QoS-Based Routing of Multimedia Streams in Computer Networks

Ronny Vogel (TUC), Ralf G. Herrtwich (RWE Telliance AG), Winfried Kalfa (TUC), Hartmut Wittig (IBM ENC), Lars C. Wolf (IBM ENC)

Chemnitz University of Technology, Department of Computer Science, D-09107 Chemnitz
RWE Telliance AG, Gildehofstr. 1, D-45127 Essen
IBM European Networking Center, Vangerowstraße 18, D-69115 Heidelberg
Germany

{rvo, kalfa}@informatik.tu-chemnitz.de,
ralf_guido_herrtwich@telliance.rwe.de
{wittig, lwolf}@vnet.ibm.com

Continuous media such as audio and video require a certain quality of service (QoS) when transferred through computer networks. The selection of a network route for a particular media stream should hence take into account which route is best-suited for providing this QoS. QoSFinder is a method for QoS-based routing of multimedia streams. It is based on a path vector protocol that takes into account throughput, delay and loss rate of individual route segments. A simulation of QoSFinder shows that its heuristic is superior to metrics that are only based on one of these parameters. QoSFinder increases the probability of finding suitable paths through networks for distributed multimedia applications.

1 Introduction

For correct processing and transport of continuous media such as audio and video it is essential to meet the timing requirements of these media. To guarantee a certain quality of service (QoS) in multimedia systems careful resource management based on these requirements is needed. This applies to both local workstation and network resources. Hence, mechanisms that enhance QoS provision in multimedia networks require special attention.

Typically, multimedia communication is based on some notion of a flow of multimedia streams from one sender through a certain set of networks and routers to one or more receivers as shown in Figure 1. Such flows can be transmitted via connectionless (as in RSVP [1]) or connection-oriented (as in ST-II [14] and ST-II+ [2]) network services.

During flow setup at first a route has to be determined. Then resource managers at the selected nodes accept requests for the desired QoS, test whether this QoS can be provided, and report back the result [17], [18]. This process is repeated hop-by-hop until all destinations are reached. The QoS parameters are exchanged in the form of flow specifications that depend on the specific protocol.

If a node cannot provide the requested QoS, the setup is typically aborted and the application is informed that the requested QoS is not available. This indicates that the path of nodes that participate in a flow setup is crucial for success: While nodes on the selected route may be overloaded, other routes in the network may still provide sufficient QoS. How to select a proper route for a flow, however, is not determined by existing flow management protocols. In fact, it is often not considered at all (and left to the source) or addressed with traditional routing algorithms.

Because the problem of calculating a path subject to multiple constraints has been proven NP-complete for many common parameter combinations, usually the complexity is reduced by choosing a subset of QoS parameters [19], [15]. There are no dynamic routing methods that use a complete set of QoS parameters to determine a route for a multimedia data flow.

In this paper a dynamic routing method is introduced that is applicable to any given parameter set. It cooperates with the resource management of network nodes and considers the QoS requirements of multimedia streams for routing decisions. Section 2 introduces the architecture of our solution called QoSFinder. In Section 3 we deal with details of the routing protocol. Section 4 reports on experience with our implementation of QoSFinder and presents simulation and test results. Finally, Section 5 provides an outlook on future research in this area.

2 QoSFinder: A Routing Method for Continuous-Media Streams

Routing methods used in circuit-switched networks, packet-switched networks and packet-switched networks with virtual connections were investigated for their suitability to route continuous media streams in [16]. The design of the new routing method for connection-oriented network services is based on the following considerations: Multimedia flows are comparatively long-lived and routing decisions are made only at connection setup. Therefore every routing decision may influence the network state for a longer time compared to per-packet routing. So a more costly routing method could be acceptable if a higher probability of successful flow setup can be achieved.

2.1 Architecture

In our model, every node routing multimedia streams runs a QoSFinder process. The separation of the routing process from the resource management and from network protocols was a key design goal. QoSFinder is neither responsible for resource reservation nor can it give any guarantees, but it uses knowledge about the network topology and state obtained by communication with the resource managers to offer routes with a high success probability. A QoSFinder process communicates with four different types of environment modules:

- The local resource management system (RMS) and the RMS of each directly connected network.

- QoSFinder modules of the neighbor routers. Communication takes place via a routing protocol.
- All instances which request routes from QoSFinder, e.g., agents of flow-oriented network layer protocols such as ST-II.
- Management modules, e.g., SNMP subagents.

These modules are depicted in Figure 2.

2.2 Choice of the Routing Protocol

The routing protocol is determinant for the characteristics of a routing method. An analysis of *distance vector protocols* has shown, that they are not suitable for QoS-driven routing. Slow convergence prevents precise reactions and leads to persistent unstable states. Especially in meshed networks it is not possible to reliably avoid slow convergence. However, especially for these networks QoS-driven methods are needed. In case of network failures it is not guaranteed that a switch to alternative routes occurs immediately. Furthermore, alternative routes to the same destination via the same neighbor are not distinguishable [7].

Linkstate routing requires reliable transmission of linkstate packets (LSPs) to all participating routers. Compared with distance vector routing, linkstate routing uses extended information about network topology. After changes in the cost of a route or network topology linkstate routing converges quickly to a stable state . In the case of state changes in a part of the network all routers have to update their databases and to calculate new routes. The first step is the distribution of LSPs, the second step the local route calculation. Eavesdropping by endsystems is problematic, because LSPs have to be delivered reliably [7].

Path vector routing has been derived from distance vector routing [13]. Here, the route calculation takes place step by step during the process of spreading update messages. These messages contain destinations, costs and complete routes. On receipt of an update message it is checked whether one of the node's own addresses is already included in the route. If so, a routing loop is detected and the message will be dropped. Otherwise the own address will be added at the beginning of the route, the cost to reach the neighbor which sent the update will be added to the cost of the route and the message will be forwarded. Path vector routing avoids static routing loops, however, transient loops during route changes can occur. Because route calculation precedes message forwarding, this problem is less critical as in linkstate routing which requires stricter coordination for loop suppression [10].

Initially, a high resource consumption was assumed. But, a detailed analysis in [9] has shown that the algorithm results in a smaller additional expense than expected.

In path vector protocols, it is only necessary to forward the information received about a path if the cost to a destination has been changed compared with the last update message received for this route. Different strategies of update forwarding are possible. For example, one strategy is to use threshold values for cost changes as an additional criterion for update forwarding.

Route information is accumulated during the spread of update messages, starting at potential destinations. It is not necessary to spread all routes. Every router may use other route selection criteria without endangering routing stability. This allows policy-based routing [11].

Because of the above-mentioned properties, in our implementation a path vector protocol is used. The availability of different routes to the same destination allows QoSFinder to support quick changes to alternative routes in the case of network failures.

2.3 Metric

Parameter Set

To avoid violations in the quality of a distributed multimedia application the set of parameters should provide all QoS parameters of the application layer. However, the communication, computation and storage complexity of the routing method is increased by every additional, equally-treated, not correlated parameter. A reduction of the number of parameters, ordering them by priorities, or combining them into a single value reduces the degree of freedom in path selection, because information about the actual QoS requirements are not known before the connection request arrives [19]. Based on the considerations that many end-to-end parameters in today's networks are not completely independent from each other (e.g., if the load increases, delay, delay jitter or loss increase too [12], [4]), and that for routes with a higher number of networks and routers passed, the QoS is more likely to be worse than to be better compared to short routes, it is assumed that only a small subset of the existing routes has to be stored. Therefore, the used parameter set is not reduced or ordered by priority. The parameters are treated equally.

Because the QoSFinder mechanism has to provide routing in heterogenous networks, the QoS parameters are abstractions from those parameters provided by each network considered. Thereby, it is possible to aggregate them across different routers and networks [3]. Considering known QoS specifications ([5], [6], [7], [8], [17]) the following common set has been selected:

throughput, *delay* and *error rate*. For comparison of two routes it is desirable to know which QoS would be available for a new flow. The available throughput can be calculated from the capacity of a network segment and the current load. Current QoS values may be determined by an inquiry to the responsible RMS.

Another strategy would consider the additional load caused by an accepted flow. A successful flow establishment is assumed and the remaining resources are calculated accordingly. These values are the basis for the route selection. Because that calculation could only be done by involving the resource management systems, this approach was rejected.

The metric of QoSFinder is as follows: the *currently available* throughput t [kbit/s], the *current* delay d [µs], and the *current* error rate e [10^{-6}]. A parameter set (t, d, e) is assigned to each network segment and to each router. For every way through the same router (for protocol processing and packet forwarding) the same parameter values (t, d, e) are used.

Aggregation of Parameter Sets

Aggregation is the calculation of total parameter values for one route from the individual parameter values of network segments and routers. The parameter sets of networks and routers are treated equally. The number of passed network segments is given by n, the number of passed routers is specified by r. The following equations describe the aggregation:

$$t_{\text{total}} = \min\{t_i | (1 \le i \le n + r)\}$$

$$d_{\text{total}} = \sum_{i=1}^{n+r} d_i$$

$$e_{\text{total}} = 1 - \prod_{i=1}^{n+r} (1 - e_i)$$

Before sending an update message the local router QoS and the QoS of the first network towards the destination will be aggregated with the QoS of the corresponding route already stored in the routing database.

Comparison of Parameter Sets

The comparison of two parameter sets for a given demand yields the parameter set which is "better suited" to satisfy the requirements of that flow. This function is used for routing decisions, if more than one route to the specified destination is known. Before the comparison, the local router parameters and the parameters of the first network to pass on the route have to be aggregated to the route QoS. Then, it is checked whether the parameter sets fulfill the demand. A QoS (*t, d, e*) of a route is assumed to be sufficient for a demand (t_d, d_d, e_d), if:

$$t \geq t_d \wedge d \leq d_d \wedge e \leq e_d$$

In case alternative routes exist, parameter sets have to be compared. Table 1 shows the four possible cases. In Cases 2 and 3 the decision is obvious, because only one parameter set fulfills the demand. Cases 1 and 4 require further investigations. Although in Case 1 neither of both routes satisfies the demanded parameters, the "better" should be selected. Even if none of the known routes fulfills the demand, the routing process offers one of them. The routing process is not responsible for refusing a flow establishment. In Case 4 the "better" one of the two routes has to be selected.

To decide which route is the "better" one, it is necessary to define an order across the parameter sets. A relative quantity called *availability* is introduced:

$$\mathrm{A}_t = \frac{t}{t_d} \qquad \mathrm{A}_d = \frac{1/d}{1/d_d} \qquad \mathrm{A}_e = \frac{1/e}{1/e_d}$$

An availability of a parameter with a value of 1 means, that the tested parameter value is equal to the demanded parameter. An availability greater 1 indicates reserves. Availabilities less than 1 mean that the parameter value does not meet the demand. The selection of the preferred route follows these rules:

$$\min\{\mathrm{A}_{t1}, \mathrm{A}_{d1}, \mathrm{A}_{e1}\} > \min\{\mathrm{A}_{t2}, \mathrm{A}_{d2}, \mathrm{A}_{e2}\} \Rightarrow \text{path } 1$$
$$\min\{\mathrm{A}_{t1}, \mathrm{A}_{d1}, \mathrm{A}_{e1}\} < \min\{\mathrm{A}_{t2}, \mathrm{A}_{d2}, \mathrm{A}_{e2}\} \Rightarrow \text{path } 2$$
$$\min\{\mathrm{A}_{t1}, \mathrm{A}_{d1}, \mathrm{A}_{e1}\} = \min\{\mathrm{A}_{t2}, \mathrm{A}_{d2}, \mathrm{A}_{e2}\} \Rightarrow \text{refer to text}$$

In the case of equality the minimum is removed from both availability sets and the rules are applied to the resulting sets again. If this process terminates with empty sets, the parameter sets are assumed equally well suited. Hence, that path is refused which has the lowest value among all availabilities. If two parameter sets are equal, the route with less routers to pass (lower hop number) is selected. If these numbers are equal as well, the least recently used route is offered.

This heuristic has the following favourable properties: It is extendable to a higher number of parameters, for example by adding monetary cost or delay jitter. Hence, it can be adapted easily to other parameter sets. Furthermore, the same importance is attached to all parameters. Other policies may be realized by a suitable choice of (t_d, d_d, e_d). For example, a delay optimization can be done by reducing the demanded delay d_d to a very low value. Since a route request is only rejected by QoSFinder if there is no known route, the requesting instance will obtain the route with the minimal delay. On the other hand, a very high d_d value ensures that this parameter does not contribute to the routing decision. That way it is possible to take into account application's priorities concerning the parameters.

3 Protocol Description

Based on the principal mechanisms of QoSFinder presented in the previous section, we describe in the following subsections the communication with the resource managers, with route-requesting instances and the routing protocol agents. Figures 3 to 8 show the formal specification of the basic functionality of QoSFinder. All messages (protocol data units, PDUs) and events (timer expirations) are processed in the order they arrived.

3.1 Metric Aquisition

The metric is demanded from the resource management systems with a simple request-response-protocol. The message types are QOS_LOCAL_REQUEST/ QOS_LOCAL_RESPONSE and QOS_NET_REQUEST/QOS_NET_RESPONSE. QoSFinder can request current information periodically or event-driven. In case of periodic requests a high frequency introduces additional overhead, whereas low frequencies reduce relevance to the present. An event-driven request should be sent a short period after a routing decision, because there is a high probability of a change in the amount of available resources which implies route calculation. Additional mechanisms are necessary for event-driven requests, e.g., the recognition of events such as flow termination or QoS changes. The event-driven method can provide newer information but is more expensive.

Furthermore, a RMS might inform routing processes without a prior request (periodically or event-driven) about a changed resource utilization. With regard to the expense and the age of routing information this procedure would be optimal. A local RMS informs only the routing process at the same node. A network RMS has to inform all routers connected to the corresponding network segment. Broadcast mechanisms are used if possible. The prototype implementation of QoSFinder uses only periodic requests.

3.2 Requesting a Route

A system gets access to the information about routes via the following two message types.

ROUTE_REQUEST

This message is sent from the requesting entity to the QoSFinder process. In the PDU the required QoS and a list of targets are specified. On receipt of a ROUTE_REQUEST message, for each target the possibly best route is chosen among the available routes using the described heuristic.

ROUTE_RESPONSE

This response message contains the address of the router to be queried next for each target specified in the request message. The request messages may have been sent not only from the local protocol automaton but also from an endsystem which is attached to one of the directly connected networks. In this case, the ROUTE_RESPONSE message must only contain router addresses directly reachable by the endsystem, i.e., those addresses must be on the same network as the endsystem.

If the request was sent by a remote system, the following rule is applied during the address insertion: If the first address of the chosen route is on the same network as the system querying, then that address is inserted into the response. Otherwise, that router address is stored in the message which is on that particular network.

3.3 Routing Protocol Specification

All routing protocol messages contain a uniform protocol header including *type*, *version*, and *checksum* for the header and the parameter part. As shown in Table 2, the routing protocol consists of seven PDU types which are described in the following.

I_AM_HERE

Using this PDU a new QoSFinder process announces its existence to the QoSFinder processes at its neighbor routers. It is transmitted at the beginning of the initialization phase. A router which receives this PDU transmits a NOTIFY_UP message to indicate the newly-started router. The new router is added to the list of known neighbor routers if it is not already included in that list. The list entry is updated, i.e., the new router is marked as up and the time stamp for the last received HEARTBEAT message is set to the current time.

I_GO_DOWN

As part of its shutdown processing, a QoSFinder process sends an I_GO_DOWN message to all its neighbor routers to indicate that it will no longer be available. This way, these routers are able to switch quicker to other routes since they can react directly and do not have to wait for failure indications due to missing HEARTBEAT messages. A router which receives the I_GO_DOWN message transmits a NOTIFY_DOWN message to announce that no communication is possible via its link to the (stopping) router. Furthermore, the receiving router marks the stopping router as down in its list of neighbor routers, however, the list entry is not removed.

ALL_REQUEST

The ALL_REQUEST message is used to query a specific neighbor router about its routing tables and its table containing the information about breakdowns which have occurred. It is sent to a router from which a HEARTBEAT message has been received but which is either not contained in the list of neighbor routers or which is marked as down, e.g., if some routers are active and another router starts.

A new or restarted router itself transmits an ALL_REQUEST message to each router for the first HEARTBEAT message it receives from that neighbor. On receipt of an ALL_REQUEST message, a router sends the content of its routing table via UPDATE messages and the information from the table of breakdowns via NOTIFY_DOWN messages directly to the sender of the ALL_REQUEST message.

UPDATE

The purpose of the UPDATE messages is to provide the neighbor routers with information about the own routing table. Each message describes one route and the QoS available on it. As soon as the initialization phase has been completed, a router sends UPDATE messages for its entire routing table. At this time, the routing table contains entries about directly connected networks and the routing information which has been requested from neighbor routers via ALL_REQUEST messages. With the UPDATE messages, the neighbor routers are informed about the targets reachable via the new router and the link between the connected networks is established (Figure 9).

During the normal operation mode, an UPDATE message is sent if a certain event occurred. The events which trigger the UPDATE message transmission are changes larger than a specified threshold in the QoS of the local system or of directly connected networks, and the receipt of UPDATE messages from other routers.

When a router receives an UPDATE message, a routing loop detection test is performed by comparing the network parts of the router addresses contained in the route (from the second to the last address given in the route) with the network numbers of the directly connected networks. In the case that identical numbers are found, a routing loop has been detected and the further processing of this message is aborted. This way, it is guaranteed that the routes stored in the routing table are free of loops. If the complete addresses instead of only the network number part of the addresses would be compared, then only loops in which the router is passed multiple times are found but no loops due to the network topology (Figure 10).

Then it is checked whether the target is already known. If not, the route is added to the routing table, the QoS parameters are aggregated and an UPDATE message is sent to the other neighbors. If the target is already stored in the routing table, it has to be considered whether the advertised route or only other routes to the target are known.

If the routing table contains already information about the advertised route, then the QoS via this route is set to the new value. In the case that it differs significantly from the previous QoS an UPDATE PDU is transmitted. If the QoS of the route is non-ambiguously worse than the QoS available using a different route without breakdowns then the information about the considered route is removed. If the QoS via the route is non-ambiguously better than that available via another route than the latter route is removed (except there is a breakdown in the route with the changed QoS).

Similar operations are performed in case of known target and unknown advertised route. If the router knows already a better route to the target (without breakdowns) then the new route is not stored and the PDU is dropped. If there is no break in the new route and it is non-ambiguously better than an available route to that target, the old route is removed. If an old route has been removed, the new route is stored in the routing table and an UPDATE message is sent.

NOTIFY_DOWN

NOTIFY_DOWN messages are used to distribute information about breakdowns. The PDU contains the two addresses of adjacent routers between which no communication is possible. It is assumed that a breakdown is bidirectional, i.e., affects communication in either direction.

As seen in Figure 11, a NOTIFY_DOWN message is sent if no HEARTBEAT message has been received from a router within a certain time interval. The first address field of the PDU is set to NULL, the second address contains the address of the unreachable neighbor router. That way it is taken into account that a router has several addresses. A router which receives a NOTIFY_DOWN message with a first address set to NULL overwrites this field with the address of the sender. A NOTIFY_DOWN message with both addresses different from NULL is not modified.

The PDU is ignored in the case that the failure is already stored in the router's tables. Otherwise, the information is added to the table of breakdowns, the time stamp for that entry is set to the current time and a NOTIFY_DOWN message is sent to the other neighbor routers. To limit the protocol overhead, only one message (the first) is transmitted further if a router receives a NOTIFY_DOWN via two different routes.

NOTIFY_UP

The NOTIFY_UP message uses the same format as the NOTIFY_DOWN message. It informs routers that a path, which has been reported as malfunctioning by NOTIFY_DOWN messages, has become active again. If the table of a router which receives such a message contains an entry for that breakdown, it removes that information and sends a NOTIFY_UP to the neighbors. Otherwise the PDU is dropped.

HEARTBEAT

HEARTBEAT messages are sent periodically. On receipt of such a PDU, it is checked whether the sender is known. If so, the router updates the time stamp for the last HEARTBEAT message from that sender. If the sender is marked as down, then a NOTIFY_UP message is sent and the routing table from the sender is queried via ALL_REQUEST. In the case that the sender of the HEARTBEAT message is unknown, it is added to the routing table and the same steps as for a recovered router are performed, i.e., NOTIFY_UP and ALL_REQUEST messages are transmitted.

Here the purpose of the I_AM_HERE PDU becomes clear. Using this PDU, a new router registers itself at its neighbors. On receiving the first HEARTBEAT message from this new router, the neighbors do not have to react by sending ALL_REQUEST messages. The new router is already known because it distributed its accumulated information already at the end of its initialization phase.

4 Tests and Results

The metric and PDUs of QoSFinder have been implemented and tested. A test environment has been developed to evaluate QoSFinder's functionality. The results have shown that the QoSFinder approach is superior to other routing mechanisms with respect to the probability of a successful flow setup.

4.1 Test Environment

To perform the evaluation, the QoSFinder processes of the involved routers are embedded into the test environment which is a capsule for the communication with the resource managers, neighbored routers and network protocol agents. The protocol messages to these modules are sent from the sender router to the receiver router via this test environment. To evaluate QoS-Finder in larger network scenarios than available, the test environment also simulates the networks, the RMSs and the flow setups.

The message traffic is traced. A specific language has been developed to modify the parameters of the test environment. For example, this language is used to describe the infrastructure of the network and to send flow setup request events to the routers.

A flow setup is simulated by (1) checking the local resources, (2) sending the ROUTE_REQUEST message to the related QoSFinder processes and (3) adapting the QoS parameters of the networks and routers. A flow setup request is only accepted if the required resources are available and if no QoS guarantees for existing reservations would be violated. As a restriction of the test environment only linear dependencies of the network parameters delay and error rate from the bandwidth utilization are simulated.

4.2 Test Parameters

The following metrics have been used for qualitative and quantitative comparison with the QoSFinder metric:

1. minimum number of routers passed (hop count),
2. maximum throughput, and
3. minimum delay.

4.3 Network A: Qualitative Comparison

This subsection contains a qualitative comparison of the metrics. The topology of the test network is shown in Figure 12. An application at the sender node requests two flows to a receiver. The flows can only be established if the network provides the following QoS parameters: Flow X requires 1 Mbit/s minimum throughput, 50 ms maximum delay, and a maximum error rate of 20. To set up flow Y, 4 Mbit/s minimum throughput, 150 ms maximum delay, and a maximum error rate of 20 are required.

Each of the considered metrics is able to find a route in the network for one of these flows. However, the QoSFinder metric is the only metric which finds a way for both flows. For example, by using the hop metric and the minimum delay metric the request for flow Y is rejected because the throughput requirement is not fulfilled. Using the maximum throughput metric flow X is rejected because the delay exceeds the maximum delay allowed. Table 3 shows the results of this qualitative comparison. Hence, for this scenario the probability of successful flow setups has been increased by using the QoSFinder metric.

4.4 Network B: Quantitative Comparison

A larger test network (see Figure 13) is used for a quantitative comparison of the metrics. The test network is a meshed network comprising 13 network segments and 12 routers. Every network segment contains endsystems connected to the network. In the test network these endsystems are simulated by having one endsystem per network segment with the accumulated capacity (not shown in Figure 13). The sender, the receiver, the QoS parameters, the time of the request, and the duration were randomly generated for each of the flows. A total number of 1000 flow setup requests has been simulated. The results of the simulation are presented in Figure 14.

The experiment has shown that the QoSFinder metric is the best metric. By using the QoSFinder metric 79% of all flow setup requests were accepted. The minimum delay metric reached 71%, the minimum hop metric 62%, and the maximum bandwidth metric 58%. These probabilities are based on the relation between the successful flow setup to the overall number of flow setup requests[1].

Additional measurements in the network B have shown that the QoSFinder metric realizes the second best load balance (Table 4). However, a balanced load does not imply that the QoS requirements are met. The advantage of the QoSFinder approach is a higher probability of successfull flow setups.

5 Summary and Outlook

The QoSFinder method uses a set of QoS parameters when routing multimedia flows in computer networks. The heuristic is flexible and extensible. As shown by simulating various scenarios in diverse network topologies, it is superior to other dynamic routing methods by increasing the acceptance probability of multimedia flows. It also reduces flow setup times.

The simulations have also shown, that the number of routes to be stored per destination is much less (avg. 1.6 routes per destination in network B) than the total number of paths to that

1. Another measurement is the relation between the flows accepted and the number of flows acceptable. Because this evaluation requires complex graph searches regarding the structure of the network, the sequence of flow setup requests, and the parameters of the flows previously set up, it has not been implemented.

destination (avg. 12.5 in network B). Linkstate routing, instead of path vector routing with storage of pre-computed alternate routes, would reduce the storage complexity and the communication complexity, but would increase the complexity of path selection. It is planned to carry out a detailed complexity analysis of these routing methods in combination with the heuristic of QoSFinder.

Starting with the existing implementation, we intend to integrate QoSFinder into multimedia communication systems such as an IPng/RSVP implementation. Such implementations then would also need to address the following issues:

- Optimize routing for multicast trees so that cost is minimized for accessing all targets.
- Achieve load balancing and easy rerouting of streams within the constraints of QoS parameters.
- Use routing hierarchies to ensure scaling of routing mechanisms.

In addition, the robustness of the routing protocol itself needs to be considered.

6 References

[1] B. Braden, L. Zhang, S. Berson, S. Herzog, S. Jamin: *Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification*. Internet Draft, February 1996.

[2] L. Delgrossi, L. Berger, *Internet STream Protocol Version 2 (ST2) Protocol Specification - Version ST2+*. RFC 1819, August 1995.

[3] D. Jordan, M. Paterok, C. Vogt: *Layered Quality of Service Management in Heterogenous Networks.* IBM Technical Report TR 43.9304, Heidelberg, 1993.

[4] D. Minoli: *Broadband Network Analysis and Design.* Artech House, Boston, 1993.

[5] C.J. Parris, D. Ferrari: *A Dynamic Connection Management Scheme for Guaranteed Performance Services in Packet-Switching Integrated Services Networks.* The Tenet Group, TR-93-005, Computer Science Division, UC Berkeley and ICSI Berkeley, 1993.

[6] C. Partridge: A Proposed Flow Specification. RFC 1363, 1992.

[7] R. Perlman: *Interconnections: Bridges and Routers*. Addison Wesley Publishing Company, Reading, MA, 1992.

[8] J. Ramaekers, G. Ventre: *Quality of Service Negotiation in a Real-Time Commmunication Network.* TR-92-023, ISCI, Berkeley, CA, April 1992.

[9] Y. Rekhter: *IDRP protocol analysis: storage complexity.* ACM Computer Communications Review, Vol. 22, No. 2, 1992.

[10] Y. Rekhter: *Inter Domain Routing Protocol (IDRP).* Internetworking: Research And Experience, Vol. 4, No. 2, June 1993.

[11] Y. Rekhter, T. Li: *A Border Gateway Protocol 4 (BGP-4)*. RFC1654, July 1994.

[12] T. N. Saadawi, M. H. Ammar, A. El Hakeem: *Fundamentals of Telecommunication Networks*. John Wiley, New York, 1994.

[13] K.G. Shin, M. Chen: *Performance Analysis of Distributed Routing Strategies Free of Ping-Pong-Type Looping.* IEEE Transactions on Computers, Vol. COMP-36, No. 2, February 1987.

[14] C. Topolcic: *Experimental Internet Stream Protocol, Version 2 (ST-II).* RFC 1190, October 1990.

[15] D. C. Verma, P. M. Gopal: *Routing Reserved Bandwidth Multi-point Connections.* Proceedings of the ACM SIGCOMM'93, San Francisco, ACM CCR Vol. 23, No. 4, 1993.

[16] R. Vogel: *QoS-Driven Routing Algorithms* (in german). Diploma Thesis, Chemnitz University of Technology, June 1994.

[17] C. Vogt, R.G. Herrtwich, R. Nagarajan: *HeiRAT: The Heidelberg Resource Administration Technique.* KIVS'93, München, 1993.

[18] C. Vogt, L.C. Wolf, R.G. Herrtwich, H. Wittig: *HeiRAT – Quality-of-Service Management for Distributed Multimedia Systems*. submitted for publication.

[19] Z. Wang, J. Crowcroft: *Bandwidth-Delay Based Routing Algorithms.* Proceedings of the IEEE GlobeCom'95, Singapore, November 1995.

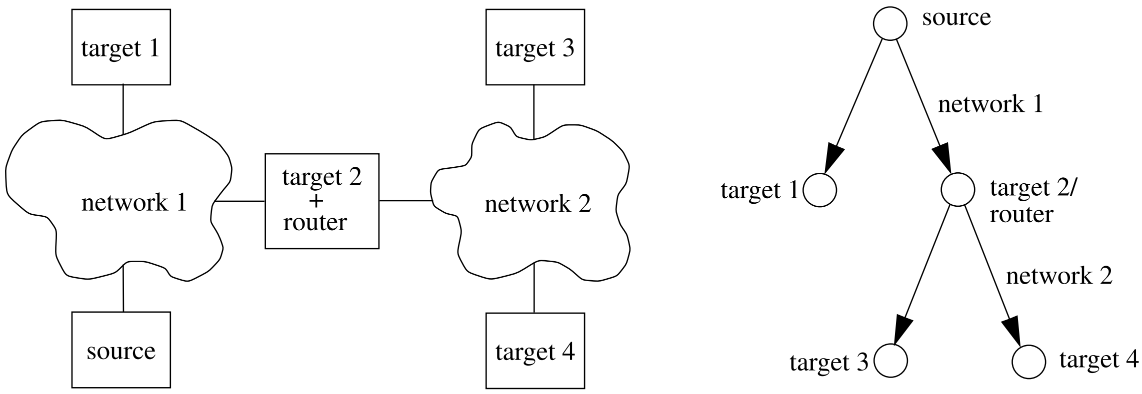


Figure 1: Flow of Multimedia Streams

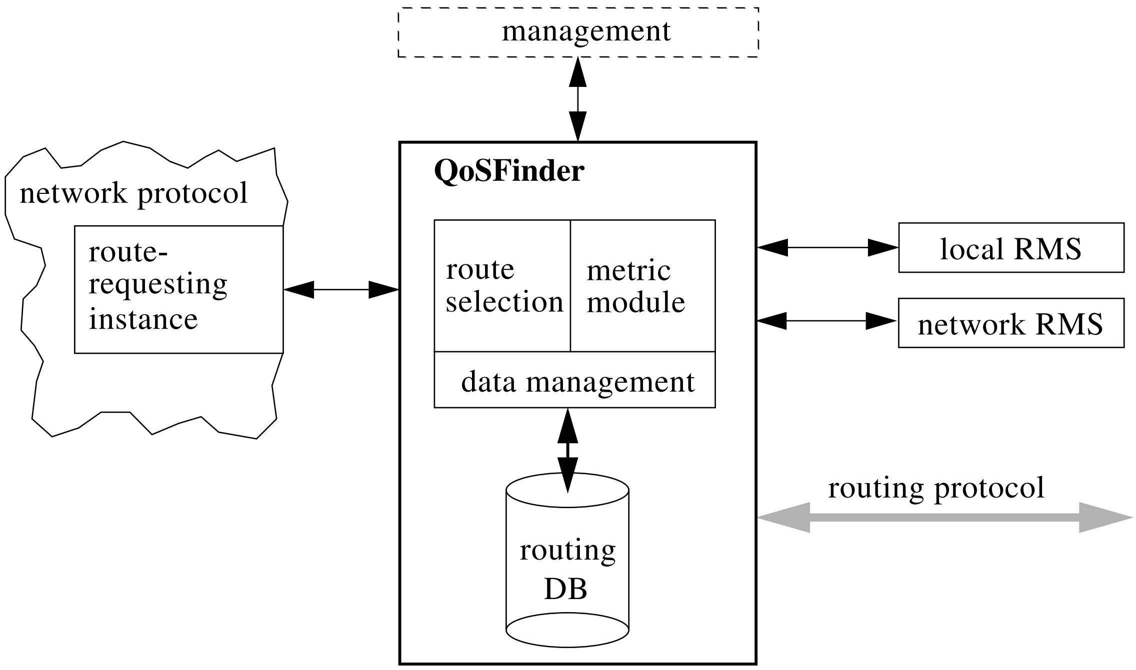


Figure 2: Architecture and Communication Relations of QoSFinder

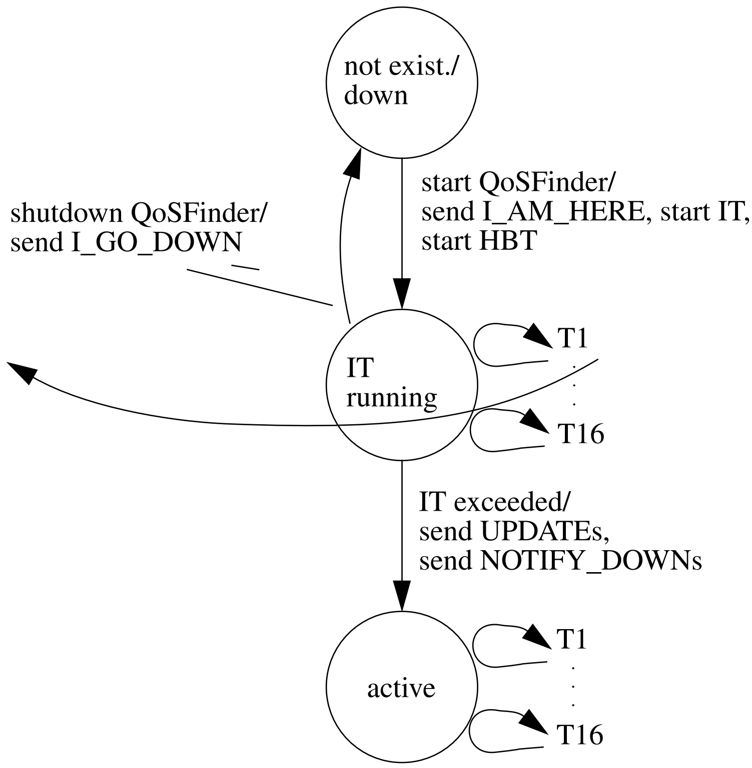


Figure 3: Global States of QoSFinder

A	set of own router addresses
N	set of addresses of neighbor routers
d	address of a destination
qos	QoS parameter set
a	address of a router
R	set of stored routes (routing table)
r	entry in routing table, r = (d, qos, (a_1, ..., a_n))
B	set of known breakdowns
b	entry for a break, b = (a_1, a_2)
e	address of an endsystem
any	wildcard for any value
$<$	“completely” worse in qos comparison
DDT	declare down timer (for every $a_i \in N$)
MDT	maximum down timer (for every $a_i \in N$ and $b_j \in B$ marked down)
IT	init timer
HBT	heartbeat timer (one global)
QIT	QoS inspect timer (one global)
WFRT	wait for response timer (one global)

Figure 4: Symbols of Formal Description

T1: I_AM_HERE from a_x

send NOTIFY_UP(0, a_x) to all $a_i \in N \setminus \{a_x\}$;
if $a_x \in N \wedge$ state(a_x) = DOWN **then**
    delete MDT(a_x);
end if
if $a_x \notin N$ **then**
    $N := N \cup \{a_x\}$;
end if
start DDT(a_x);
state(a_x) := UP;

T2: I_GO_DOWN from a_x

send NOTIFY_DOWN(0, a_x) to all $a_i \in N \setminus \{a_x\}$;
if $a_x \in N$ **then**
    state(a_x) := DOWN
end if

T3: ALL_REQUEST from a_x

for all $r_i \in R$ **do**
    send UPDATE(r_i) to a_x;
end for
for all $b_i \in B$ **do**
    send NOTIFY_DOWN(b_i) to a_x;
end for

Figure 5: Formal Specification

T4: UPDATE(d, qos, (0, ..., a_n)) from a_x

```
(d, qos, (0, ..., a_n)) → (d, qos, (a_x, ..., a_n));     // first address := a_x
for all a_i in (a_2, ..., a_n) do                          // looptest
    for all a_j ∈ A do
        if network_part(a_i) = network_part(a_j) then
            ignore message and return;
        end if
    end for
end for
qos_agg := aggregate(qos, qos_local, qos_net of first network towards d);
if (d, any, any) ∉ R then                                  // destination unknown
    R := R ∪ {(d, qos, (a_x, ..., a_n))};
    send UPDATE(d, qos_agg, (0, a_x, ..., a_n)) to all a_i ∈ N \ {a_x};
else if (d, any, (a_x, ..., a_n)) ∈ R then                 // destination and route known
    if qos changed significantly then
        R := R \ {(d, qos_old, (a_x, ..., a_n))} ∪ {(d, qos, (a_x, ..., a_n))};
        send UPDATE(d, qos_agg, (0, a_x, ..., a_n)) to all a_i ∈ N \ {a_x};
    end if
    if ¬ (break in (a_x, ..., a_n)) then                   // delete complete worse routes
        for all r_i = (d, qos_i, any ≠ (a_x, ..., a_n)) do
            if qos_i < qos then
                R := R \ {r_i};
            end if
        end for
    end if
    if ∃ (d, qos_i, route = (any ≠ (a_x, ..., a_n))) ∈ R ∧ qos_i > qos ∧ ¬ (break in route) then
        R := R \ {(d, qos, (a_x, ..., a_n))};
    end if
// destination known, route unknown:
else if (d, any, any ≠ (a_x, ..., a_n)) ∈ R ∧ (d, any, (a_x, ..., a_n)) ∉ R then
    if ∃ (d, qos_i, route = (any ≠ (a_x, ..., a_n))) ∈ R ∧ qos_i > qos ∧ ¬ (break in route) then
        ignore message and return;
    end if
    updateflag := FALSE;
    if ¬ (break in (a_x, ..., a_n)) then
        for all r_i = (d, qos_i, any ≠ (a_x, ..., a_n)) do
            if qos_i < qos then
                R := R \ {r_i};
                updateflag := TRUE;
            end if
        end for
    end if
    if updateflag then
        send UPDATE(d, qos_agg, (0, a_x, ..., a_n)) to all a_i ∈ N \ {a_x};
        R := R ∪ {(d, qos, (a_x, ..., a_n))};
    end if
end if
```

Figure 6: Formal Specification (continued)

T5: NOTIFY_DOWN(a_k, a_l) from a_x

if $a_k = 0$ **then** $a_k := a_x$;
end if
if $(a_k, a_l) \in B$ **then**
  ignore message and return;
else
  $B := B \cup (a_k, a_l)$;
  send NOTIFY_DOWN(a_k, a_l) to all $a_i \in N \setminus \{a_x\}$;
  start MDT((a_k, a_l));
end if

T6: NOTIFY_UP(a_k, a_l) from a_x

if $a_k = 0$ **then** $a_k := a_x$;
end if
if $(b = (a_k, a_l)) \in B$ **then**
  $B := B \setminus \{(a_k, a_l)\}$;
  delete MDT(b);
  send NOTIFY_UP(a_k, a_l) to all $a_i \in N \setminus \{a_x\}$;
end if

T7: HEARTBEAT from a_x

if $a_x \in N$ **then**
  if state(a_x) = DOWN **then**
    state(a_x) := UP;
    delete MDT(a_x);
    send NOTIFY_UP(0, a_x) to all $a_i \in N \setminus \{a_x\}$;
    send ALL_REQUEST to a_x;
  end if
else
  $N := N \cup \{a_x\}$;
  send NOTIFY_UP(0, a_x) to all $a_i \in N \setminus \{a_x\}$;
  send ALL_REQUEST to a_x;
end if
start DDT(a_x);

T8: QOS_LOCAL_RESPONSE($qos_{local, new}$) from local RMS

if $qos_{local, new} \neq qos_{local}$ **then**
  $qos_{local} := qos_{local, new}$;
  qos_changed := TRUE;
end if

T9: QOS_NET_RESPONSE($qos_{network\ i, new}$) from network RMS

if $qos_{network\ i, new} \neq qos_{network\ i}$ **then**
  $qos_{network\ i} := qos_{network\ i, new}$;
  qos_changed := TRUE;
end if

Figure 7: Formal Specification (continued)

T10: ROUTE_REQUEST(id, qos, (d_1, ..., d_n)) from a_x or from e_x

for all d_i in (d_1, ..., d_n) **do**
    select best suited route to d_i; // see subsection 2.3.
end for
send ROUTE_RESPONSE(id, (a_1, ..., a_n)); // see subsection 3.2.

T11: HBT exceeded

send HEARTBEAT to all $a_i \in N$;
start HBT;

T12: QIT exceeded

send QOS_LOCAL_REQUEST to local RMS;
send QOS_NET_REQUEST to all directly connected network RMS;
start QIT;
start WFRT;

T13: WFRT exceeded

if qos_changed = TRUE **then**
    for all $r_i \in R$ **do**
        qos_{agg} := aggregate(qos_i, qos_{local}, qos_{net} of respective network);
        if qos changed significantly **then**
            send UPDATE(d, qos_{agg}, (0, a_1, ..., a_n)) to all $a_j \in N \setminus \{a_1\}$
        end if
    end for
end if
qos_changed := FALSE;

T14: DDT($a_x \in N$) exceeded

state(a_x) := DOWN;
send NOTIFY_DOWN(0, a_x) to all $a_i \in N \setminus \{a_x\}$;
start MDT(a_x);

T15: MDT($a_x \in N$) exceeded

for all r_i = (any, any, (a_x, any, ..., any)) $\in R$ **do**
    $R := R \setminus \{r_i\}$;
end for
$N := N \setminus \{a_x\}$;

T16: MDT($b_x = (a_1, a_2) \in B$) exceeded

for all r_i = (any, any, (any, ..., (a_1, a_2), ..., any)) $\in R$ **do**
    $R := R \setminus \{r_i\}$;
end for
for all r_i = (any, any, (any, ..., (a_2, a_1), ..., any)) $\in R$ **do**
    $R := R \setminus \{r_i\}$;
end for
$B := B \setminus \{b_x\}$;

Figure 8: Formal Specification (continued)

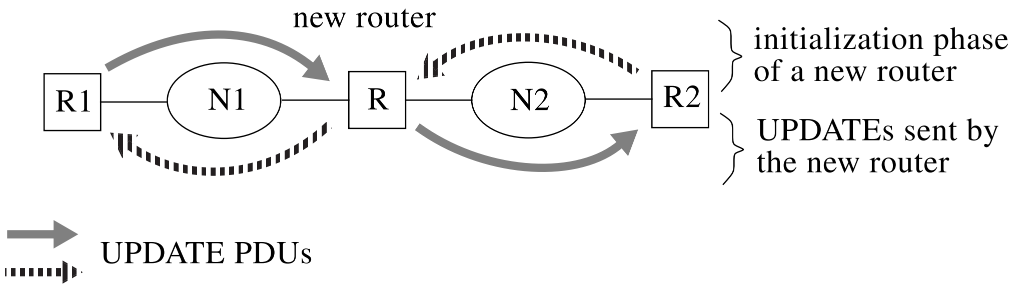


Figure 9: Initialization Phase of a Router

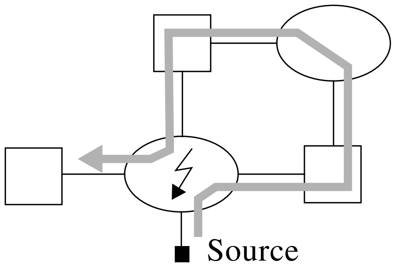


Figure 10: Routing Loop through a Network

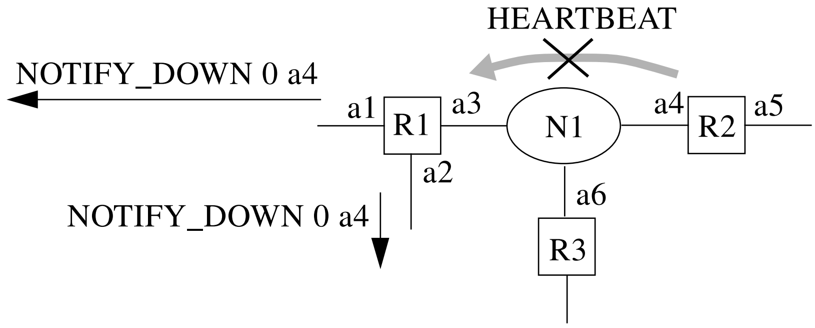
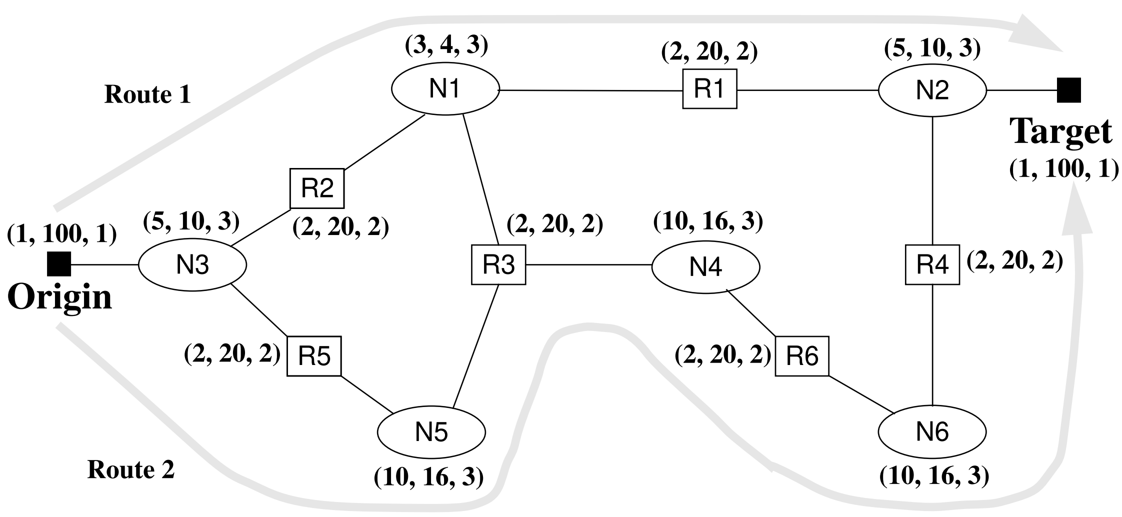


Figure 11: Transmission of NOTIFY_DOWN



QOS = (Delay [ms]; Throughput [Mbit/s]; Error rate [10^{-6}]), values before connection establishments

Figure 12: Network A with QoS Parameters

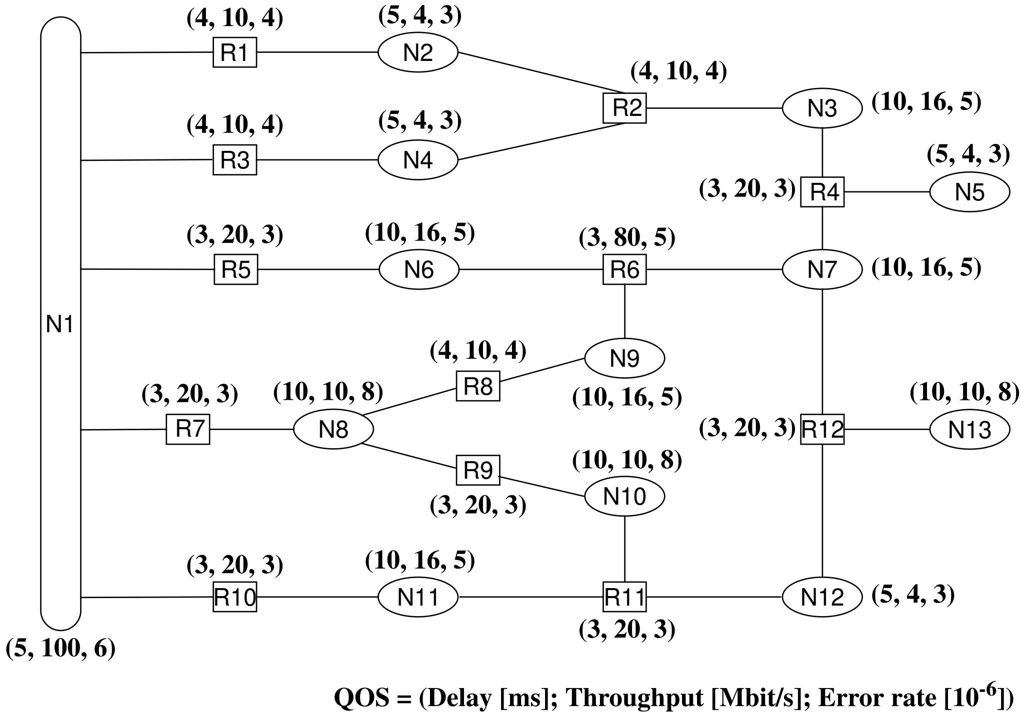


Figure 13: Network B with QoS Parameters

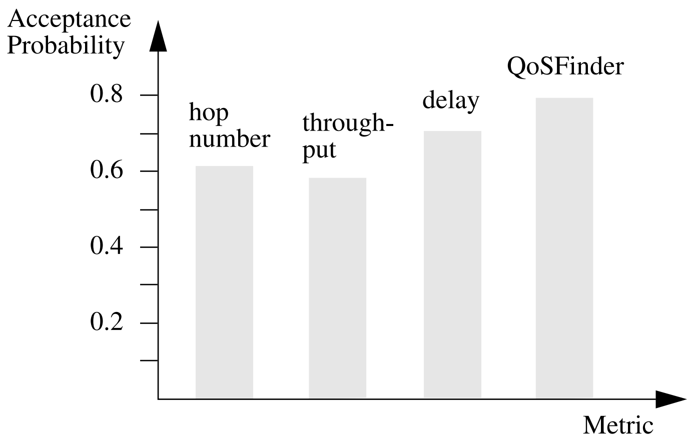


Figure 14: Probability of Successful Flow Setups

Case	Demand satisfied?		Selected
	(t_1, d_1, e_1)	(t_2, d_2, e_2)	
1	no	no	see text
2	yes	no	(t_1, d_1, e_1)
3	no	yes	(t_2, d_2, e_2)
4	yes	yes	see text

Table 1: Comparison of Parameter Sets

PDU Type	Occurrence	Receiver	Parameters
I_AM_HERE	once	all neighbors	—
I_GO_DOWN	once	all neighbors	—
ALL_REQUEST	event-driven	one neighbor	—
UPDATE	event-driven	all neighbors (limited)	Target, QoS, Route, Length of Route
NOTIFY_DOWN	event-driven	all neighbors (limited)	two addresses
NOTIFY_UP	event-driven	all neighbors (limited)	two addresses
HEARTBEAT	periodic	all neighbors	—

Table 2: PDU Types of the Routing Protocol

Metric	Flow X	Flow Y
minimum hop	route 1 ✓	route 1 not successful
maximum throughput	route 2 not successful	route 2 ✓
minimum delay	route 1 ✓	route 1 not successful
QoSFinder	route 1 ✓	route 2 ✓

Table 3: Results of the Qualitative Comparison

Metric	Average network load	Standard deviation
minimum hop	24.6	16.1
maximum throughput	18.8	14.0
minimum delay	24.6	16.1
QoSFinder	25.4	15.1

Table 4: Network Load