.

A model of a packet traversal path in the destination-oriented cloud is shown in the Figure 2. In the figure, r^i_j is the traffic originated at node i for destination j and f^i_{ja} is the incoming traffic as seen by a for destination j on link (i, a). Routing is done on a hop-by-hop basis independently at each node. Scheduling at a node is done by maintaining permit bucket filters at each node for all active destinations. Because of hop-by-hop routing mechanism at each node along the path, any node along the path from a source to a destination can be a potential source contributing to the traffic flow towards that destination. Therefore, the traffic at a node corresponds to the traffic arriving at a node from its upstream neighbors (f^i_{ja}) and also the traffic originated at the node itself (r^i_j). This will be discussed further in the following sections. A weighted fair queueing mechanism is used to ensure that all traffic streams receive a fair amount of resources [6].

Each flow type has a separate destination-oriented permit bucket for data transfer. The QoS required by the application can be divided into different classes and each class is serviced separately, independent of other flows, to ensure flow isolation.

3 – Routing Protocol

A multipath protocol based on congestion information is used within the packet-switched network to guarantee that all packets incident on the connectionless network will be delivered to their respective destinations. The protocol consists of three functional units, namely: *packet scheduling and transmission*, *congestion based credit mechanism* and *maintenance of multiple loop-free paths*. Packet forwarding to the destinations are based on two routing metrics: a *short-term metric* based on *hop-by-hop credits* to reduce congestion along a link and a *long-term metric* based on *path-delay* to minimize the end-to-end delay.

Packet scheduling is done by means of permit buckets for each destination. Packets are transmitted as individual entities at each hop. The path a packet can take to reach a destination is determined independently at each hop. Traffic at each node is regulated by permit buckets, independently for each destination.Traffic scheduling and credit aggregation is done at each hop, independently for each destination. This is required since in a packet-switched network, each router along the path from a source to a destination can be a potential source and contribute to the data flow to that destination. The number of available credits gives an estimate of the bandwidth available for each destination. Available credits are aggregated and sent towards the source along the reverse paths implied by the routing table entries. When a node becomes operational, depending on the availability of resources, credits are distributed among its neighbors.

3.1 – Message Types

Messages are exchanged among routers for the proper functioning of the protocol. We classify the messages into six types.

Explicit Credit Requests: When a new COLB source becomes active at a router, a message is sent towards the destination of that flow asking for some kind of reservation or guarantee for that flow in terms of credit availability for the transmission of data packets. Explicit credit requests are associated with a sequence number. The identifier of the COLB source that initiated an explicit credit request and the corresponding sequence number is recorded at the time of credit request. (These messages are analogous to *PATH* messages in RSVP [3]).

Explicit Credit Response: This message is sent in response to an explicit credit request message and indicates the credits available for data transmission and the sequence number of the request. The available credits are then distributed among the requested COLB sources at the source of the credit request. (These messages are analogous to *RESV* messages of RSVP).

Explicit Teardown: This message is sent to reclaim all the credits assigned to a particular flow. This message is initiated by a source on recognizing that there are no more packets associated with a flow that needs to be delivered to the destination. Credits reserved for a flow are reclaimed at each hop along the way to a destination. (These are similar to *TEARDOWN* messages of RSVP).

Fast Reservation Packet: When a node sends an explicit credit request, its neighbor along the path to the destination responds with a fast reservation packet indicating that the source can use a few credits before the actual credit allocation is made by the destination. Each source may then start transmitting data without having to wait till it receives credits from the destination, thus preventing starvation. The fast reservation credits are reclaimed when the credits from the destinations are assigned to the flow.

Periodic Credit Updates: These are simple update messages which carry credit information (the rate at which data packets are sent). Credits are updated periodically to maintain correct network state information. Periodic updates do not contain a sequence number. Credit information is exchanged among neighboring nodes periodically and credits are redis-

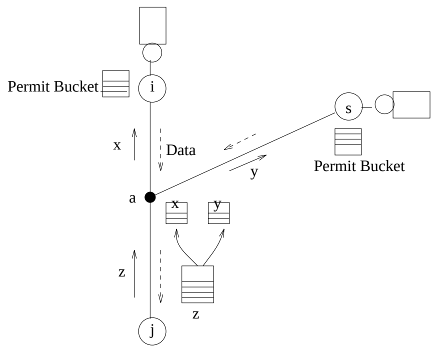


Figure 3: Credit Distribution

tributed among active flows at each node accordingly. A credit update which is sent in response to an explicit credit request will be taken into account in the periodic update after the explicit credit response was sent.

Routing Table Update: Whenever the information present in the routing table changes, a routing table update is exchanged between neighbors. These updates are also not associated with sequence numbers. Routing table updates contain both cost and credit information. The way in which the routing table entries are updated depends on the underlying routing algorithm.

3.2 – Packet Scheduling and Transmission

In a connectionless architecture, the route that a packet takes to reach a destination is determined independently at each hop. All nodes along any path from source to destination can contribute to the flow. This requires traffic scheduling to be done at each hop to regulate the incoming traffic, instead of having intermediate nodes as just forwarding nodes, as in a connection-oriented architecture [15]. We use a PGPS server to regulate the incoming traffic.

As mentioned before, the protocol uses two routing metrics, a short-term metric and a long-term metric to transmit packets to a given destination. The number of packets sent to a neighbor depends on the credits available through that neighbor or equivalently, the number of credits available in the permit buckets. Credits for a flow are sent from a destination towards the source along the reverse paths implied by the routing tables. When a node becomes operational, depending on the availability of resources at each node, credits are distributed among its neighboring nodes. The credit based mechanism is explained in the next section.

The traffic at each node is regulated by permit buckets, independently for each destination. In the traditional leaky bucket congestion-control scheme, buckets are session oriented; i.e., credits are assigned on a per session basis. Data packets are accepted from the source and the average rate of flow is controlled by a burst rate for a source-destination session. In our scheme, permit buckets (which are similar to leaky buckets) are destination oriented; i.e., at every router permit buckets are maintained for each active destination. A destination is said to be active at a router if the router is contained in a feasible path to the destination. For a given destination j, credits reach a node i at a rate ρ^i_j, which is called the *token* or *credit generation rate* for destination j at node i. The bucket size, denoted by σ^i_j gives the maximum number of packets that can be transmitted from i to j (burstiness) at time t, and σ^i_j is defined for each destination j at time $t \geq 0$ as

$$\sigma^i_j(t) = l^i_j(t) + Q^i_j(t) \tag{2}$$

where $Q^i_j(t)$ is the session backlog (depth of the queue) for destination j at time t that is, the packets which were left behind from the previous time interval and $l^i_j(t)$ is the number of credits (or tokens) in the bucket at node i for destination j at time t that can be used by packets in the current time interval. This definition is much the same given in [14], the only difference being that here we maintain leaky-bucket parameters for each active destination rather than for each session.

Credits sent by downstream nodes are aggregated at each hop for a given destination along the path from destination to source and are redistributed among its upstream neighbors. In Figure 3, if z is the number of credits received by node a from its downstream neighbors to destination j, then node a maintains a permit bucket of size z. These credits are redistributed among its upstream neighbors relative to the traffic flow along links (i, a) and (s, a) as x and y.

The number of credits left behind, is the difference in the number of credits that arrive within a given time interval and the number of arrivals. Accordingly, $[K^i_j(t) - K^i_j(\tau)] - A^i_j(\tau, t)$, where $K^i_j(t)$ is the credits that are accepted by node i at time t for destination j and $A^i_j(\tau, t)$, the traffic arriving at node i for destination j in the interval $(\tau, t]$, is the number of arrivals in units of credit. The total number of accepted credits in a time period should be less than the credit generation rate. Therefore, with $\tau \leq t$,

$$K^i_j(t) - K^i_j(\tau) \leq \rho^i_j \quad (t - \tau) \tag{3}$$

ρ^i_j is related to the number of total credits available to node i at time t and is the sum of the credits available through all the nodes downstream of node i. Consider Figure 4. The total number of credits available at router i for destination j is the sum of the credits available from its downstream neighbors a, b, and c for destination j (some neighboring nodes may not belong to the feasible set of nodes to a destination. In the above example, nodes d and e does not belong to FM^i_j) The total number of packets transmitted to destination j from node i cannot

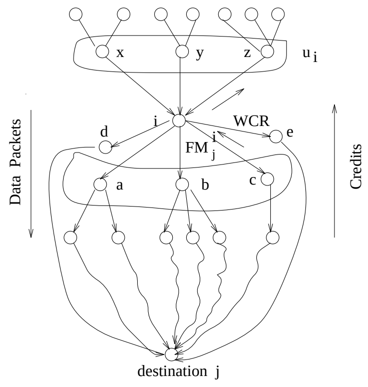


Figure 4: Multipath Graph

exceed the total available credits at i for j at time t. The number of available credits also depends on the traffic flow on that link which is an indication of the congestion state of that link as explained in the following section.

3.3 – Credit-Based Congestion Mechanism

Congestion over a given link is avoided by a hop-by-hop credit-based mechanism. Each router selects a path to the destination based on the available bandwidth through a given link, utilization of that link and the cost to reach the destination through that path. The chosen path is subject to a constraint that the bandwidth available through that path is at least equal to the required bandwidth, or equivalently that the total bandwidth allocated through a link is less than the capacity of that link. The available bandwidth is then translated into credits. Credits given by a node to its upstream neighbors for a given destination represent the number of packets that the node can accept from its upstream neighbors for that destination. Credits to upstream nodes are sent in the reverse direction of the routing tree. Upstream nodes receive credits from downstream nodes for a given destination before they can transmit data towards the downstream nodes.

Figure 5 presents a formal description of the credit allocation scheme. Procedure *Initialize* indicates the action taken by a node when it becomes active for the first time (initialization). Procedure *Receive* describes the functions performed by a node when a node receives a periodic update (steady state).

Initialization and Path selection

On initialization, credits are equally distributed among neighbors as there will be no traffic on any of the links of a new node. The number of credits available at each node i for destination j is determined by the total resources available at that node ($MaxCredits^i_j$). A part of the total available credits ($Reserve^i_j$) is reserved for our fast reservation mechanism. The fast reservation mechanism is used to allocate credits to a new source when an explicit credit request is received for a destination. This mechanism speeds up the credit allocation process by sending a minimum number of credits to the new upstream neighbor as explained below. Since there are always some reserved credits at each node, nodes upstream do not starve. The reserved credits are reclaimed when the traffic starts flowing through that path. A flow uses the reserved credits only for one unit of time. Therefore, there is no need to reserve credits explicitly for all destinations. The remaining credits, CR^i_j ($= MaxCredits^i_j - Reserve^i_j$), are equally distributed among the neighboring nodes, N_i, on startup. This is the *weighted credit* WCR^i_{jk} where i is the source, j is the destination and k is the neighbor. The hop-by-hop delay incurred for the credits to reach next-hop neighbor is taken into account while computing the available credits (explained in the following section).

Credits are dynamically assigned among all the active flows, depending on the traffic flow through each of the links. When node i selects neighbor k as one of its multiple successors to a destination, it sets the successor flag in the update message ($s\_flag^i_{jk}$) to that neighbor to indicate that the neighbor now belongs to feasible multipath set FM^i_j. When node k recognizes this, it includes the node in its set of active upstream neighbors, sends a fixed minimum amount of credit (CR_{min}) to its new neighbor indicating that i can use k as a possible multipath successor, and redistributes its credits for that destination. This is done in order to indicate that the upstream neighbor can use this node for sending packets towards destination j if required. Credit redistributions information is communicated to other nodes in the next update interval. The total credits sent to the upstream neighbor is limited by the total available credits at that node. Credit information at each node is updated periodically; this is required because we do not use a reliable credit updating mechanism.

A routing node resets its traffic counters on initialization and monitors the incoming and outgoing traffic for all its neighbors. Based on these statistics, the *routing parameters* ϕ^i_{jk}, from i to j through k for all neighbors $k \in N_i$ are computed. The permit bucket parameters are also initialized for each destination. The credit generation rate ρ^i_j is initialized to the sum of the credits available through all the neighbors of i to a given destination j in the given time period. The permit bucket size σ^i_j is initialized to the number of available credits.

Steady State

Each node monitors the traffic flowing through its incoming and outgoing links periodically and determines the traffic flow on each of its links for all destinations. A periodic update timer is maintained at each router to exchange credit information periodically. The periodic update interval Δt should be longer than the maximum round trip time (RTT) delay between two nodes in the network. Each time an update is sent, the periodic timer, R^i_j, is reset (Figure 5).

At each node, credits received from all downstream nodes are aggregated and

```
Variables:
    $p^i_{jl}$:    credits occupied by packets in transit
                   from $i$ to $j$ through $l$
    $q^i_j$:       credits due to packets already in queue

Procedure Initialize
when router $i$ initializes itself
begin
        /* some credits are reserved for fast reservation */
        $CR^i_j \leftarrow MaxCredits^i_j - Reserve^i_j$;
        do for $k \in N_i$
        begin
            $WCR^i_{jk} \leftarrow \frac{CR^i_j}{|N_i|}$;
            $s\_flag^i_{jk} \leftarrow 1$;
        end
        Send credit information $WCR^i_{jk}$ to all $k \in N_i$
        at next update interval
        Reset $R^i_j$
end

Procedure Receive($i$)
when $i$ receives periodic update from $m$
begin
        if (($s\_flag^m_{ji} = 1$) $\wedge$ ($m \notin FM^i_j$))
            /* add $m$ to $FM^i_j$ */
            $CR^i_j \leftarrow CR^i_j + CR_{min}$;
            $FM^i_j \leftarrow FM^i_j \cup \{m\}$;
        if (($s\_flag^m_{ji} = 0$) $\wedge$ ($m \in FM^i_j$))
            /* delete $m$ from $FM^i_j$ */
            $CR^i_j \leftarrow CR^i_j - WCR^m_{ji}$;
            $FM^i_j \leftarrow FM^i_j - \{m\}$;
        /* redistribute credits among shortest path
        neighbors */
        $CR^{i'}_j \leftarrow \sum_{m \in FM^i_j} WCR^i_{jm} - p^i_{jm}$;
        $CR^i_j \leftarrow CR^{i'}_j - q^i_j$;
        $WCR^i_{jm} \leftarrow CR^i_j \times \phi^i_{jm} \mid i \in FM^m_j$;
        Send credit information to all $k \in N_i$
        at next update interval
        Reset $R^i_j$
end
```

Figure 5: Credit Distribution Mechanism

are redistributed to the upstream neighbors. This can be done because the total bandwidth allocated at each link at any given time is no more than the capacity of that link. A node can send data to a downstream neighbor only if the credit value through that neighbor is greater than zero. Also, because at each hop credits are distributed based on the traffic flow, the algorithm ensures that information about all active destinations are maintained, When a new destination for which the bandwidth is not reserved becomes active, or when a node becomes part of the set of loop-free paths to a destination, credits are redistributed using a fast reservation mechanism.

Explicit Credit Request and Response

When a new source becomes active and the source needs credits to transfer data, an explicit credit request is sent to all neighbors in the feasible multipath set. This elicits a credit response. The initial credit assignment is done using fast-reservation mechanism. The explicit credit response from each neighbor indicates the amount of credits allocated by the destination through multiple paths for the requested flow.

Explicit Teardown

Each teardown message is associated with a sequence number representing the flow. Although it is not necessary to explicitly reclaim the credits of a flow, it is recommended that all sources send explicit teardown message as soon as a session is completed.

A teardown message is initiated by a source and is forwarded hop-by-hop all the way to the destination. The state of each node is updated along the way and the credits are reclaimed. These messages are not delivered reliably. The loss of a teardown message will not cause a protocol failure because the unused credits will eventually time out and are reclaimed.

One-Pass Reservation Mechanism

To ensure that there is no starvation in the protocol, a fast reservation mechanism is used. This enables the downstream nodes to respond immediately without having to wait till they hear from their destinations about the credit availability. This scheme is similar to the one-pass reservation mechanism of RSVP [3].

Sources send explicit credit requests to a destination requesting credits for data transmission. On receiving these requests, credits are allocated for that session and source; all the nodes along the way are informed about this.

Each node maintains soft-state information about the network state. This state information is refreshed periodically by credit updates. If a credit update is not received before a certain time interval, those credits are reclaimed. At each periodic timeout interval, the state of the network is updated. The soft-state information has the potential to change at every periodic time interval.

Periodic Updates

Credit (resource reservation) information is updated periodically. The allocated resources are reclaimed after a certain time if the state information is not refreshed. The timer value for updating the state information is set at each hop independently. Floyd and Jacobson [8] have shown that periodic updates generated by independent network nodes can become synchronized. This can lead to disruption in network services as the periodic messages contend with other network traffic for link and forwarding resources. Therefore, periodic credit update messages must avoid synchronization and ensure that any synchronization that may occur is not stable.

Because of this, the refresh timer should be randomly set to a value in the range $[0.5R^i_j, 1.5R^i_j]$ where R^i_j is the periodic update timer used to generate updates. The value R^i_j is chosen locally at each node. A smaller R^i_j speeds up the adaptation to network state changes but increases the protocol overhead. A node may therefore adjust the effective R^i_j dynamically to control the amount of overhead due to periodic update messages. The default value of R^i_j is set to 30 seconds as in some of the other routing protocols.

The end-to-end delay associated with packets to each destination is also estimated periodically. If the measured delay does not satisfy the required QoS, that successor will no longer be selected as a feasible successor to the given destination and this information is communicated to all the neighboring nodes. A node determines the total available credits for a given destination j, and the credits are redistributed among its upstream neighbors after reserving a fraction of the credits for the initialization phase. The philosophy behind this fast reservation mechanism is similar to a fast bandwidth reservation scheme.

Figure 6 shows the cost table (DT) at node i for destination j for the config-

Destination	Neighbor	Flag	Cost	Credits
j	a	1	a1	a2
j	b	1	b1	b2
j	c	1	c1	c2
j	d	0	d1	0
j	e	0	e1	0
j	x	0	x1	infinity
j	y	0	y1	infinity
j	z	0	z1	infinity

Figure 6: Cost Table at node i for destination j

uration in Figure 4. The flag field indicates whether the neighbor belongs to the feasible multipath set or not. The distance gives the sum of the link costs along the path to destination j and credits gives the number of available credits through that path. A credit of 0 implies that packets cannot be forwarded through that path.

The number of credits available at a node is determined by the flow on its links and the total traffic seen by that node. If f^k_{ji} is the incoming flow on link (k, i) to destination j as seen by node i, and r^i_j is the traffic originated at i for destination j, we define the total input traffic seen by i for destination j as the sum of all the incoming traffic at node i, and denote it by λ^i_j. Furthermore, by the conservation of flow, the sum of all the traffic arriving at a node must be equal to the sum of all the traffic departing from a node for each destination j. Therefore, for destination j, the total incoming flow is equal to the total outgoing flow at node i, and

$$\lambda^i_j = \sum_{k \mid i \in FM^k_j} [f^k_{ji}] + r^i_j = \sum_{m \in FM^i_j} f^i_{jm} \tag{4}$$

Note: $i \in FM^k_j$ does not necessarily mean $k \in FM^i_j$

The routing parameter, ϕ^i_{jk} is defined for each link (i, k) as the ratio of the flow on each link with respect to the total flow on all outgoing links for a given destination j. From Eq. 4 and with N_i denoting the neighbor set of i, we have:

$$\phi^i_{jk} = \frac{f^i_{jk}}{\lambda^i_j} \quad \forall k \in N_i \tag{5}$$

For $k \notin FM^i_j$, $f^i_{jk} = 0$. In other words, if a neighbor is not in the feasible multipath set of node i, flows do not go through that neighbor. Because node i itself can also contribute to the total traffic, by the conservation of flow, it must be true that

$$\sum_{k \in FM_j^i} \phi_{jk}^i \leq 1 \tag{6}$$

The rate of distribution of credits to upstream neighbors depends on the traffic flow on that link, which in turn depends on the routing variables ϕ_{jk}^i associated with that link. The number of credits a node sends to an upstream neighbor is weighted by the traffic flow on a given link. The token generation rate for a given update period Δt is proportional to the weighted credit received by that node for destination j through all its feasible multipath neighbors. Weighted credit is proportional to the routing variable ϕ of that link.

$$\rho_j^i(\Delta t) = \sum_{k=1}^{FM_j^i} WCR_{jk}^i(\Delta t) \tag{7}$$

To obtain a correct estimate of the credits generated at each node at any given time, we need to take into account the delay associated with the propagation of credits. This can be done either by estimating the credits available as in [12] or by explicitly sending a marker. We use a estimation mechanism. Credits are sent to the upstream neighbor, i.e., they propagate only one hop. The update period used for updating routing information is one round-trip delay by a data packet. Therefore, to obtain a correct estimate of the available credits at a node, we have to take into account the data packets that a sender has already forwarded over the link in the past round-trip time (RTT) and the data packets that are already queued from the past RTT. Therefore, the total available credits at node i for a destination j, is the difference between the sum of all the weighted credits available from its downstream neighbors l_i (equivalently, sum of all the credits on its outgoing links) belonging to the feasible multipath and the credits which are already being used, i.e.,

$$CR_j^i(t) = \sum_{l \in FM_j^i} [WCR_{jl}^i(t) - p_{jl}^i(t)] - q_j^i(t) \tag{8}$$

where WCR_{jl}^i is the weighted credit obtained from the downstream neighbor l over an update period (which depends on the flow on the link (i,l)), p_{jl}^i is the number of credits occupied by the packets that are already in transit on link (i,l), and q_j^i is the number of credits due to the data packets that are already in the queue at node i for destination j which were not completely transmitted since the previous update period. If a node does not have credits for a given destination j, then CR_j^i is set to zero.

3.4 – Maintenance of Loop-Free Multipaths

The primary objective of maintaining multiple loop-free paths is to minimize the end-to-end path delay by reducing network congestion along the path. The cost reported to node i for destination j by a downstream neighbor k is denoted by D^i_{jk} (this is the distance between k and j) and node i's cost to its neighbor k is denoted by d_{ik} (cost of the link (i,k)). The cost from node i to destination j through node k is denoted by $\breve{D}^i_{jk}$, and is equal to, $\breve{D}^i_{jk} = D^i_{jk} + d_{ik}$.

The delay at node i to destination j is computed as the weighted average path delay through all the nodes in the feasible multipath at node i; it is denoted by D^i_j. This delay is weighted by the fraction of the traffic going through that path, i.e.,

$$D^i_j(t) = \sum_{k \in FM^i_j(t)} \phi^i_{jk}(t) \quad \breve{D}^i_{jk}(t) = \sum_{k \in FM^i_j(t)} \frac{f^i_{jk}(t)}{\lambda^i_j(t)} \quad \breve{D}^i_{jk}(t) \tag{9}$$

The flow from i to each neighbor in FM^i_j depends on the credits available through that neighbor. Assuming that packets are of fixed size and that each packet corresponds to one credit, we can say that a packet flows on a link if at least one credit is available on that link. That is, packet flow on a link implies credits are available on that link. This implies that the number of packets that can flow on a link is equal to the number of credits available through that link; for simplicity we assume all packets are of the same size. Therefore,

$$D^i_j(t) = \frac{1}{\lambda^i_j(t)} \sum_{k \in FM^i_j(t)} [WCR^i_{jk}(t) \quad \breve{D}^i_{jk}(t)] \tag{10}$$

Multiple loop-free paths from each node to a destination are maintained by means of a feasible multipath routing algorithm (FMRA), which is based on our prior work on path-finding algorithms [11, 16, 10]. Any change in cost trigger update messages. A marker tag^i_j is used to update the routing table. For a given destination j, tag^i_j specifies whether the entry corresponds to a simple path ($tag^i_j = correct$), a loop ($tag^i_j = error$) or a destination that has not been marked ($tag^i_j = null$).

An update message from router i consists of a vector of entries; each entry specifies a destination j, an update flag u^i_j, a successor flag $s\_flag^i_{jk}$, the reported cost to that destination RD^i_j, the reported predecessor to the destination rp^i_j, and the reported credits available to destination j through that neighbor. The update flag indicates whether the entry is an update ($u^i_j = 0$), a query ($u^i_j = 1$) or a reply to a query ($u^i_j = 2$). The predecessor is the second-to-last hop in the best path to the destination. Using predecessor information, we can implicitly derive a loop-free path to a destination.

A detailed specification of FMRA is given in Figures 7 and 8. Procedures *RT_Init* and *DT_Init* are used for routing table and cost table initialization. Procedure *Message* is executed when a router processes an update message, a query or a reply; procedures *Link_Up*, *Link_Down* and *Link_Change* are executed when a router detects a new link, link failure or a change in the link cost, respectively. We refer to these as the event handling procedures. An update, query or a reply is handled by procedures *Update*, *Query* and *Reply*, respectively. Procedure *Update_Timer* updates the periodic update timer and updates the credit information. Procedures *DT_Update* and *RT_Update* updates the cost and the routing tables, respectively; details of how these procedures work are provided in [16, 10] and omitted here for brevity.

For any destination j, a router i can be in one of the two states, active or passive, at any given time. Node i is active for destination j if it does not have a feasible successor to destination j and is waiting for at least one reply from a neighbor; node i is passive otherwise. A router i initializes itself in the passive state with an infinite cost to all its known neighbors and a zero cost to itself. The maximum allowable cost to reach neighbor, defined below, is also set to ∞. Routers send updates containing cost and credit information about themselves to all their neighbors. When the destinations become operational, they inform their neighbors about the available credits to all other destinations.

Each routing update contains cost and credit information. Cost and credit information are exchanged among neighbors when the state of the network changes; credit information is also updated periodically. An update can contain the full routing table or increments of the routing table in different update messages. After initialization, only incremental updates are sent.

For a given destination, a router updates its routing table differently depending on whether it is *active* or *passive* state for that destination. When router i is passive, it reports the current value of D^i_j in all its updates and replies. An active router cannot send an update message for the destination for which it is active. However, when a active router receives a query, it sends a reply with infinite distance.

When a passive router receives a routing update for a destination it tries to obtain a *feasible successor* for that destination. A feasible successor is obtained using a feasible multipath condition which states at time t, a router i can make node $k \in N_i(t)$ as a feasible successor if and only if $D^i_{jk}(t) < MAD^i_j(t)$ where MAD^i_j is the *maximum allowable cost* for destination j, and is equal to the minimum value obtained for D^i_j since the last time router i transitioned from active to passive state for destination j. From router i's standpoint, a feasible successor

```
Variables:
  $N$:       Set of all nodes
  $N_i$:     Set of all neighbors of node $i$
  $p^i_j$:   Predecessor of node $j$ from $i$
  $p^i_{jx}$: Predecessor of node $j$ from
             $i$ through $x$
  $FCSET$: Set of feasible successors
  $s^i_j$:   Successor of node $i$ to $j$
  $s^i_{jx}$: Successor of $i$ to $j$ through $x$
Procedure RT_Init
when router $i$ initializes itself
do begin
    set a link-state table with
    costs of adjacent links;
    $N \leftarrow \{i\}; N_i \leftarrow \{x \mid d_{ix} < \infty\}$;
    for each $(x \in N_i)$
    do begin
        $N \leftarrow N \cup x$; $tag^i_x \leftarrow$ null;
        $s^i_x \leftarrow$ null; $p^i_x \leftarrow$ null;
        $D^i_x \leftarrow \infty$; $MAD^i_x \leftarrow \infty$
    end
    $s^i_i \leftarrow i$; $p^i_i \leftarrow i$; $tag^i_i \leftarrow$ correct;
    $D^i_i \leftarrow 0$; $MAD^i_i \leftarrow 0$;
    for each $j \in N$ call DT_Init($x$, $j$);
    for each $(n \in N_i)$ do
    Send message (0, $i$, 0, $i$) to $n$;
end

Procedure DT_Init($x$, $j$)
begin
    $D^i_{jx} \leftarrow \infty$; $p^i_{jx} \leftarrow null$;
    $s^i_{jx} \leftarrow null$; $rf^i_{jx} \leftarrow 0$;
end

Procedure Reply($j$, $k$)
begin
    $rf^i_{jk} \leftarrow 0$;
    if $(rf^i_{jn} = 0, \forall n \in N_i)$
    then if $((\exists x \in N_i \mid D^i_{jx} < \infty)$
      or $(D^i_j < \infty))$
        then call Passive_Update($j$)
    else call Active_Update($j$, $k$)
end

Procedure Update($j$, $k$)
begin
    if $(rf^i_{jx} = 0, \forall x \in N_i)$
    then begin
        if $((s^i_j = k)$ or $(D^i_{jk} < D^i_j))$
        then call Passive_Update($j$)
    end
    else call Active_Update($j$, $k$)
end
Procedure Message
when router $i$ receives a message
  on link $(i, k)$
begin
    for each entry $(u^k_j, j, RD^k_j, rp^k_j)$
      such that $j \neq i$
    do begin
        if $(j \notin N)$
        then begin
            if $(RD^k_j = \infty)$
            then delete entry
            else begin
                $N \leftarrow N \cup \{j\}$;
                $MAD^i_j = \infty$;
                for each $x \in N_i$
                   call DT_Init($x$, $j$)
                $tag^i_j \leftarrow null$;
                call DT_Update($j, k$)
            end
        end
        else
            $tag^i_j \leftarrow null$;
            call DT_Update($j, k$)
    end
    for each entry $(u^k_j, j, RD^k_j, rp^k_j)$
    left such that $j \neq i$
    do case of value of $u^i_j$
        0: [Entry is an update]
           call Update($j$, $k$)
        1: [Entry is a query]
           call Query($j$, $k$)
        2: [Entry is a reply]
           call Reply($j$, $k$)
    end
    call Send
end

Procedure Passive_Update($j$)
begin
    $DT_{min} \leftarrow Min\{D^i_{jx}\ \forall\ x \in N_i\}$;
    $FCSET \leftarrow \{n \mid n \in N_i,\ D^i_{jn} = DT_{min},$
      $D^n_j < MAD^i_j\}$;
    if $(FCSET \neq \emptyset)$ then begin
      call RT_Update($j$, $DT_{min}$);
      $MAD^i_j \leftarrow Min\{D^i_j, MAD^i_j\}$
    end
    else begin
       $MAD^i_j = \infty$; $rf^i_{jx} = 1, \forall x \in N_i$;
       $D^i_j = D^i_{j\,s^i_j}$;
       $p^i_j = p^i_{j\,s^i_j}$;
       if $(D^i_j = \infty)$ then $s^i_j \leftarrow$ null;
       $\forall$ x $\in N_i$ do begin
         if (query and x = k)
         then $rf^i_{jk} \leftarrow 0$;
         else Send message
            (1, $j$, $\infty$, null) to $x$
       end
    end
end

Procedure Query($j$, $k$)
begin
    if $(rf^i_{jx} = 0 \forall x \in N_i)$
    then begin
        if $(D^i_j = \infty$ and $D^i_{jk} = \infty)$
        then Send message
           (2, $j$, $D^i_j$, $p^i_j$) to $k$
        else begin
           call Passive_Update($j$);
           Send message (2, $j$, $D^i_j$, $p^i_j$)
              to $k$
        end
    else call Active_Update($j$, $k$)
end
```

Figure 7: FMRA Specification

toward destination j is a neighbor router k that satisfies the maximum allowable cost condition (MACC) given by the following two equations:

$$
\begin{aligned}
D^i_j(t) &= D^i_{jk}(t) + d_{ik}(t) = Min\{D^i_{jp}(t) + d_{ip}(t) \mid p \in N_i(t)\} \\
D^i_{jk}(t) &< MAD^i_j(t)
\end{aligned}
\tag{11}
$$

Router i adjusts MAD^i_j depending on the congestion level of the network. Figure 9 gives a graphical representation of how MAD is updated. The point at which a new computation for finding a feasible successor starts is a *synchronization point* [9]. It can be noted that between two synchronization points the value of MAD can only decrease or remain the same and can never increase. Every time the MAD

```
Procedure Link_Up $(i, k, d_{ik})$
when link $(i, k)$ comes up do begin
    $d_{ik}$ ← cost of new link;
    if $(k \notin N)$ then begin
        $N \leftarrow N \cup \{k\}$; $tag^i_k \leftarrow$ null;
        $D^i_k \leftarrow \infty$; $MAD^i_k \leftarrow \infty$;
        $p^i_k \leftarrow$ null; $s^i_k \leftarrow$ null;
        for each $x \in N_i$ do
            call DT_Init$(x, k)$
    end
    $N_i \leftarrow N_i \cup \{k\}$;
    for each $j \in N$ do
        call DT_Init$(k, j)$;
    for each $j \in N - k \mid D^i_j < \infty$ do
        Send message $(0, j, D^i_j, p^i_j)$ to $k$;
end

Procedure Link_Down$(i, k)$
when link $(i, k)$ fails do begin
    $d_{ik} \leftarrow \infty$;
    for each $j \in N$ do begin
        call DT_Update$(j, k)$;
        if $(k = s^i_j)$ then $tag^i_j \leftarrow null$
    end
    delete column for $k$ in distance table;
    $N_i \leftarrow N_i - \{k\}$;
    delete $rf^i_{jk}$;
    for each $j \in (N - i) \mid k = s^i_j$
        call Update$(j, k)$
end

Procedure Update_Timer:
when the periodic update timer
expires
begin
    reset timer $R^i_j$;
    update credit information;
    for all $i \in N_i$
        send periodic update;
end
```

```
Procedure Link_Change $(i, k, d_{ik})$
when $d_{ik}$ changes value do begin
    $old \leftarrow d_{ik}$;
    $d_{ik}$ ← new link cost;
    for each $j \in N$ do begin
        call DT_Update$(j, k)$;
        for each $j \in N$
        do if $(D^i_j > D^i_{jk}$ or $k = s^i_j)$
        then $tag^i_j \leftarrow$ null;
    end
    for each $j \in N$ do begin
        if $(d_{ik} < old)$
        then for each $j \in N - i$
            $\mid D^i_j > D^i_{jk}$
            do call Update$(j, k)$
        else for each $j \in N - i$
            $\mid k = s^i_j$
            do call Update$(j, k)$
    end
end

Procedure DT_Update$(j, k)$
begin
    $D^i_{jk} \leftarrow RD^k_j + d_{ik}$;
    $p^i_{jk} \leftarrow rp^k_j$;
    for each neighbor $b$ do
    begin
        $h \leftarrow j$;
        while $(h \neq i$ or $k$ or $b)$ do
            $h \leftarrow p^b_h$;
        if $(h = k)$ then begin
            $D^i_{jb} \leftarrow D^i_{kb} + RD^k_j$;
            $p^i_{jb} \leftarrow rp^k_j$;
        end
        if $(h = i)$ then begin
            $D^i_{jb} \leftarrow \infty$;
            $p^i_{jb} \leftarrow null$;
        end
    end
end
```

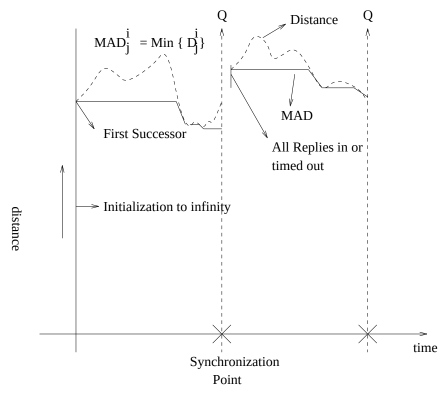
```
Procedure RT_Update$(j, DT_{min})$
begin
    if $(D^i_{j\,s^i_j} = DT_{min})$
    then $ns \leftarrow s^i_j$;
    else $ns \leftarrow b \mid \{b \in N_i$ and
        $D^i_{jb} = DT_{min}\}$;
    $x \leftarrow j$;
    while $(D^i_{x\,ns} = Min\{D^i_{xb}\ \forall\ b \in N_i\}$
      and $(D^i_{x ns} < \infty)$
      and $(tag^i_x = null))$
    do $x \leftarrow p^i_{x\,ns}$;
    if $(p^i_{x\,ns} = i$ or $tag^i_x =$ correct$)$
    then $tag^i_j \leftarrow$ correct;
    else $tag^i_j \leftarrow$ error;
    if $(tag^i_j =$ correct$)$
    then begin
        if $(D^i_j \neq DT_{min}$ or $p^i_j \neq p^i_{j\,ns})$
        then Send message
            $(0, j, DT_{min}, p^i_{j\,ns})$ to
            $x, \forall x \in N_i$
        $D^i_j \leftarrow DT_{min}$; $p^i_j \leftarrow p^i_{j\,ns}$;
        $s^i_j \leftarrow ns$;
    end
    else begin
        if $(D^i_j < \infty)$
        then Send message
            $(0, j, \infty$, null$)$ to
            $x, \forall x \in N_i$;
        $D^i_j \leftarrow \infty$; $p^i_j \leftarrow$ null;
        $s^i_j \leftarrow$ null;
    end
end

Procedure Active_Update$(j, k)$
begin
    if $(k = s^i_j)$ then begin
        $D^i_j \leftarrow D^i_{jk}$; $p^i_j \leftarrow p^i_{jk}$;
    end
end
```

Figure 8: FMRA Specification (cont.)

has to increase, a new synchronization point has to be computed. This ensures that the algorithm is loop-free.

If a passive router i finds a feasible successor, it remains passive for that destination. It sets MAD^i_j equal to the smaller of the updated value of D^i_j and the present value of MAD^i_j. In addition, it updates its cost, predecessor, and successor making sure that only simple paths (paths that are loop-free) are used [16]. Router i then prepares an update message to its neighbors if its routing table entry changes. Alternatively, if router i finds no feasible successor, then it sets $MAD^i_j(t) = \infty$ and updates its cost and predecessor to reflect the information reported by its successor. If $D^i_j(t) = \infty$, then the successor (next-hop node) $s^i_j(t)$ = null. Router i also sets the reply status flag $(rf^i_{jk} = 1)$ for all $k \in N_i$ and sends a query to all its neighbors, which reports an infinite cost to the destination. Router



$MAD^i_j = \text{Min}\{D^i_j\}$
Q
Distance
Q
MAD
First Successor
All Replies in or timed out
Initialization to infinity
distance
time
Synchronization Point

Figure 9: Maximum Allowable Cost Condition

i is then said to be *active*, and cannot change its successor information until it receives all the replies to its query. When a neighbor moves, that neighbor is assumed to have failed (a neighbor is said to be no longer existent if a node does not hear from its neighbor for a period of time). All the pending replies from that neighbor are assumed to have been received and it is removed from the neighbor set N_i.

Once active for destination j, router i cannot change its feasible successor, neighbors, or its entry in the routing table, until it receives all the replies to its query. A reply received from a neighbor indicates that such a neighbor has processed the query and has either obtained a feasible successor to the destination, or determined that it cannot reach the destination.

Queries and replies are processed in a manner similar to the processing of an update described above. The input event that causes router i to become active is a query in which router k reports an infinite cost. This is the case, because router k's query, by definition, reports the latest information from router k. Router i will send a reply with finite distance to k if router i has a feasible path to j; else i enters active state and replies to k with an infinite distance. An update is sent to k when i becomes passive again. A link-cost change is treated as a link going down and coming up with a new cost. When router i is active and receives a reply from router k, it updates its distance table and resets the reply flag ($rf^i_{jk} = 0$).

Router i becomes passive at time t when it receives replies from all its neighbors, which means they have processed its query reporting an infinite cost. Therefore, router i is free to choose any neighbor that provides the shortest cost, if there is any. If such a neighbor is found, router i updates the routing table with the minimum cost as described and sets $MAD^i_j(t) = D^i_j(t)$. A router does not wait indefinitely for replies from its neighbors because a router replies to all its queries regardless of its state. Thus there is no possibility of deadlocks due to the interneighbor coordination mechanism.

Ensuring that updates will not be sent in the network when some destination is unreachable is easily done. If node i has set $D^i_j(t) = \infty$ already and receives an input event (a change in cost or status of link $(i,\ k)$, or an update or query from node k) such that $D^i_{jk}(t) + d_{ik}(t) = \infty$, then node i simply updates D^i_{jk}, and sends a reply to node k with the reported distance $RD^i_j(t) = \infty$ if the input event is a query from node k. When an active node i has an infinite maximum allowable cost and receives all the replies to its query such that every neighbor offers an infinite cost to the destination, the node simply becomes passive with an infinite cost.

When node i establishes a link with a neighbor k, it updates the value of d_{ik} and assumes that node k has reported infinite costs to all destinations and has replied to any query for which node i is active. Furthermore, if node k is a previously unknown destination, node i sets $s^i_k = null$, and $D^i_k(t) = RD^i_k(t) = MAD^i_k(t) = \infty$. Node i also sends to its new neighbor k an update for each destination for which it has a finite cost.

When node i is passive and detects that link $(i,\ k)$ has failed, it sets $d_{ik} = \infty$ and $\breve{D}^i_{jk} = \infty$. After that, node i carries out the same steps used for the reception of a link-cost change in the passive state.

Because a router can become active in only one interneighbor coordination mechanism per destination at a time, it can expect at most one reply from each neighbor. This is true because a router cannot send updates for a destination for which it is active. Accordingly, when an active node i loses connectivity with a neighbor n, node i can set $rf^i_{jn} = 0$ and $D^i_{jn} = \infty$, i.e., assume that its neighbor n has sent any required reply reporting an infinite cost. When node i becomes passive again, it must find a neighbor that satisfies the MACC using the value of MAD^i_j set at the time node i became active in the first place (Eq 11).

When nodes choose their successors from the feasible successor set, the path from source to destination obtained as a result of this is loop free at every instant.

4 – Correctness

The proof of correctness and loop-freedom of FMRA is basically the same as that provided in [10] for LPA. In this section we prove that the scheduling policy which we have used to map connection-oriented sessions to destination-based flows and the credit-based mechanism are both correct and live. For this purpose, we assume that all sources are well-behaved.

4.1 – Scheduling Scheme

Lemma 1: Each destination-oriented permit-bucket always demultiplexes the credits fairly among the connection-oriented sessions.
Proof: The proof of this involves two parts. (a) fairness of credit distribution on explicit credit requests (b) fairness of credit distribution on receipt of periodic credit updates

Each DOLB is fed by a set of COLBs. Let N_s be the number of COLBs associated with each DOLB at any given time.
(a) When a new COLB source becomes active, DOLB sends an explicit request for credits towards destination j. Each of these explicit requests is associated with a sequence number.

When such a request is sent to the feasible multipath neighbors, the state of the source along with the sequence number is saved i.e., the COLB source due to which an explicit request was initiated. When a credit response (analogous to an acknowledgment) for that sequence number is received, the credits are fairly distributed among the requesting sources. i.e.,

$$CR_j^i = \frac{co\_r_i}{\sum_{req} r_{req}} CR_j^S$$

where r is the set of requesting sources. Therefore, credit demultiplexing at DOLB is fair. This proves part (a).

(b) At each periodic interval, credits are assigned to neighbors depending on the traffic flow seen through those neighbors. When a periodic credit request is received by a DOLB source, the credits are fairly distributed to all the active COLB sources associated with it. i,e.,

$$CR_j^i = \frac{r_i}{\sum_A r_a} CR_j^S$$

where A is the set of active sources. (In this case, active source set and requested source set will be the same). This proves part (b).

Therefore Lemma 1 is true. □
Lemma 2: The scheduling mechanism is live.
Proof: Whenever a new source becomes active, credits are explicitly requested by that source using a explicit credit request message. Since each of these messages are associated with a sequence number, according to Lemma 1, when the source receives a credit response, credits are assigned proportionally to all requesting sources. In the worst case, a credit request might have to go all the way down to the destination before it gets the desired allocation. Since we maintain loop-free paths to all destinations, the hop-count from source to a destination is finite.

Therefore, the scheduling mechanism is live. This proves Lemma 2. □
Theorem 1: The scheduling mechanism is correct.

Proof: Lemma 1 and 2 ensure that the scheduling mechanism will not have any deadlocks and all the active sources have a fair share of the available credits at all times. This proves theorem 1. □

4.2 – Credit-based Scheme

The correctness of the credit based mechanism (i.e., showing that it has no deadlocks and that packets are not dropped) can be proven similarly as for virtual-circuit connections [13]. For the purposes of such a proof, we make the following assumptions.

- the protocol is initialized properly i.e., all nodes have correct credit and routing information
- there are no link errors or link failures and node failures (steady state)

Lemma 3: The congestion-oriented credit mechanism never drops packets accepted into the network.
Proof: The total credits available at a node is given by (from Eq. 8):

$$CR^i_j(t) \geq \sum_{k \in FM^i_j} [WCR^i_{jk}(t) - p^i_{jk}(t)] - q^i_j(t)$$

Also, the total of the available credits is less than the maximum resources at a node.

$$MaxCredits^i_j(t) \geq CR^i_j(t)$$

This is true since we reserve some credits for the fast reservation mechanism.

At any node, the number of credits available for a given destination j is at least equal to the packets sent downstream for that destination. This is true since a packet can not be forwarded unless we have a credit to do so. If a packet is in flight from a sender to a receiver, then $p^i_{jl} = 0$ and $CR^i_j < MaxCredits^i_j$. Thus, the received packet will not be dropped as there is at least one empty buffer that can be used.

This proves the Lemma. □

Lemma 4: The credit based mechanism is starvation free.
Proof: To prove this Lemma, we consider three cases – (a) initialization (b) when the nodes are operational and (c) when a explicit credit request is received.
Case (a): On initialization, when a new node comes up, the node distributes its credits equally among its neighbors and receives a minimum number of non-zero credits from its neighbors indicating the presence of a possible path to a destination through that neighbor. Since on startup a minimum number of credits is always available to the new node, the credit mechanism is starvation free.

Case (b): When a new node becomes a member of the feasible multipath set (FM_j^i), the successor flag will be set in the update message. On receiving this update, the neighbor first sends a minimum number of credits initially using the reserved bandwidth available through the fast reservation mechanism. Later on the neighbor dynamically updates the credits depending on the traffic on the link.
Case (c): An explicit request for credits from a source is received by means of a routing table update. This update sets the successor flag in the update message. Therefore, the message will be treated as any other routing table update and is processed similar to case (b) if the router has credits to send to its upstream neighbors. Otherwise, the router sends routing table updates to its feasible multipath neighbors (downstream neighbors) requesting for credits to send packets to the destination. In the worst-case, this process can continue till the routing table update reaches the destination and will terminate at the destination. This is true because the credits to a flow are assigned at the destination.

This ensures that a node will always have credits when it requires to forward a packet. Therefore, the protocol is starvation free. $\square$

Theorem 2: In the congestion-based credit mechanism, if there is a packet at a node to be sent, it will be eventually sent (finite time).
Proof: From Lemma 3, packets which are accepted into a network are not dropped unless there is a resource failure (this is true in case of connection-oriented networks also). This means, a packet arriving at a node will either be forwarded or queued till the resources are available to forward it.

From Lemma 4, whenever an intermediate node needs to forward a packet, it always has the credits to do so. This implies packets arriving at a node will be forwarded eventually.

This proves the theorem $\square$

5 – Worst-Case Steady-State Delay

In this section, we derive an upper bound on the end-to-end steady-state path delay, D_j^{i*}, from node i to destination j as a function of the credits available through each path under steady state. Steady-state means that all costs and credit information are correct at every router. This bound demonstrates that it is possible to provide performance guarantees in a connectionless routing architecture. The delay experienced by a packet accepted into the network is the time required by a data packet to reach its destination from a source. This includes both the propagation delays and the queueing delays. Path delay can also be interpreted as the time it would take for a destination j backlog at i to clear.

Parekh and Gallager have analyzed worst-case session delay in a connection-oriented network architecture [15]. We adopt a similar approach for each destination in a connectionless architecture. To do this, we assume a stable topology in which all routers have finite cost paths to each other. We also make use of the fact that FMRA enforces loop-freedom at every instant on all paths in the feasible multipath sets.

In a connectionless network, the path taken by a packet can change dynamically depending on the congestion level in the network. Routing is done on a hop-by-hop basis, independently at each router. The total traffic at a node will be the sum of the traffic on all its links connecting to upstream neighbors. To obtain an expression for the worst-case bound, we make the following assumptions:

1. Each node can send traffic to destination j only if credits are available (non-zero) for that destination along any of its chosen paths.

2. At every node m, traffic for every destination is treated independently.

3. Traffic arriving at a node i for destination j in the interval $(0, t)$ (denoted by A^i_j) is the sum of the traffic from all its upstream neighbors to destination j and the traffic originated at the node i itself, denoted by $r^i_j(t)$, i.e.,

$$A^i_j(t) \quad = \quad r^i_j(t) + \sum_{i \in FM^n_j(t), n \in N_i} f^n_{ji}(t) \tag{12}$$

Each router in the connectionless network can itself be a source to any given destination. At each node, traffic to destination j is constrained by a permit bucket filter. The worst-case delay and backlog is bounded above by an additive scheme due to Cruz [5]. The rate at which the packets are serviced at each node depends on the permit bucket or leaky bucket parameters σ^i_j and ρ^i_j for a given destination j. The parameter σ^i_j gives the permit bucket size, and ρ^i_j gives the credit accrual rate at node i. Therefore, the number of packets that are being serviced at a node is a function of σ^i_j and ρ^i_j.

The per-hop delay on a link (i, k) d^i_{jk} for a given destination j is the sum of the queueing delay and the propagation delay on that link. Propagation delay is defined as the time taken for a packet to reach a destination from a source. Every packet is time-stamped when it leaves a node, and the time at which the packet reaches the neighbor is noted. The difference between the two gives a one-hop delay. The average of this delay over a given period of time gives the propagation delay δ^i_k.

The queueing delay is the time a packet has to wait at a node before it is processed. The waiting time of a packet depends on the number of packets already present in the queue at the time a packet arrives. This is referred to as the backlog at node i for destination j and is denoted by Q^i_j. Therefore, the delay on link (i,l) for destination j at time t is

$$d^i_{jl}(t) = \delta^i_l(t) + Q^i_{jl}(t)\delta^i_l(t) = \delta^i_j(t)[1 + Q^i_{jl}(t)] \tag{13}$$

The backlog number of packets for a given destination j at a given time t can be defined as the difference in the incoming and the outgoing traffic at a node, i.e.,

$$Q^i_j(t) = A^i_j(t) - S^i_j(t) \tag{14}$$

This takes into account both the processing delay and the queueing delay experienced at each hop. For every interval $(\tau, t]$,

Let $\tau < t$ be the time at which there are no backlogged packets in the network. Then,

$$S^i_j(\tau, t) \geq \rho^i_j(t - \tau) \tag{15}$$

We assume that variation in the size of the packet does not contribute significantly to the delay component. A delay bound for the case in which packet size is not negligible is presented by Murthy and Garcia-Luna-Aceves[11]. The total number of arrivals at each node i is the sum of the arrivals at all the upstream nodes for destination j and the traffic originated at node i itself. For all $t \geq \tau \geq 0$ we have,

$$A^i_j(\tau, t) = r^i_j(\tau, t) + \sum_{l | i \in FM^l_j(t)} \phi^i_{jl}(\tau, t)\ A^l_j(\tau, t) \tag{16}$$

The maximum backlog traffic Q^{i*}_j for destination j is the difference between the arrivals in the interval $(\tau, t]$ and the total packets transmitted in the same interval at node i. For $\phi^i_{jl} > 0$,

$$\begin{aligned} Q^{i*}_j(\tau, t) &= A^i_j(\tau, t) - S^i_j(\tau, t) && (17) \\ &\leq r^i_j(\tau, t) - S^i_j(\tau, t) \\ &\quad + \sum_{l | i \in FM^l_j(t)} [\phi^i_{jl}(\tau, t)\ A^l_j(\tau, t)] && (18) \\ &\leq [r^i_j(t) - r^i_j(\tau)] - S^i_j(\tau, t) \\ &\quad + \sum_{l | i \in FM^l_j(t)} [\phi^i_{jl}(\tau, t)\ A^l_j(\tau, t)] && (19) \end{aligned}$$

The difference $(r^i_j(t) - r^i_j(\tau))$ determines the amount of traffic arriving at node i in the interval $(t - \tau)$; the maximum of which is the sum of the tokens available at node i at τ and the tokens received in the interval $(t-\tau)$. At every node, each destination is constrained independently by a permit bucket scheme. Following Parekh and Gallager's approximation [15], we assume the links to be of infinite capacity. The results for the infinite capacity case supply a upper bound for the finite capacity case. The arrival and the service functions at each router depend on the maximum path delay and the link flows. Substituting for the arrivals and the number of packets serviced in terms of the permit bucket parameters from the previous section we have

$$\begin{aligned} Q_j^{i*}(\tau,t) &\leq [\sigma_j^i(t-\tau)+\rho_j^i(t-\tau)] - \rho_j^i(t-\tau) \\ &+ \sum_{l \mid i \in FM_j^l(t)} \frac{f_{jl}^i}{\lambda_j^i}[\sigma_j^i(t-\tau)+\rho_j^i(t-\tau)] \end{aligned}$$

Because $f_{jl}^i \leq \lambda_j^i$ for any j and $l \in FM_j^i(t)$,

$$Q_j^{i*}(\tau,t) \leq \sigma_j^i(t-\tau) + \sum_{l \mid i \in FM_j^l(t)} [\sigma_j^l(t-\tau)+\rho_j^l(t-\tau)] \tag{20}$$

Making $\tau = t - \Delta t$, we can write

$$Q_j^{i*}(t) \leq \sigma_j^i(\Delta t) + \sum_{l \mid i \in FM_j^l(t)} [\sigma_j^l(\Delta t)+\rho_j^l(\Delta t)]$$

Therefore, the backlog at node i to destination j depends on the leaky bucket parameters at node i and the permit bucket parameters of all the upstream neighbors of i for which node i is in the feasible multipath set.

The delay at each node i can be computed as the weighted average path delay through all its multipath neighbors; therefore,

$$D_j^i(t) = \sum_{k \in FM_j^i(t)} \phi_{jk}^i(t)\breve{D}_{jk}^i(t) \tag{21}$$

The cost from i to j through neighbor k can be expressed as the sum of the costs from k to j and the link cost from i to k. The link cost is the sum of the cost and the propagation delay of that link. Therefore,

$$\begin{aligned} \breve{D}_{jk}^i(t) &= D_{jk}^i(t) + d_{ik}(t) \\ D_j^i(t) &= \sum_{k \in FM_j^i(t)} \phi_{jk}^i(t)[D_{jk}^i(t) + d_{ik}(t)] \end{aligned} \tag{22}$$

From Eq. 13, $d_{ik}(t) = \delta_k^i(t)[1 + Q_j^i(t)]$, which implies that

$$D_j^i(t) = \sum_{k \in FM_j^i(t)} \phi_{jk}^i(t)[D_{jk}^i(t) + \delta_k^i(t)(1 + Q_j^i(t))] \tag{23}$$

Because MACC must be satisfied by every $k \in FM_j^i(t)$, $D_{jk}^i(t) < MAD_j^i(t)$. Then, if $D_j^{i*}(\tau)$ is the maximum path delay from i to j at time τ and $Q_j^{i*}(t)$ is the maximum backlog from i to j at time t, we obtain from Eq. 23 that

$$\begin{aligned} D_j^{i*}(t) \quad < \quad & \sum_{k \in FM_j^i(t)} [\phi_{jk}^i(t)\ \delta_k^i(t)(1 + Q_j^{i*}(t))] \\ & + MAD_j^i(t) \sum_{k \in FM_j^i(t)} \phi_{jk}^i(t) \end{aligned} \tag{24}$$

Let the maximum link propagation delay of all the links from i to a node in $FM_j^i(t)$ be

$$\Delta_j^i(t) = \max_{k \in FM_j^i(t)} \delta_k^i(t) \tag{25}$$

Therefore, the maximum path delay from i to j becomes

$$\begin{aligned} D_j^{i*}(t) \quad < \quad & \Delta_j^i(t) \sum_{k \in FM_j^i(t)} \phi_{jk}^i(t)[1 + Q_j^{i*}(t)] \\ & + MAD_j^i(t) \sum_{k \in FM_j^i(t)} \phi_{jk}^i(t) \end{aligned} \tag{26}$$

Noticing that $Q_j^{i*}(t)$ is independent of k and substituting Eq. 6 in Eq. 26 we obtain

$$D_j^{i*}(t) < \Delta_j^i(t)[1 + Q_j^{i*}(t)] + MAD_j^i(t) \tag{27}$$

The above equation is an upper bound on $D_j^i(t)$ that should be expected. It states that $D_j^i(t)$ must be smaller than the sum of the product of the backlog for i at node i times the maximum link propagation delay in node i's feasible multipath, plus $MAD_j^i(t)$. The first term of Eq. 27 corresponds to the delay incurred by sending all backlogged packets at time t to a neighbor with the longest link propagation delay. The second term corresponds to the maximum delay incurred by any neighbor receiving the backlog packets; because any such neighbor must be on $FM_j^i(t)$, that delay can be at most equal to $MAD_j^i(t)$.

Substituting Eq. 20 in Eq. 27, we can represent the same bound in terms of permit bucket parameters as follows:

$$D_j^{i*}(t) \quad < \quad MAD_j^i(t) + \Delta_j^i(t)\{1 + \sigma_j^i(t) \sum_{l|i \in FM_j^l(t)} [\sigma_j^l(t) + \rho_j^l(t-\tau)]\} \tag{28}$$

The bound given by Eqs. 27 and 28 for router i is based on a maximum delay offered by the neighbor of i and a maximum backlog allowed at router i. This is possible because of two main features of FMRA: datagrams are accepted only if routers have enough credits to ensure their delivery, and datagrams are delivered along loop-free paths. In contrast, in traditional datagram routing architectures, any datagram presented to a router is sent towards the destination, and the paths taken by such datagrams can have loops; therefore, it is not possible to ensure a finite delay for the entry router or any relay router servicing a datagram.

6 – Supporting Subnets

Having defined the basic protocol to support several quality-of-service types in a mobile integrated services network, we now extend the same to larger networks with hierarchical topologies. Two issues that are of concern here are *scalability* and *fairness* of the proposed credit-based mechanism.

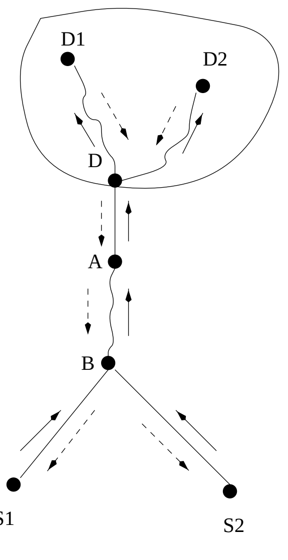


Figure 10: Credit Aggregation

6.1 – Credit Aggregation

Destination-oriented credits are aggregated at all intermediate nodes from destination towards source. Therefore, a credit value at each hop gives the total credits available for a specified destination through all the feasible paths through the downstream nodes.

To illustrate this, let us assume that the network is divided into clusters. Each cluster is viewed as a single entity from any node outside the cluster. All the information (distribution of credits) within the cluster is transparent to the nodes outside the cluster. The border node(s) at each cluster is responsible for sending aggregated credit information to all the nodes/clusters outside its own cluster and then to distribute the credits fairly among the routers within the cluster.

Available credits are aggregated for each destination (cluster) at all routers (boundary nodes) along the path from destination to a source. When a source needs credits to send traffic, it sends an explicit credit request towards the desired destination. Each destination in turn redistributes credits (reserving resources along the way). The number of packets entering the network is controlled at the COLB sources themselves. Because of this, the sources cannot misbehave and consume the resources allocated for other sources. The COLB sources are not allowed to send data unless they have credits to do so. This is because a packet is not allowed into the network unless it has resources to reach the destination. This ensures that the sources will not misbehave.

Because the sources are well-behaved, we can aggregate the available credits for destinations. This can be done without maintaining any additional information about the subnetworks, i.e., the topology of the subnetwork is transparent to the routers outside the subnetwork. Therefore, the credit-based approach is scalable.

Figure 10 shows an example of how the credits are aggregated in a subnet. $S1$ and $S2$ are the two sources and $D1$ and $D2$ are the two destinations for which the sources are sending data. The solid arrows indicate the direction of the data flow and the dashed arrows indicate the direction of the credit flow. But, from the point of view of the two sources, the information about both the destination have been integrated into a single destination D which is the boundary node of the destination cluster. D aggregates the credits available for the two destinations within its cluster and passes on that information towards sources $S1$ and $S2$.

Because the sources are well behaved and cannot send data to the destination unless they have credits to do so, the credits obtained are distributed between the two sources in proportion to their traffic arrival rate.

6.2 – Fairness

The system under consideration is said to be fair if two sources with the same flow specification get the same type of service from the system in terms of resource allocation and flow handling. Fairness can be ensured in the proposed destination-oriented credit mechanism among all active destinations, because all destinations are served according to the traffic rate using a weighted fair queueing scheme at all nodes. The available credits are distributed among active destinations in proportions relative to their traffic arrival rate. Furthermore, the congestion control scheme is exercised hop-by-hop on a per destination basis, so that a misbehaved flow for one destination does not block the flows for other destinations.

At each hop, credits are allocated to each flow depending on its relative input traffic rate into that node. Because the sources can not transmit packets into the network unless they have credits to do so (sources are well behaved), all sources get a fair share of the available bandwidth.

7 – Conclusions

We have presented a new routing architecture for mobile integrated services networks. Our approach is based on a hop-by-hop destination-oriented credit mechanism and a loop-free multipath routing algorithm. The new routing architecture dynamically adapts to congestion, so that the entries in the routing tables define long-term connections and thus reflect the state of the network at any time.

We have demonstrated by analysis that it is possible to provide performance bounds for the delivery of packets in a network in which routing decisions are made on a destination-oriented basis, rather than a connection-oriented manner.

A two-tier architecture to support end-to-end connections on top of a connectionless architecture was also presented to support several levels of quality of service. We have also showed that this credit-based mechanism is scalable and is fair.

We believe that the new routing architecture represents an attractive approach to extend wireline integrated services packet networks (ISPNs) with wireless mobile networks capable of supporting multimedia applications while network connectivity changes.

References

[1] D. Bertsekas. Dynamic behavior of shortest path routing algorithms for communication networks. *IEEE Trans. Automat. Control*, AC-27:60–74, 1982.

[2] D. Bertsekas and R. Gallager. *Data Networks.* Prentice-Hall Inc, 2nd edition, 1992.

[3] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource reservation protocol (RSVP) – version 1 functional specification. RFC 2205, Sept. 1997.

[4] D.D. Clark, S. Shenker, and L. Zhang, Supporting Real-Time Applications in an Integrated Services Packet network: Architecture and Mechanisms, *Proc. ACM SIGCOMM 92*, August 1992.

[5] R.L. Cruz. A calculus for network delay, part II: Network analysis. *IEEE Trans. Information Theory*, 37:132–141, 1991.

[6] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *ACM SIGCOMM*, pages 1–12, 1989.

[7] D. Ferrari, A. Banerjea, and H. Zhang, Network Support for Multimedia—A Discussion of The Tenet Approach, *Computer Networks and ISDN Syst.* Vol. 26, No. 10, pp 1167-1180, 1994.

[8] S. Floyd and V. Jacobson. The synchronization of periodic routing messages. *IEEE/ACM Trans. Networking*, 2(2):122–136, April 1994.

[9] J.J. Garcia-Luna-Aceves. Loop-Free Routing using Diffusing Computations. In *IEEE/ACM Trans. Networking*, pp 130–141, Feb. 1993.

[10] J.J. Garcia-Luna-Aceves and S. Murthy, A Path Finding Algorithm for Loop-Free Routing, *IEEE/ACM Trans. Networking*, February 1997.

[11] J.J. Garcia-Luna-Aceves and S. Murthy. A loop-free path-finding algorithm: Specification, verification and complexity. In *INFOCOM*, Boston, April 1995. IEEE.

[12] H.T Kung, Blackwell. T, and Chapman. A. Credit-based flow control for ATM networks: credit update protocol, adaptive credit allocation, and statistical multiplexing. In *ACM SIGCOMM*, volume 24 of *CCR*, pp 101–14, London, UK, Aug/Sept 1994. ACM.

[13] C. Ozveren, R. Simcoe, and G. Varghese. Reliable and efficient hop-by-hop flow control. In *SIGCOMM*, pp 89–110, London, Aug/Sept. 1994.

[14] A.K. Parekh and R.G. Gallager. A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case. *IEEE/ACM Trans. Networking*, (1(3)):344–357, June 1993.

[15] A.K. Parekh and R.G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: The multiple node case. *IEEE/ACM Trans. Networking*, 2(2):137–150, April 1994.

[16] S. Murthy and J.J. Garcia-Luna-Aceves, An Efficient Routing Protocol for Wireless Networks, *ACM Mobile Networks and Applications Journal*, Vol. 1, No. 2, 1996.

[17] Z. Wang and J. Crowcroft. QoS routing for supporting resource reservation. http://boom.cs.ucl.ac.uk/staff/zwang/pub.htm, 1996.