The Forwarding on Gates Architecture: Merging IntServ and DiffServ

Florian Liers, Thomas Volkert, Andreas Mitschele-Thiel
Integrated Communication Systems Group
Technical University of Ilmenau
Ilmenau, Germany
e-mail: {Florian.Liers, Thomas.Volkert, Mitsch}@tu-ilmenau.de

***Abstract*—Quality of Service (QoS) will be a major enabler for Future Internet applications and services. However, today's Internet provides no suitable QoS support for end-to-end connections due to several drawbacks of IntServ and DiffServ. Therefore, this paper proposes the "Forwarding on Gates" architecture, which uses a new network protocol designed to handle IntServ and DiffServ in an integrated way. The architecture supports resource reservations for QoS guarantees, like in IntServ scenarios, and prioritized traffic, like in DiffServ scenarios. Furthermore, a combination of both is supported. This paper introduces the architecture and defines the network protocol used to implement these features. The evaluation includes theoretical descriptions of the network configuration for the different scenarios and simulation results concerning the protocol overhead in large-scale networks. Our new architecture is able to support QoS in a scalable way, since it allows a network providing QoS to move states and delegate decisions about the QoS usage to the entity using the QoS.**

Keywords—Future Internet; network protocol; architecture; QoS.

I. INTRODUCTION

The Future Internet will be faced with much more applications requiring Quality of Service (QoS). Among the first, Internet video streaming is already stressing today's Internet. Forecasts predict that in 2015 about 62% of the traffic will be video [1]. For live video streams, as required for remote medical operations or for football games, QoS is required. Due to the large number of end hosts on the Internet and the large number of connections between them, scalability is a major concern of QoS provisioning.

IP only provides a best effort service. Therefore, IntServ and DiffServ were developed to handle QoS on the Internet. However, both have pros and cons. The IntServ approach [2] with its signaling protocol RSVP provides end-to-end QoS by introducing states in each intermediate router a flow passes through. According to [3], the Resource Reservation Protocol (RSVP) has to handle states for classification, scheduling and signaling. The classification states define how incoming packets are mapped to flows. With RSVP, such a state consists of a source address, a destination address and a protocol number (and optionally port numbers). Scheduling states define how flows are handled. For example, a flow can be mapped to the queue of an outgoing hardware interface with a priority for a scheduler. Finally, signaling states represent management information like authentication information and timers. For each flow, an intermediate node requires one set of these states. Due to memory limitations, such an approach causes scalability problems for scenarios with many flows [4].

DiffServ [5] aims at solving the scalability issue by introducing a small set of QoS classes used inside networks. Each QoS class defines a type of service and requires scheduling and signaling states. Thus, the number of states does not depend on the number of connections. However, DiffServ is not able to provide guarantees, since it is not aware of each individual flow. Edge routers of a network contain the classification states of a DiffServ network, in order to map incoming packets to the internal QoS classes. The classification states represent the rules for this mapping. Since most interfaces with incoming traffic will transport multiplexed flows, like multiple TCP connections over one Ethernet link, the classification is mainly done by (more or less deep) packet inspection. For example, port numbers or packet sizes can be used for classification.

IntServ and DiffServ can be combined to leverage the advantages of both approaches. IntServ provides the signaling for flows between ingress routers and DiffServ provides a set of QoS classes used inside networks [6, 7]. However, the scalability problem of IntServ now appears at the ingress routers. They have to store classification states per flow. Since the number of scheduling and signaling states remains limited due to the DiffServ classes, the classification states are the main problem. In order to maintain the states, signaling is required. The processing load of handling these messages increase the burden on a network. In the past, proposals focused on reducing the number of flows, e.g., through aggregation [3].

Our key contribution is the proposal of an orthogonal strategy: move the classification states away from ingress routers to routers handling smaller amounts of flows. Furthermore, some decision-making authority is delegated from the QoS provider to the entity using QoS in order to reduce the required signaling overhead. As discussed in more detail in Section IV, today's network protocol IP is not able to support both in all use cases. Therefore, this paper presents a new network protocol enabling the movement of classification states and the delegation of decisions between routers. Our solution is suitable for IntServ and DiffServ scenarios and is able to handle combinations of both. Its main feature is the flexible placement of the classification states according to the network graph and the load in the

 ISBN: 978-1-61208-211-0

system. It enables the handling of both QoS approaches in a single mechanism.

The remainder of this paper is structured as follows: Section II describes our system architecture. Section III introduces the protocol and how its header is processed. Afterwards, in Section IV, the implementation of the use cases based on our architecture and protocol are presented. Section V shows evaluation results from a protocol simulation. Section VI discusses related work. In the end, the main results of this work are summarized and an outlook to future work is given.

II. FORWARDING ON GATES ARCHITECTURE

The "Forwarding on Gates" (FoG) architecture splits forwarding and routing into two logical components. Both encapsulate specific tasks. The forwarding component is responsible for relaying packets between routers and hosts. It handles the resource management and enforcement of resource reservations in order to take non-functional properties such as delay and bandwidth into account. The routing component is responsible for calculating paths through the network with respect to non-functional requirements given by applications [8]. Both are linked via a route definition. The routing component specifies a route and the forwarding component forwards packets along the route. The authentication component is the third logical component of FoG. It checks the authentication of signaling messages in order to secure access to the management functions. The authentication is the basis for authorization decisions and accounting of QoS. In this paper, we use the term *flow* in a more abstract manner than *connection*. However, a flow can be a connection between end hosts.

For the following discussion, we introduce the term *QoS function*, which generalizes QoS provisioning regardless of the QoS architecture. A QoS function represents the setup required to send packets with QoS constraints. Examples of QoS functions are setups implementing a DiffServ class or an IntServ reservation. QoS functions can provide guarantees ranging from "hard" with fixed limits over "soft" with probabilistic QoS guarantees to vague goals like "optimized for delay" or "best-effort". A QoS function comprises its scheduling and signaling states. The classification states are not included.

In addition to the separation of routing and forwarding [9], our architecture has some more features. Examples are reduced forwarding table size [10], enabling routers to choose their address format [11], enabling applications to specify their requirements [8], hiding addresses from applications and support for various intra-network techniques. However, they are shared with other approaches from literature and are not the focus of this paper.

A. Forwarding Component

Today's Internet operates over interfaces of routers and hosts and links in between. FoG's forwarding component uses a virtual representation of the network, which has the form of a graph. Host and routers are represented by one or more vertices, which are called *forwarding nodes*. Edges between forwarding nodes are called *gates* and represent uni-directional links between them. In order to support QoS, multiple edges between adjacent nodes are allowed. An edge is equivalent to a link with a QoS function between two routers or hosts. Each outgoing gate of a forwarding node is assigned a *gate number*, which is unique in the scope of this forwarding node. Each FoG packet has a header, which contains the order of gates to pass through explicitly. Details about the route and how the forwarding node processes it are given in Section III.

Gates are set up with a FoG-specific management protocol. The forwarding nodes process the signaling messages of that protocol and modify the graph of forwarding nodes and gates as requested. In order to secure this management, signaling messages are signed via the authentication service by the sender. The receiver uses the authentication service to verify the signature.

The forwarding component informs the routing component about available gates and forwarding nodes to enable route calculations based on them. However, gates not intended for other flows can be hidden in order to exclude them from the routing calculations.

B. Routing Component

The routing component is responsible for route calculations based on the information received from the forwarding component. Based on a starting forwarding node and a destination address, it calculates a route through the graph of gates and forwarding nodes. QoS requirements serve as constrains for the calculation. If all gates to the destination are known, an explicit route is the result. If some gates are not known, the route is only a partial one. A partial route contains the destination address or the address of an intermediate node. During the forwarding process, the missing part of the route will be calculated based on this address.

In common scenarios, the subset known to a routing instance is a connected graph, which represents the "area" of the network itself and its surroundings. The knowledge about the parts outside of this subset is more abstract. A routing component knows about the connectivity but does not know the gate numbers required to specify the route explicitly. This is comparable with the situation known from the Border Gateway Protocol (BGP). A BGP entity knows about the existence of a route (and its cost) but does not know the outgoing ports of the intermediate routers which have to be used.

If there are insufficient gates to form an end-to-end route, a routing component can request the setup of new gates by a forwarding component. In particular, routing components can request gates with specific QoS capabilities in order to satisfy the QoS requirements of a particular route request.

 ISBN: 978-1-61208-211-0

C. Authentication Component

The authentication component is used to generate signatures for signaling messages and to check such signatures. Based on the authentication check, authorization decisions and accounting are done. It is mainly used by the forwarding component to secure gate management. Furthermore, applications can use the authentication in order to sign data packets. Thus, the authentication has an impact on the packet structure described in the next section. However, due to the QoS focus of this paper, the authentication component is not described in more detail.

III. NETWORK PROTOCOL AND ITS PROCESSING

The FoG packet structure is shown in Figure 1. It starts with a header comprising all information required to decide the next hop. This enables a router to make a forwarding decision before the packet is fully received. The packet ends with a trailer containing all information that can be added to a packet after receiving it completely. The trailer is optional and can be omitted. The header and trailer elements are defined as follows:

- Header
  - Length of the header in bytes: This field is required to access the payload as subsequent versions of the FoG protocol can add new header fields between the header fields defined in this paper and the payload.
  - Flag field which indicates
    - if the reverse route in the trailer is present,
    - if the packet is a signaling message, and
    - if the authentication information in the trailer is present.
  - Modification counter to prevent routing loops due to invalid gate setups
  - Length of the payload in bytes
  - Forward route for the packet
- Trailer
  - Length of the authentication information in bytes
  - Authentication information itself
  - Revers route for answers

Each route starts with a length field followed by a stack of route segments. There are two types of segments:

- The explicit route segment is a stack of gate numbers, defining explicitly a sequence of gates to travel through.

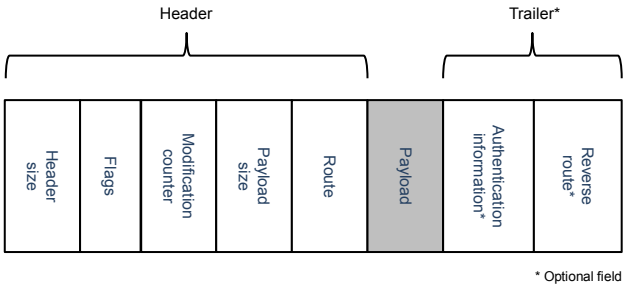

Figure 1. FoG protocol packet structure.

- The destination route segment contains a destination name or address and the requirements for the remaining route to this destination.

Each forwarding node processes the route of a packet as shown in Figure 2. If the route is empty, the packet has reached its destination. If the signaling flag is set, the packet contains signaling information dealing with the setup of gates and connection establishment between applications. The signaling message is handled by the host or router, and it updates the gate and forwarding node graph accordingly. If the signaling flag is not set and the forwarding node has a socket attached to it, it removes the FoG header and trailer and stores the payload in the receive buffer of the socket. If the route is not empty, the forwarding node processes the topmost segment. If it is an explicit route segment, it removes the topmost gate number and uses it to lookup one of its outgoing gates. If there is an outgoing gate with this gate number, the packet is handed over to this gate for further processing. If the explicit route segment is empty, it is removed from the route and the procedure restarts. If the topmost segment is a destination segment, the forwarding node has to contact the routing component in order to get the next explicit route segment.

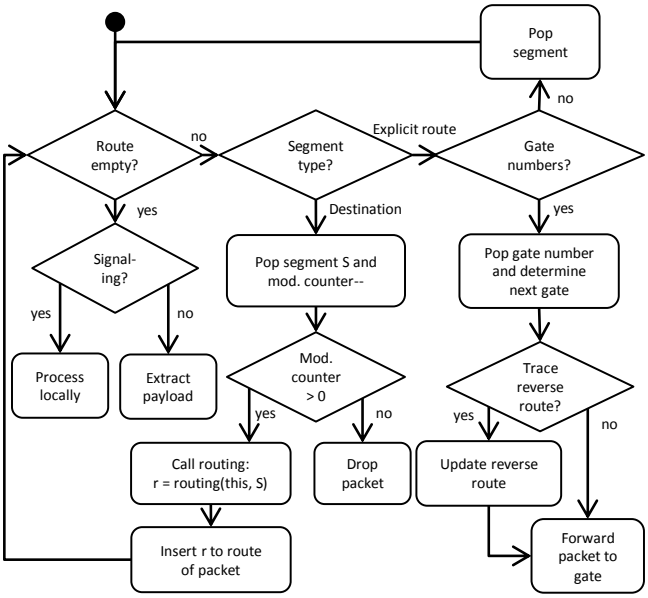

Figure 2. Forwarding node procedure without error cases.

The routing component calculates a route from the forwarding node to the destination. The requirements given in the destination segment are used as requirements for the route. The route returned by the routing component is added to the route of the packet and the procedure is restarted.

The modification counter is decremented if the route is changed in a way that loops might occur. It is used to avoid routing loops.

The reverse route of a packet is recorded if the reverse route flag in the packet header is set. A forwarding node has to derive the reverse gate number from the gate chosen for the forward direction. The reverse route does not need to be symmetric to the forward route. Furthermore, an intermediate forwarding node, which is not able or allowed to record the reverse route using explicit route segments, can insert a destination segment to the reverse route. The reverse route can be used by the receiver of a packet to reply to the sender. The main benefit of using the reverse route instead of an address is twofold. First, routing requests for reply packets are avoided and second, addresses for sending nodes are not required. The latter is useful for hosts acting only as clients. A server can reply by using the reverse route without forcing the client to have an address. If a reply with a traced backward route is received by a client, it knows the route the request packet has traveled. In most cases, this route contains less destination segments and more explicit route segments. Therefore, the client can use this route for subsequent packets in order to reduce the routing overhead and the delay for its packets.

The destination segment is not necessarily the last segment in a route. As shown by the use cases in the next section, it might define only an intermediate node of the route, due to missing knowledge about gate numbers. Theoretically, multiple destination segments in one route are possible, which would emulate loose source routing. Due to security considerations [12], policies might restrict that.

IV. USE CASES

Based on the FoG architecture and its network protocol, we will now investigate three different use cases showing the provisioning of QoS functions ranging from IntServ to DiffServ to a combination of both. For simplicity, the same example network is used in each case. Only the gate setup and the responsibility for the classification states (CS) differ.

Figure 3 and 4 show three networks with network 3 providing QoS functions in the form of gates to network 1 and 2. Gates are depicted as straight lines between the forwarding nodes (FN_i), which are shown as dots. The dotted lines represent connectivity through some other network, where the gate numbers are not known. Known gate numbers are depicted with small letters. Each network has not only the forwarding component but also the routing component as defined by the architecture. It is depicted as an extra box with its known components inside.

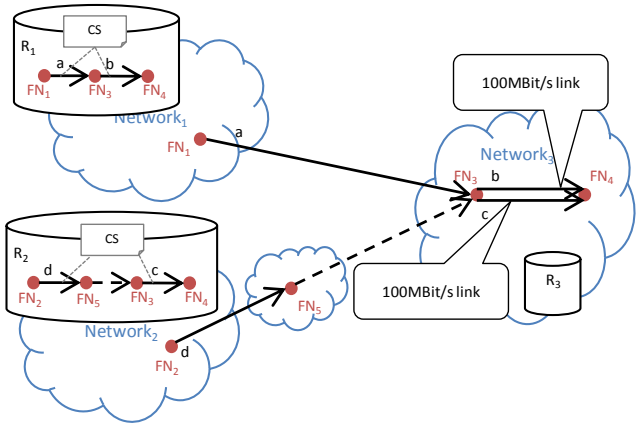

Figure 3. Gates representing IntServ reservations.

A. IntServ gates

In Figure 3, networks 1 and 2 have requested QoS functions from network 3. Network 3 has set up one gate for each request and network 1 and 2 have informed their routing component about these gates. For example, each gate represents a (virtual) link providing 100 MBit/s. The router, which is represented by FN_3, has to store the scheduling and signaling states required to provide and enforce these QoS functions. However, the classification states are not stored by FN_3 but have been moved to network 1 and 2, respectively. Their routing components know about gates *b* and *c*, respectively, and handle the decision about which flows are mapped to these gates.

If network 1 would like to establish a flow, the entity responsible for flow creation (e.g. one of the control nodes or the network edge) starts sending a signaling message with a route, which just contains a destination segment with the address of the destination and the requirements for the route, e.g., a minimum bandwidth of 10 MBit/s. In this example, the destination is FN_4. The packet with the route [[address(FN_4)]] is inserted into the forwarding component FN_1, which proceeds as described in Section III. Since the topmost segment of the route is a destination segment, it contacts the routing component R_1. In the given case, R_1 knows a route with all gate numbers to the destination. We assume that R_1 did not map too many flows on these gates and that therefore there is enough remaining capacity. R_1 maps this new flow on the gates *a* and *b* and updates its classification states. It returns the route [[*a*, *b*]] containing only one explicit route segment. FN_1 removes the destination segment from the packet and inserts this new route into the route field in the packet. FN_1 restarts the procedure with the explicit route segment as topmost segment. It pops the gate number *a* from the gate number stack and looks it up in its list of outgoing gates. Then, it hands over the packet to gate *a*. The gate transports the packet to the next hop via a link layer, e.g., Ethernet. The packet arrives at FN_3 with the route [[*b*]]. It pops *b* from the stack and hands over the packet to gate *b*. FN_4 receives the packet with an empty route and processes the packet locally.

 ISBN: 978-1-61208-211-0

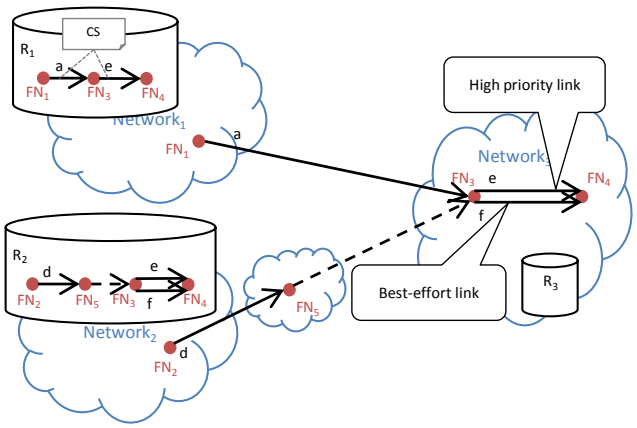

Figure 4. Gates representing DiffServ classes.

For network 2, the process is similar. However, there is a difference in the route required to reach FN_3. R_2 calculates a route with three route segments: [[*d*], [address(FN_3)], [*c*]]. In contrast to the route calculated by network 1, one more request to the routing component has to be done. FN_5 receives a packet with [address(FN_3)] as topmost segment, which triggers the remaining route request.

B. DiffServ gates

In Figure 4, network 3 provides one gate with a high priority and a best effort gate with a low priority. The latter is included in the scenario to demonstrate the integration of non-QoS links in FoG. The routing components of network 1 and 2 have slightly different views on this situation. R_1 only knows about *e* and R_2 knows about both *e* and *f*. The reason might be that R_1 decided to omit *f* or network 1 did not request a best-effort gate from network 3. Moreover, R_1 and R_2 have different strategies for tracking the flows mapped to these gates. R_1 stores the classification states as in the previous scenario. However, the criterion for using gate *e* differs. Instead of bandwidth as in the previous example, R_1 might use a cost metric (e.g. money to pay to network 3) to decide which flow is important enough to justify the usage of gate *e*. R_2 does not limit the usage of gate *e* nor *f* and does not store any classification states.

The routes calculated are similar to the previous scenario. The main difference is in the policy for selecting gates in R_1 and R_2. In addition to the previous scenario, R_2 shows the benefit of knowing a broader set of gates available for a link. Depending on the requirements for a route, R_2 can decide to use *e* or *f*. For example, it can return the route [[*d*], [address(FN3)], [*f*]] for flows with no QoS requirements. Gate *e* can be used by R_2 without having to signal to FN_3. Furthermore, FN_3 does not have to know the details about flows and can just follow the gate numbers given in a packet. This reduces the load of the router providing FN_3.

C. Combined scenario

Both gate types can be combined in a single scenario. Such a scenario can be constructed by merging the two scenarios shown before. This combined scenario has four gates between FN_3 and FN_4 representing different QoS functions. In such a scenario, R_1 would have two options (since it does not know all gates) for a route from FN_3 to FN_4:

- Gate *b*: The usage is limited by bandwidth already reserved by R_1 for other flows. Through proper management of R_1, a minimal bandwidth can be guaranteed.
- Gate *e*: The usage is limited by the cost network 1 is willing to pay for a flow (if it is charged by network 3). A certain amount of bandwidth cannot be guaranteed. However, the delay is minimized.

Which gate to choose, depends mainly on the requirements for a flow and the requesting entity.

D. Discussion

Neither network 1 nor network 2 knows about the techniques used by network 3 to provide the QoS. These implementation issues are hidden by the abstract gate description for the routing and by the gate number used for the forwarding.

In all scenarios, FN_3 does not store any classification states and does not know which flows are mapped to its gates by network 1 and 2. It delegated the decision to these networks. However, the states required to enforce the characteristics of the gates remain in network 3. Thus, even though network 3 does not know which and how many flows are mapped to a gate, it can enforce that the combined traffic does not use better QoS than requested. Another benefit of this delegation is reduced signaling overhead. In particular, no signaling messages from network 1 or 2 are required to inform FN_3 about a new mapping.

The routes calculated by network 2 show an important case, which is not supported by MPLS and IP today. While the route [[*d*], [address(FN_3)]] can be implemented with MPLS handling the explicit route segment and IP handling the destination segment, a subsequent explicit route segment ([*c*] in the example) is not directly supported. In IP, the ingress router doing the IP forwarding has to have some classification states, which link a packet to the subsequent explicit route segment. However, FoG moves this state to other routers and thus reduces the number of states to be maintained by the ingress router.

The main use case of FoG consists of a network that would like to sell some degree of QoS to its customers (its own end users and other networks). Thus, deploying FoG in order to implement a best effort network provides only limited advantages compared to IP. However, the degree of deployment is critical for QoS scenarios, too. The delegation of states and decisions cannot be done only by network 3, since it requires the support of network 1 and 2. Consequently, a partial deployment in today's Internet might not benefit from these two features. However, the more networks support FoG, the better the exploitation of the advantages. A migration strategy for introducing FoG to existing networks depends mainly on the legacy systems, which should be supported. For example, MPLS might be

 ISBN: 978-1-61208-211-0

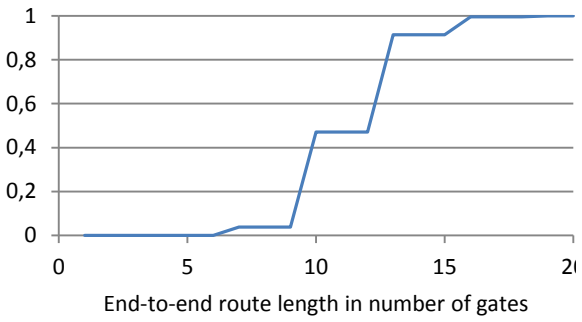

Figure 5. CDF of route lengths for scenario with 5000 nodes.

integrated by representing LSP as gates. However, suitable migration strategies are a subject for future work.

V. IMPLEMENTATION & SIMULATION RESULTS

We have implemented the FoG architecture in a Java-based discrete event simulator. It can be switched to an emulator mode that handles events in real time. The FoG emulator includes interoperability solutions combining FoG and IP based networks [13]. FoG applications specify their QoS requirements directly via an interface and FoG reacts accordingly. The interface bases on the GAPI defined in [8].

For evaluating FoG in the context of QoS management, the route length of explicit route segments is a specific concern. The more hops a packet has to travel, the more gate numbers are required and the longer the header. In order to estimate the FoG protocol overhead, the length of explicit routes for large-scale scenarios has been analyzed.

The network used for evaluation should match the characteristics of today's Internet but be smaller in size in order to reduce simulation time. Therefore, the network has been generated with the GLP algorithm implemented in BRITE [14] and the parameters derived in [15]. Therefore, the graph has similar characteristics to the real world Internet graph on the autonomous system level. The scenario consists of 5000 nodes and 12437 links between them. In addition, a different graph generated with the default parameters of BRITE (5000 nodes and 8974 links) was used for simulation. Since the results do not differ significantly, only the results from the first graph are presented.

The analysis is based on the total explicit route lengths of 6000 connections between randomly chosen FoG nodes. Figure 5 shows the cumulative distribution function of route lengths. Since each intermediate node uses three gate numbers and each end node uses two gate numbers in order to encode its routing decision, only specific route lengths such as 4 and 7 are possible. An end-to-end route contains 12.1 gate numbers in average. If each gate number is encoded in one byte, 91% of all routes remain below the size of an IPv6 address. The average number of hops between two FoG nodes $L = 3.7$ matches the expectations for the Internet [16].

VI. RELATED WORK

QoS for networks has a long research history. A survey about today's approaches is given in [17]. As discussed in the introduction, IntServ [2] and DiffServ [5] can be used to provide QoS. However, they do not support the movement of states.

MPLS uses routes comparable to the explicit route segment. In combination with IP, some use cases can be supported. However, Section IV.D points out important cases where the combination of IP and MPLS does not allow the movement of classification states. Furthermore, FoG does not require a standardization of gate numbers as required for IP's TOS field values in an inter-network scenario [18].

Other forwarding approaches using a stack of locally valid numbers to describe routes, like PARIS [19], Sirpent [20] or Pathlet [10], already introduce the split between forwarding and routing. In PFRI [9] the numbers are even globally unique in order to enable the end host to specify a loose source route based on links. An entry in a forwarding table represents virtual [10] or physical [21] next hops. Some (esp. the older) approaches are more related to intra-networks. Pathlet [10], the newest one, deals specifically with policy issues in inter-network routing. However, QoS and other application requirement aspects are not discussed in detail. Only Pathlet [10] mentions QoS but does not describe any details about how to integrate IntServ and DiffServ and a network protocol.

QoS protocols, like RSVP or NSIS [22] are able to signal QoS requirements. Either of these protocols or similar approaches are suitable to signal the setup of gates.

Other proposals for new inter-network architectures focus more on the overall architecture and do not address scalability of state information, e.g., NewArch [23], IPC [11], RNA [24].

VII. CONCLUSION AND OUTLOOK

In this paper, we have presented the Future Internet architecture "Forwarding on Gates" (FoG). It uses a network protocol, which provides the capability to explicitly define a route, use the destination address plus requirements for a route or a combination of both. This enables the movement of classification states between routers. IntServ and DiffServ are merged by introducing QoS functions, which are represented by directed gates in the FoG architecture. Routes can be defined by using the gates without knowing about their implementation. The protocol enables the flexibility to move classification states from the router implementing a QoS function to other routers, which take over the mapping of flows to QoS functions. This delegation of the mapping decisions reduces the amount of required signaling messages.

Based on three use cases, the setup of gates in IntServ, DiffServ and mixed scenarios is described. Although the route length is dynamic, the protocol overhead remains low. A protocol simulation in a large-scale network with 5000

 ISBN: 978-1-61208-211-0

nodes showed that 91% of the routes are shorter than an IPv6 address. The results presented in this paper show the flexibility of FoG in providing QoS in a scalable way. It indicates that the FoG architecture seems to be a promising basis for a Future Internet.

In the future, we plan to develop deployment and migration strategies from today's network to FoG. Furthermore, we will use route repair techniques known from MPLS to evaluate the robustness of FoG routes against link and node failures.

ACKNOWLEDGMENT

This work is funded by the German Federal Ministry of Education and Research under the project G-Lab_FoG (code 01BK0935). The project is part of the German Lab [25] research initiative.

REFERENCES

[1] Cisco Systems, "Cisco Visual Networking Index: Forecast and Methodology, 2010–2015", white paper, 2011, http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360.pdf [retreived: July 2012].

[2] R. Braden, D. Clark, and S. Shenker, "Integrated Services in the Internet Architecture: an Overview," IETF, RFC 1633, June 1994.

[3] C. Deleuze and Serge Fdida, "A scalable IntServ architecture through RSVP aggregation," Networking and Information Systems Journal, vol. 2, 1999, no. 5-6, pp. 665-681.

[4] B. E. Carpenter and K. Nichols, "Differentiated service in the Internet," Proc. IEEE, vol. 90, no. 9, pp.1479-1494 , 2002.

[5] S. Blake et al., "An Architecture for Differentiated Services," IETF RFC 2475, Dec. 1998.

[6] Y. Bernet et al., "A Framework for Integrated Services Operation over DiffServ Networks," IETF RFC 2998, Nov. 2000.

[7] X. Masip-Bruin et al., "The EuQoS System: A solution for QoS Routing in Heterogeneous Networks," IEEE Communications Magazine, Vol.45 No.2, pp. 96-103, February 2007.

[8] F. Liers, et al., "GAPI: A G-Lab Application-to-Network Interface," 11th Würzburg Workshop on IP: Joint ITG and Euro-NF Workshop "Visions of Future Generation Networks" (EuroView2011) , Würzburg Germany, August 2011.

[9] K. L. Calvert, J. Griffioen, and L. Poutievski, "Separating Routing and Forwarding: A Clean-Slate Network Layer Design," In proceedings of the Broadnets 2007 Conference, pp. 261-270, September 2007.

[10] P. B. Godfey, I. Ganichev, S. Shenker, and I. Stoica, "Pathlet Routing," In proceedings of SIGCOMM 2009, pp. 111-122 August 2009.

[11] J. Day, I. Matta, and K. Mattar, "Networking is IPC: A Guiding Principle to a Better Internet," In Proceedings of ReArch'08, Article no. 67, Madrid, Spain, December 2008.

[12] A. Reitzel, "Deprecation of Source Routing Options in IPv4, IETF," Internet-Draft, August 29, 2007.

[13] F. Liers, T. Volkert, and A. Mitschele-Thiel, "Scalable Network Support for Application Requirements with Forwarding on Gates," Demo at 11th Würzburg Workshop on IP: Joint ITG and Euro-NF Workshop "Visions of Future Generation Networks" (EuroView2011) , Würzburg Germany, August 2011.

[14] A. Medina, A. Lakhina, I. Matta, and J. Byers, "BRITE: an approach to universal topology generation," In IEEE MASCOTS, pp. 346–353, Cincinnati, OH, USA, August 2001.

[15] H. Haddadi, D. Fay, S. Uhlig, A.W. Moore, R. Mortier, A. Jamakovic, and M. Rio, "Tuning Topology Generators Using Spectral Distributions," In proceedings of SIPEW, pp.154-173, 2008.

[16] CAIDA, "Comparative analysis of the Internet AS-level topologies extracted from different data sources," http://www.caida.org/~dima/pub/as-topo-comparisons.pdf [retreived: July 2012].

[17] D. Vali, S. Paskalis, L. Merakos, and A. Kaloxylos, "A Survey of Internet QoS Signaling," IEEE Communications Surveys & Tutorials, Volume 6, Fourth Quarter, pp. 32-43, 2004.

[18] Cisco Systems, "Implementing Quality of Service Policies with DSCP," http://www.cisco.com/application/pdf/paws/10103/dscpvalues.pdf [retreived: July 2012].

[19] Israel Cidon and I. S. Gopal, "PARIS: An approach to integrated high-speed private networks," International Journal of Digital ans Analog Cable Systems, pp. 77-85, 1988.

[20] D. R. Cheriton, "Sirpent: a high-performance internetworking approach," In proceedings of ACM SIGCOMM '89: Symposium proceedings on Communications architectures & protocols, pp. 158-169, 1989.

[21] H. T. Kaur, S. Kalyanaraman, A. Weiss, S. Kanwar, and A. Gandhi, "BANANAS: An evolutionary framework for explicit and multipath routing in the Internet." In Proc. ACM SIGCOMM 2003, pp. 277-288, FDNA Workshop, Aug. 2003.

[22] R. Hancock et al., "Next Steps in Signaling (NSIS): Framework," IETF, RFC4080, Jun 2005.

[23] D. Clark, K. Sollins, J. Wrolawski, D. Katabi, J. Kulik, X. Yang, R. Braden, T. Faber, A. Falk, V. Pingali, M. Handley, and N. Chiappa, "NewArch: Future Generation Internet Architecture," Technical Report, 2003, http://www.isi.edu/newarch/iDOCS/final.finalreport.pdf [retreived January 2012].

[24] J. Touch and V. Pingali, "The RNA Metaprotocol," Proc. IEEE International Conf. on Computer Comm. (ICCCN), pp. 1-6, Aug. 2008.

[25] German Lab Homepage, http://www.german-lab.de [retreived: July 2012].

 ISBN: 978-1-61208-211-0