The Forwarding on Gates Architecture: Flexible Placement of QoS Functions and States in Inter-Networks

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

***Abstract*—A main driver of Future Internet applications and services is Quality of Service (QoS). Current Internet technologies provide no suitable QoS support for end-to-end connections due to several drawbacks of IntServ and DiffServ. In this article, we propose the "Forwarding on Gates" (FoG) architecture, which answers the QoS questions by the help of a new inter-network architecture. It applies its own new network protocol, which was designed to handle IntServ and DiffServ in an integrated way. FoG supports resource reservations for QoS guarantees in IntServ scenarios and prioritized traffic in DiffServ scenarios as well as a combination of both. As core advantage, the QoS support of FoG works in a scalable way by allowing a network to move QoS states and delegate decisions about the QoS usage to the entities demanding for QoS. This article describes the architecture, its network protocol, and solutions for interoperability with current networks. The evaluation includes theoretical descriptions of network configurations for a use case not supported by IP. Moreover, simulations show that the protocol overhead is comparable to IPv6, although packets can select QoS explicitly. Measured routing graph sizes for various setups show the flexibility of the FoG architecture.**

***Keywords*—Future Internet; network protocol; architecture; QoS; routing graph; interoperability.**

I. INTRODUCTION

The Future Internet will be faced with much more applications requiring Quality of Service (QoS) than today's networks. In the broad field of Future Internet research, we focus on the applied protocol of the network layer. In [1], we started to describe an inter-network architecture that supports QoS support in a flexible and scalable way. In this article, we give more details about our inter-network architecture "Forwarding on Gates" (FoG) such as its incremental routing process and possible interoperability with IP. Moreover, we present recent evaluation results for its routing.

Applications requiring QoS support will be the main driver for protocols in future networks. The most important application is video streaming, which already stresses today's networks, especially the Internet. Forecasts predict that in 2015 about 62% of the traffic in the Internet will be video data [2]. For live video streams, as required for remote medical operations and for football games, QoS is required. Due to the large number of hosts and connections in the Internet, scalability is the crux for QoS.

For IP (both version 4 and 6), add-ons called IntServ and DiffServ have been developed in order to tackle QoS requirements in the Internet. However, both have pros and cons. The IntServ approach [3], with its signaling protocol "Resource Reservation Protocol" (RSVP), provides end-to-end QoS by introducing states on each intermediate router, which is passed by a flow. According to [4], RSVP is used to distribute states for classification, scheduling and reservation. The classification state defines how incoming packets are mapped to flows. With RSVP, such a mapping consists of a source address, a destination address and a protocol number (and optionally port numbers). A scheduling state defines how flows are handled. For example, on a node a flow can be assigned to an own prioritized packet queue for the outgoing hardware interface. Finally, signaling states represent management information, e.g., authentication data and timers. For each flow, an intermediate node requires one set of the described states. Due to memory limitations, such an approach causes scalability problems for scenarios with many flows [5].

DiffServ [6] was developed with main focus on scalability. It introduces a small set of QoS classes, which are used inside networks. Each QoS class defines a type of service and requires scheduling and signaling states. Thus, in comparison to IntServ, 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. For a DiffServ network, edge routers of a network store and handle the classification states in order to map incoming packets to network internal QoS classes. The classification states represent the rules for this mapping. Since most interfaces with incoming traffic transport multiplexed flows, e.g., multiple TCP connections over the same Ethernet link, the classification is mainly done by (more or less deep) packet inspection. For example, protocol numbers, port numbers and even packet sizes are used for such classifications.

Besides their single application, IntServ and DiffServ can also be used in a combined way in order 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 [7, 8]. However, the scalability problem of IntServ now appears at the ingress routers. They have to store the classification states per flow. Since the amount of scheduling and signaling

states remains limited due to the limitation of possible DiffServ classes, the distribution of classification states is the main problem. In order to maintain the states, signaling is required. The processing load of handling these messages increases the burden on a network. In the past, proposals focused on reducing the number of flows, e.g., through aggregation [4].

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 required use cases. Therefore, we focus on a network protocol enabling the movement of classification states and the delegation of decisions between routers. Our solution is suitable both for IntServ and DiffServ scenarios. It is further 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 system. It enables the handling of both QoS approaches in a single mechanism.

The remainder of this article 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. Subsequently, Section V shows possibilities for interoperability with current networks. Afterwards, Section VI shows our recent evaluation results from protocol simulations and Section VII shows how FoG is classified within related work. In the end, the main results of this work are summarized and an outlook about future steps is given.

II. Forwarding on Gates Architecture

As fundamental design aspect, the "Forwarding on Gates" (FoG) architecture separates the forwarding from the routing. It splits them into two logical components, which encapsulate their 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 [9]. Both are linked via a route definition. The routing component specifies a route and the forwarding component forwards packets along this route. The authentication component is the third logical component of FoG. It checks the authentication of the sender of signaling messages in order to secure access to management functions. This authentication component is the basis for authorization decisions and accounting for QoS provisioning. Figure 1 depicts these components and their interactions.

For the following discussion, we introduce the term *QoS function*, which generalizes QoS provisioning regardless of the underlying QoS architecture. A QoS function represents the setup, which is 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, e.g., "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 [10], our architecture has some more features. It reduces forwarding table sizes [11], enables routers to choose their address format [12], hides addresses from applications, and supports various intra-network techniques. However, they are shared with other approaches from related work (see references) and are not in the focus of this article.

FoG applications specify their QoS requirements explicitly via an interface and FoG reacts accordingly. The interface is based on the "G-Lab Application-to-Network Interface" (GAPI) defined in [9].

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 in the form of a graph. Hosts and routers are represented by one or more vertices, which are called *forwarding nodes* (FNs). Edges between forwarding nodes are called *gates* and represent uni-directional (virtual) 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 to a *gate number*. The gate number is always unique to the scope of the forwarding node, to which the gate belongs to. Each FoG packet includes a header, which describes explicitly the order of gates the packet has to pass.

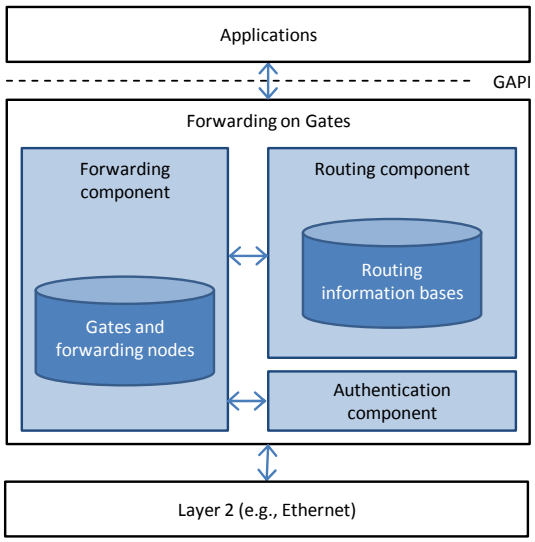

Figure 1: Logical components of the FoG architecture

Details about such a route, and how a 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 again.

The forwarding component informs the routing component about available gates and forwarding nodes to enable route calculations based on this information. However, gates that are not intended for other data flows (either connection based or connectionless) can be hidden in order to exclude them from the routing calculations.

B. Routing component

The interaction between the forwarding and routing is most important for FoG. Routes define how both are interacting. Whenever a route ends and the destination is not reached, the forwarding has to contact the routing component for the next (partial) route leading closer to the destination.

FoG routes are defined as stacks of segments. Two types of segments are possible:

- Explicit route segment: a stack of gate numbers
- Destination segment: the name or address of the destination, if a route is a partial one. The format and length depend on the routing and might range from DNS names to IPv4/6 addresses. Additionally, the QoS requirements are included. They describe the desired application requirements, which have to be considered by further route calculations.

A forwarding node uses the topmost segment of the route for its forwarding decision. If it is an explicit route segment, the forwarding node extracts the topmost gate number. This number is used for the following lookup for the next outgoing gate. In the simplest implementation, the gate number is an index representing the position of the outgoing gate in a stored gate vector. The packet passes this outgoing gate with a route, which is shortened by the used gate number. If the route segment is empty, it is removed from the route and the forwarding node proceeds with the next segment. If this is a destination segment, the forwarding node has to contact the routing component in order to know the next partial route towards the packet destination. For such a request, the forwarding node itself is the starting point for the route calculation. In general, the architecture does not restrict the format of the name or address, which are used in order to describe the packet destination. An implementation of the routing component can operate with its own format. If the destination segment includes QoS requirements, they are also included in the route request. Otherwise, best-effort routing based on hop costs is provided. The request result from the routing component is added to the route in the packet header by the forwarding component and the forwarding node will re-start its forwarding procedure once again. If there are no more route segments, the packet has reached its destination.

Destination segments can also be combined. In such a case, a destination segment is not necessarily the last segment in a route. It might define only the ingress router to an AS and a subsequent explicit route segment defines the rest of the route to the destination host. Theoretically, multiple destination segments in one route are possible. This allows for an *incremental routing* process, which is comparable to loose source routing of IP. Due to security considerations [13], policies can limit the utilization of this routing method. Since such limitations are mainly important at AS borders, multiple destination segments could be used within intra-networks. For example, this could be used to select the route through a local network.

In general, the subset of a network that is known to a routing component is a connected graph, which represents the nodes and links 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, which are required to specify the route explicitly. Such abstract connectivity information can be represented by gates without gate numbers. They represent connectivity enabled by an unknown set of forwarding nodes and gates. 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 each distant intermediate router, which has to be used in order to reach the destination.

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

The routing component can be implemented in multiple ways and with various protocols. We have developed a hierarchical approach for the implementation of the FoG routing component. It supports QoS and is called “Hierarchical Routing Management” (HRM). It clusters proactively the network at different hierarchical levels where each cluster has its own coordinator instance. The higher such a coordinator is located in the hierarchy, the more abstract is its network topology view. HRM uses these coordinators for distributing aggregated topology information among the physical nodes. As a result of this, each node knows the next hop for every existing destination without having global knowledge about the entire network topology. Further details about the approach are described in [14]. Additionally, we have already showed [15] the qualitative advantages of HRM for QoS demanding applications.

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 the 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 article, the authentication component is not described in more detail.

III. FOG PACKETS

The FoG packet structure is shown in Figure 2. 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:

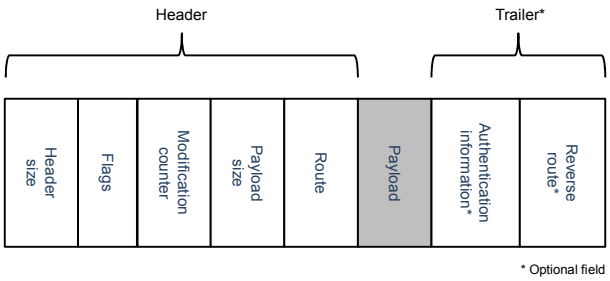

Figure 2: FoG protocol packet structure

- Header
  - *Header size in bytes:* This field is required to access the payload as subsequent versions of the FoG protocol specification could add more fields to the protocol header and before the payload.
  - *Flags indicate:*
    - 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*, which is used to avoid infinite routing loops due to invalid gate setups
  - *Payload size* in bytes
  - *Route* for the packet

- Trailer
  - *Authentication information* (including a size field, which stores the length of the authentication information in bytes)
  - *Reverse route* for answers: This field includes the reverse route as it was recorded by already passed forwarding nodes.

Each route starts with a length field followed by a stack of route segments. Each stack entry contains the following fields:

- Segment type indication, where two types are allowed:
  1. An *explicit route segment*: This is a stack of gate numbers, defining explicitly a sequence of gates, which have to be passed by the packet.
  2. A *destination segment*: It contains the name or the address of the desired destination, and the requirements for the remaining route to this destination.
- Segment length in bytes
- Segment content with variable size: It contains the gate numbers of an explicit route segment or an address plus requirements of a destination route segment.

Each forwarding node processes the route of a packet as shown in Figure 3. 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 an attached socket to an application, 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 as locally unique identification for addressing one of its outgoing gates. If there is an outgoing gate with this gate

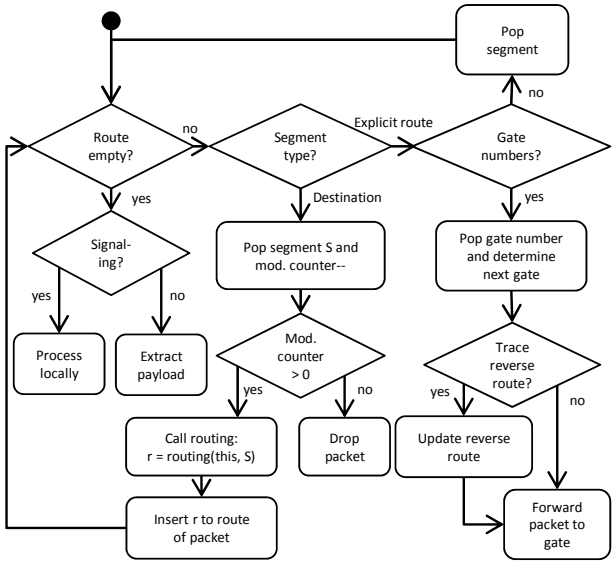

Figure 3: Forwarding node procedure without error cases

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 starts again. 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. Based on the received request data, the routing component calculates a route from the current forwarding node to the destination. The requirements given in the destination segment are used as requirements for the route. The resulting route from the routing component is added to the route of the packet by the forwarding component and the forwarding procedure is started again.

In order to reduce the impact of routing loops, a modification counter is decremented by one each time the route is changed in a way that loops might occur. As a result of this, a routing loop can be limited by forwarding only packets with a modification counter above 0 and dropping all other packets.

To allow a receiver of a packet to reply to the sender without knowing its exact address, 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 can be asymmetric 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 add a destination segment to the reverse route instead. Such a reverse route can be used by the receiver of a packet to reply to the sender. The main benefit of using such reverse routes instead of an address is twofold. First, routing requests for reply packets are avoided and second, addresses for request-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 a unique and routable address. If a reply with a traced backward route is received by a client, it knows the route the received request packet has taken. 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.

Section II.B describes that a destination segment of a route may define only an intermediate node of the route because not all network details (e.g., gate numbers) are known. Theoretically, multiple destination segments in one route are possible, which is comparable to loose source routing of IP. This is utilized in the use cases presented in the next sections.

Due to clarity reasons, Figure 3 does not show the error cases, which cause the drop of a packet. Example error cases are invalid formatted packet headers, invalid segment types, and data packets that routes end at forwarding nodes that do not represent a connection end point.

IV. USE CASES

Based on the FoG architecture and its network protocol, this section describes three different use cases showing the provisioning of QoS functions ranging from IntServ to DiffServ and 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.

Figures 4 and 5 show three networks with network 3 providing QoS functions in form of gates to networks 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 labeled with small letters. Each network includes not only the forwarding component but also the routing component R_i as defined by the architecture. The routing components are depicted as extra round boxes with their network view inside.

The scenarios assume that at least one node within each network runs an instance of each component type. For simplicity reasons, each network can be seen as a network with just a single node running all three components. In real world deployments, edge nodes typically have to host instances of the forwarding component. There may be more instances of the forwarding component on core nodes, if a network uses FoG also as intra-network technology. The locations of the instances of the routing component differ depending on their implementations. An example may be a central routing instance within a network with proxies on edge nodes.

The following description further assumes that each component can be implemented and focuses on the architectural level. The protocols used by components, e.g., the routing component, depend on their specific implementation. More details about our implementation are given in Section VI. However, there may be other ways to implement the same architecture.

In the following, gate numbers are represented by italic letters. Routes and route segments are encapsulated with square brackets. For example, [[*b*]] is a route (outer brackets) with a single explicit route segment (inner brackets), which contains only the single gate number *b*.

A. IntServ gates

Figure 4 shows a scenario, in which networks 1 and 2 have requested QoS functions from network 3. Depending on the implementation, they might have used a simple request/response signaling procedure. Network 3 has set up one gate for each request with the gate numbers *b* and *c*. As stated in Section II.A, these numbers have to be unique only in the scope of FN_3. Thus, network 3 is allowed to select freely the numbers *b* and *c* according to the needs of the components implementation. Networks 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

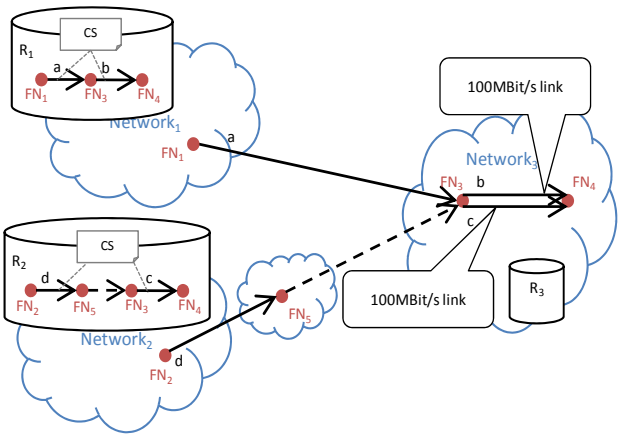

Figure 4: Gates representing IntServ reservations

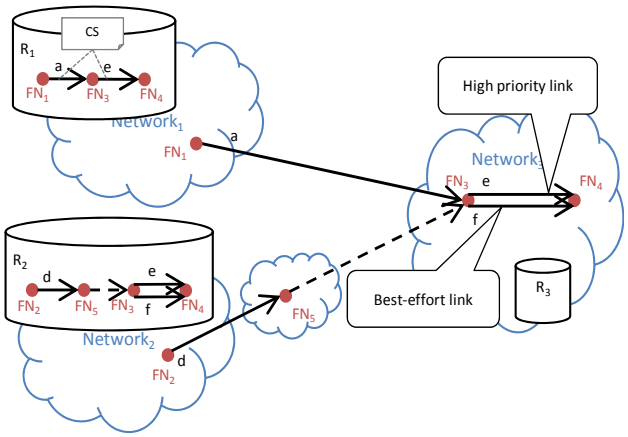

Figure 5: Gates representing DiffServ classes

enforce these QoS functions. However, the classification states are not stored by FN_3 but have been moved to networks 1 and 2, respectively. Their routing components know about gates *b* and *c*, and handle the decision about which flows are mapped to these gates.

If network 1 wants to establish a flow, the entity that is responsible for flow creation (e.g., a control node or a node at the network edge) starts sending a signaling message with a route, which contains only one destination segment containing the destination address 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 enough capacity is available. 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 up this number in its list of outgoing gates. Afterwards, 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*]]. This forwarding node extracts (and removes) *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.

For network 2, the process is similar. However, the route required to reach FN_3 is different. 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 has to be used. FN_5 receives a packet with [address(FN_3)] as topmost segment, which triggers an additional route request at FN_5.

B. *DiffServ gates*

Figure 5 depicts a scenario, in which 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 networks 1 and 2 have slightly different views on this situation. R_1 only knows *e* and R_2 knows 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 metric, 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* and *f* and does not store any classification states. It basically treats both as virtual best-effort links.

The calculated routes of this scenario are similar to the previous scenario. The main difference is the applied 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 without 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 hosting FN_3, *e*, and *f*.

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 the bandwidth, which is already reserved by R1 for other flows. By proper management of R1, a minimal bandwidth could be guaranteed.
- Gate e: The usage is limited by the costs, which 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 has to be chosen depends 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 QoS. These implementation details are hidden from the routing, based on the abstract gate description, and from the forwarding, based on the used gate numbers.

In all scenarios, FN_3 does not store any classification states and does not know which flows are mapped to its gates by networks 1 and 2. It delegates 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 a reduced signaling overhead. In particular, no signaling messages from networks 1 or 2 are required to inform FN_3 about new mappings.

The routes calculated by network 2 indicate an important case showing the difference between FoG and a combined MPLS / IP solution. The route [[*d*], [address(FN_3)]] could be implemented by the help of MPLS, handling the explicit route segment, and IP, handling the destination segment. However, a subsequent explicit route segment ([*c*] in the example) is not supported by MPLS and IP. 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 provides 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 also critical for QoS scenarios. The delegation of states and decisions cannot be done only by network 3, since it requires the support of networks 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 is 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 integrated by representing each "label switching path" (LSP) by gates.

V. Interoperability with current Networks

In the previous section, we discussed how FoG handles QoS functions. However, the architecture and its gates enable much more in-network functions. From an abstract point of view, gate numbers represent decisions of the routing, which are executed in the forwarding. Interpreting them as indices in a gate vector of a forwarding node is the simplest case. In general, gate numbers can be used to move information from an intermediate AS to the hosts at the border of the network or up-stream ASs without telling them about that shift. However, the usage of gate numbers is not limited to this.

A. Using layer 2 addressing

A gateway could store the MAC address of the destination node in a route. The forwarding would not need to create gates for all nodes, which are connected to an Ethernet domain. It would just use the MAC address given in the route. This enables stateless gateways, which do not have to store an address mapping (e.g., between IP and MAC addresses). The shift is transparent, since others see a list of gate numbers and do not need to know the meaning of each number. If the network has to prevent others from guessing MAC addresses, the routing can encrypt the address with a key, which is only known by the forwarding component. Based on this, the forwarding component can check whether a MAC address, given in a route, has actually been provided by the official routing component. In general, the resulting scheme, which is used to encode the routing decisions, can be adapted to the level of security, which is required for the network.

B. Network policies

Knowing the representation of decisions and how they are secured, introduces dependencies between the routing and forwarding component. In an inter-network scenario, this requires both components being operated by the same provider. Fortunately, the incremental routing process can be used to ensure this. By not announcing gates, others are forced to involve the forwarding of an AS in the route calculation, as shown in use case 1. Thus, ASs have the opportunity to insert self-generated route segments into the route. How these route segments encode the decisions for the forwarding instance is solely up to the AS

C. Protocol tunnels

Since gates hide the implementation of the QoS-aware data transport, any technology (e.g., MPLS) can be used. Therefore, gates can be interpreted as tunnels transporting packets through networks. By constructing explicit routes, tunnels are concatenated. Such a tunnel semantic can be implemented by having "tunnel gates". In particular, such gates are useful for hiding intermediate structures and to

shorten route descriptions in packets. For example, this approach can be utilized in order to implement interoperation with IP based networks.

Figure 6 shows an example scenario consisting of two FoG networks, which are connected via an IP based transit network. If node A of network FoG_1 wants to send packets to node D, located in network FoG_2, it can use the route description [[*1, 5, 4*]]. This directs packets along gate 1, 5 and 4. Gate 5 is located on FN_1 and implements the desired tunneling function by splitting each FoG packet into payload fragments, which are headed by an additional fragmentation header. These fragments have to fit into IP packets and are sent via the IP based network interface of FN_1 to the IP based network interface of FN_2. For this purpose, gate 5 has to know the IP address of FN_2. FN_2 detects such tunneling packets by checking the "protocol" field of the IP header from received packets for a FoG specific ID. As a result of this, FN_2 is able to reconstruct the original FoG packets and continues their processing as described in Section III in order to forward them towards their destination in the FoG network.

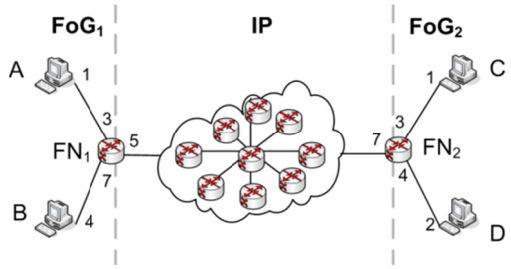

Figure 6: Two FoG networks are connected via an IP transit network

D. Gateway interoperability with IP

The goal of gateway interoperability is to provide a transparent mechanism for direct data exchange between a FoG and an IP based network. Figure 7 shows an example where applications on node A and B exchange directly data via FN_2 with applications on the IP based node C.

For addressing purposes on the IP side, both layer 3 addresses and layer 4 port numbers are used. Therefore, such a data exchange has to use TCP, UDP or SCTP (or any other transport protocol, which uses port numbers) on IP side.

If an *IP client* wants to send data to a *FoG service*, a dedicated gateway application has to be started before on the gateway node (FN_2 in Figure 7). The application owns a network server socket towards the IP network and receives IP based packets from IP clients. The application transforms these packets into FoG packets and sends them towards the service providing node in the FoG network. For this purpose, the application has to know a static mapping from its network socket (providing the service end point towards the IP network) to a FoG service name. For each new IP based client, the application stores an additional mapping between a dedicated FoG connection and IP status data, which allows a reverse mapping for answer packets from the FoG service.

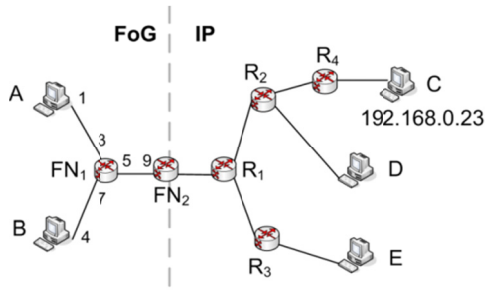

Figure 7: Direct connection between a FoG and an IP based network

For a transmission from a *FoG client* to an *IP based service*, a dedicated gateway application is used. Similar to the previous paragraph, it is located on FN_2. Each time a FoG client starts a communication to an IP based service, such an application instance is created. The FoG client sends its packets towards the gateway node. For addressing the correct receiver at IP side, the route in each packet includes a special destination route segment, which describes the IP receiver. For each FoG connection, the gateway application has to store a mapping from a FoG client to an IP based destination. This is also used for deriving the receiver at FoG side for each answer packet from an IP service.

VI. Evaluation

The applicability of FoG to combine various types of QoS for our use cases was shown in the previous sections. In the following, we present simulation results analyzing the scalability of FoG for large networks. The analysis focuses on one type of gates such as prioritized relaying. Thus, the results are general and can be interpreted easily for situations with more gate types by combining individual analyses for each gate type.

We measured three important metrics:

- Length of explicit routes: In order to evaluate the overhead induced by the FoG packet header, we measured the length of explicit end-to-end routes. This overhead is compared to the overhead of IP.
- Size of the routing graph required by a FoG node to store information about networks: As shown by the previous use cases, the size of the routing graph stored by the routing component depends on the number of gates known from others. We measured the graph size for different setups and compare them to the situation for source routing and IP networks.
- Number of request for (partial) routes: Since the knowledge of the routing components can vary, the incremental routing process may have to call the routing component more or less often. We measured the number of requests in order to approximate the routing overhead.

For the functional evaluation and measurements, we have implemented the entire FoG architecture in a Java-based

discrete event simulator called "FoGSiEm". 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 [16]. The software supports Windows, Linux and OS X and is available as open-source [17]. It includes the gateway interoperability as described in Section V.D with client and server sockets on FoG and IP side. We demonstrated this feature for web browsing and video streaming [18].

A. Simulation setup

The evaluation relies on simulations of large inter-networks. We derived the simulated networks from generated graphs modeling the Internet on the level of autonomous systems.

1) Network graph

The network used for evaluation matches the characteristics of today's Internet but has a smaller size in order to enable simulations. Its network graph has been generated with the GLP algorithm implemented in BRITE [19] and the parameters derived in [20]. Therefore, the graph has similar characteristics as the real world Internet graph on the autonomous system level. It consists of 5,000 nodes and 12,437 links between them. In addition, a different graph generated with the default parameters (α=0.45, β=0.64) of BRITE (5,000 nodes and 8,974 links) was used for simulation. Since the results do not differ significantly, only the results from the first graph are presented.

Each node of a graph represents an autonomous system. In the simulation, such a node is represented by one forwarding node per edge and one forwarding node representing the core of an autonomous system. The edge forwarding nodes are linked to the core forwarding node with a star topology, which represents the internal connectivity within the autonomous system. Each link between autonomous systems is represented with a link between two edge forwarding nodes.

2) Announcing gates to neighbors

The use cases discussed in the previous sections comprises only a small excerpt from a whole network. For the evaluation of the routing graph sizes, we need to define how many gates are known to the routing components of each FoG node of a large network. In the following, the algorithm used to configure the routing component instances used by our simulation is described.

We assume that FoG nodes join groups. Those group members trust each other and exchange information about gates with them. More specific, all members of a group announce their gates and gate numbers to all other members of the group. They are allowed to reuse these gates in the incremental routing process of Section II.B.

Groups are defined by clustering the nodes of a network. Clusters are generated by choosing cluster heads randomly at the start of a simulation. Nodes that are no cluster head join the cluster represented by the nearest cluster head. If two or more cluster heads are available within the same distance, one is chosen randomly. If no node decides to become a cluster head, a single node is selected randomly by the simulation.

Within a cluster, routing components exchange their knowledge. With nodes outside of a cluster, only abstract information is exchanged. This abstract information enables nodes to determine the shortest path to a destination but does not include any gate numbers. Thus, this information is comparable to the information exchanged in IP networks, e.g., between BGP entities.

The clustering is a possible and simple way to define the announcement policy. In reality, nodes will follow a more complex scheme based on the business plans of their operators. Our approach reduces the complexity but includes typical setups of today's networks as shown later on.

B. Overhead due to explicit route

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 is the header. In order to estimate the FoG protocol overhead, the length of explicit routes for large-scale scenarios has been analyzed.

1) Results

The analysis is based on the lengths of explicit routes of 100,000 connections between randomly chosen FoG nodes. Figure 8 shows the empirical distribution function of the 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 11.95 gate numbers in average. The average number of hops between two FoG nodes $L = 3.65$ matches the expectations for an Internet-like graph [21].

2) Discussion

Figure 8 shows the distribution of the route lengths in number of gate numbers. For a comparison with the address length of IP, an encoding of the routes is required. If a gate number, the route length field, the route segment length field, and the route segment type field from a FoG header (cp. packet format in Section III) are encoded with a single octet, a route with 13 gate numbers requires 16 octets in total. Thus, such a route would require the same number of octets

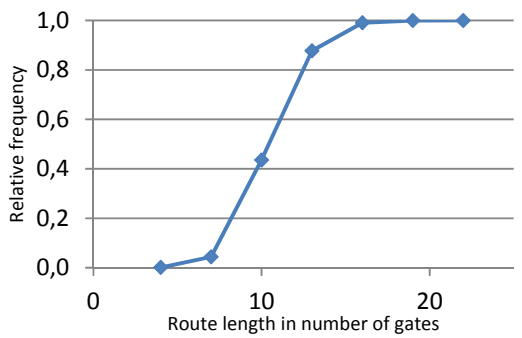

Figure 8: Empirical distribution function of end-to-end route lengths

as an IPv6 address. According to Figure 8, 87% of the routes have 13 gate numbers or less. Thus, most routes have a length, which is shorter or equal to an IPv6 address.

Since the setup requires three gate numbers to cross one node, the encoding with one octet per gate number seems to be realistic (in other words: three octets per hop). Thus, the overhead induced by transporting explicit routes seems to be acceptable in comparison with IPv6.

In average, only half of the gate numbers are transported over links, because gate numbers are successively removed from the packet header by the forwarding component. Since FoG may trace a reverse route, which increases the packet size again, the average route length mentioned above includes both routes. Thus, two routes with a total average size of 12 gate numbers replace the source and destination addresses of an IP packet. Even if the encoded size of a gate number is doubled to two bytes, 87% of the FoG packets contain less overhead than an IPv6 packet.

C. Routing graph size

In order to evaluate the scalability of the routing component, we simulated different setups of FoG's routing by varying the knowledge base of the routing components. Therefore, we cluster a network to model groups of trusted nodes as described in Section V.A.2. There are two extreme cases of the clustering:

- There is just a single cluster including all nodes. All network elements know the entire network graph and, thus, can use all gates for constructing routes. Such nodes can calculate complete end-to-end routes. This is equivalent to source routing.
- Each node forms its own cluster. All nodes know only their own gates. Additionally, they know the other network parts in an abstract way in order to find a shortest path. This setup is comparable to the operations of IP, since routing is done in a hop-by-hop manner.

In our simulations, a node knows the gates and forwarding nodes of the cluster it belong to and can calculate only partial routes for the cluster. For the construction of an end-to-end route, FoG has to combine multiple partial routes as described by the incremental routing process presented in Section II.B. Basically, an ingress node of a cluster calculates the route through its cluster, since it has the required knowledge.

1) Results

The results shown in the following are average results derived from 10 simulation runs with randomly chosen cluster heads. Error bars indicate the min/max results of the 10 runs. Each run contains 25,000 flows, which exist between randomly chosen nodes.

Figure 9 shows the number of gates and forwarding nodes (edges and vertices) of the detailed routing graph a node has to store. Figure 10 shows the same numbers for the abstract graph, which does not include gate numbers. For example, R_2 in Figure 4 has 4 vertices and 2 edges in its detailed graph. In its abstract graph, R_2 has one gate (dotted one) and its start and end forwarding nodes. Since a node has to store both graphs, Figure 11 shows the average graph sizes (vertices plus edges) of both graphs and the sum of both.

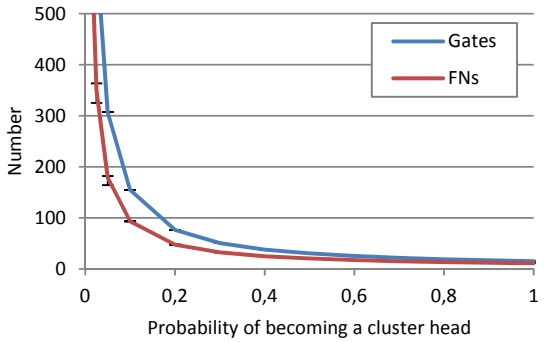

Figure 9: Number of gates and forwarding nodes (FNs) for detailed graph. Maximum values for probability zero: 76k gates and 30k FNs

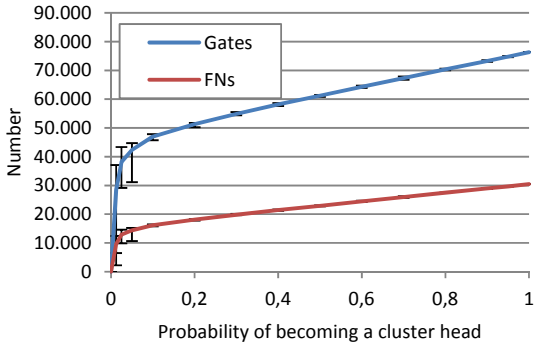

Figure 10: Number of gates and forwarding nodes (FNs) for abstract network graph.

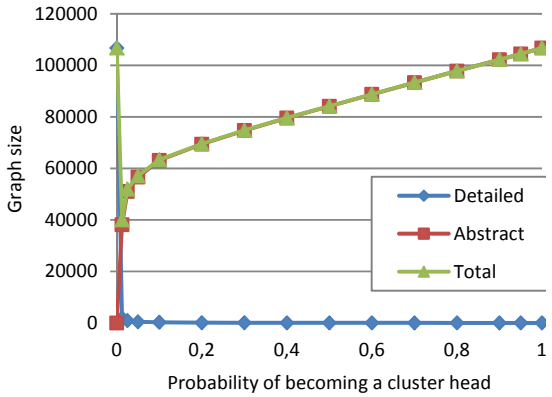

Figure 11: Average routing graph sizes

The x-axes of these three figures show the probability of a node to become a cluster head at the start of a simulation. In the source-routing setup (probability is zero) on the left hand side, each routing component knows the entire network with all gates. As expected, the graph for such approaches is very big. For hop-by-hop setups (probability is one), the amount of detailed information is significantly smaller as shown on the right hand side. Each node knows only the gates it hosts and remote gate numbers are not known. However, the abstract graph is rather big, since a lot of clusters and gates connecting them are known. In total, a routing component in a network configured in a hop-by-hop manner has to store slightly more information than a routing component in a source-routing network.

2) Discussion

The evaluation shows that there are a lot of configurations beside the extreme cases: source- and hop-by-hop routing. They require smaller overall routing component graphs. For example, the average routing component size for a probability of 0.025 with approximately 52k elements is half of the average sizes of the two extreme cases. From the perspective of the use cases, these intermediate configurations seem to be the more realistic ones. Some nodes request gates from remote peers and their routing components have the knowledge about these gates. They can combine them to routes for their data flows. These nodes are not interested in all gates as in the source routing configuration. They delegate decisions about route parts, which are "unimportant" for them, to others by using the incremental routing process. For the important parts, they exploit FoG's partial routes in order to define these route parts explicitly.

In relation to BGP, the results represent a worst-case scenario, because the nodes store information about all links. A BGP peer would filter the information according to its policy (e.g., only shortest path) and not announce all information to its peers.

D. Number of routing requests

The overall overhead of the routing is not only determined by the graph size. It also depends on the number of route calculations within this graph. Thus, we measured the average number of requests required to setup one end-to-end route. Figure 12 shows the results that are based on 10 simulation runs and 25,000 flows per run.

1) Results

The x-axis of Figure 12 shows the probability that a single node decides to become a cluster head at the start of a simulation run as described in the previous section. If the probability is zero, the source-routing configuration induces a single route calculation per flow. Between zero and 0.1, the number of route requests increases disproportionately. Above 0.1, the number continues to increase nearly linearly. If the probability is one, the hop-by-hop configuration requires 4.7 requests on average. This number includes one request per hop (3.7 hops on average as mentioned in Section V.B) plus a last calculation at the destination in order to determine the forwarding node within the destination node.

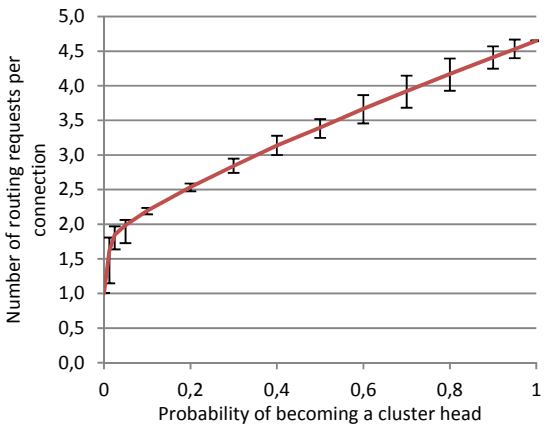

Figure 12: Average number of requests for (partial) routes in order to construct one end-to-end route.

2) Discussion

A request has to be answered by a routing component with a partial route towards the destination. This answer has to be calculated by an algorithm based on the graphs analyzed in the previous section. Thus, the computational overhead depends on the number of requests, the graph sizes and the applied algorithms. If the algorithm is fixed and its runtime complexity depends on the graph size, a trade-off between the number of requests and the graph size is required. In our example, we used the Dijkstra for each calculation. Thus, a probability of 0.0125 provides an optimal trade-off with a graph size of around 40k and 1.6 requests.

However, such a trade-off depends on the algorithm. Thus, different algorithms may lead to different optimal trade-offs. For example, if a forwarding information base (FIB) is introduced, the runtime of the algorithm does not depend directly on the graph size anymore. We refrain from analyzing FIBs in this article, because they depend strongly on the distribution of addresses. Since FoG supports a variety of implementations of the routing component, neither the algorithm nor the addressing or the type of addresses are fixed. For example, FoG can operate with IP addresses, BGP or with HRM with its hierarchical addresses. Thus, the optimal trade-off depends on the implementation. Our analysis underpins the flexibility of FoG and the new opportunities for designing routing components.

VII. Related Work

The question about how to provide an adequate QoS for networks has a long research history. A survey about today's approaches is given in [22]. As discussed in the introduction, IntServ [3] and DiffServ [6] can be used to provide QoS. However, they do not support the transparent movement of QoS classification states among nodes as it is enabled by the FoG architecture. By combining MPLS with IP, many QoS use cases can be implemented. However, Section IV.D

points out important cases where the combination of IP and MPLS does not allow the movement of classification states. Furthermore, the FoG architecture does not require a standardization of gate numbers as it is required for the IP TOS field in an inter-network scenario [23].

In general, FoG implements a strict split of packet processing into forwarding and routing components similar to PFRI [10], which uses labeled "channels" and anonymous nodes to model the available connectivity of lower layers. The channel labels are globally bijective and are comparable to FoG gates representing lower layer connections. However, the PFRI channel labels are not comparable to gate numbers, since gate numbers in FoG are only bijective in the scope of a forwarding node.

Other forwarding approaches use a stack of locally valid numbers to describe routes, e.g., PARIS [24], Sirpent [25] or Pathlet [11]. In PFRI [10], these numbers are even globally unique in order to enable the end host to specify a loose source route based on links. Such entries in a forwarding table represent either virtual [11] or physical [26] next hops. In a FoG network, each gate has a locally unique number in the scope of a node. The number assignment is controlled by the corresponding parent node. Globally unique numbers are not mandatory in a FoG network. But server application must have globally unique addresses to be addressable by clients, which can be located everywhere in the entire network. Furthermore, FoG uses names for forwarding nodes and not gates, because we expect a network to have more links than nodes.

Some older architecture approaches [24, 25] are focused on intra-networks. Other proposals for new inter-network architectures focus more on the overall architecture and do not address scalability of state distribution, e.g., NewArch [27], IPC [12], RNA [28]. The Pathlet approach, as the newest one, deals specifically with policy issues in inter-network routing. However, QoS and other application requirement aspects are not discussed in detail. In the Pathlet descriptions, QoS is only mentioned but any details about how to integrate IntServ or DiffServ, and a network protocol are missing. The FoG architecture includes these aspects with respect to scalability. It is focused mainly on inter-networks. QoS path reservation protocols, e.g., RSVP or NSIS [29], signal QoS requirements. One of these protocols or similar approaches are suitable to trigger the setup of gates.

The combination of function blocks can either be done at design time (similar to Netlets [30]) or at runtime (similar to SONATE [31]). The architecture of FoG decides everything during runtime. It reacts dynamically on inputs from applications (e.g., QoS requirements) and network states (network policies and available routes). While SONATE [31] is focused on selecting and composing function blocks on end hosts, FoG can also be used to integrate function blocks residing on intermediate routers.

The impact of using heap structures for header information as proposed by the Role-based Architecture [32] has not been analyzed. The "roles" of this architecture are comparable to gates. However, RoleIDs (and, thus, role addresses) are not comparable to gate numbers because a RoleID contains an identifier for the type of function and not for a special instance.

VIII. Conclusion and Outlook

In this article, we presented the Future Internet architecture "Forwarding on Gates" (FoG). It applies its own network protocol in order to provide three different ways for selecting a route for packets: define the route explicitly, define the route indirectly based on the destination address plus requirements or – as third way – use 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. FoG enables the flexibility to move classification states from the router, which implements a QoS function, to other routers, which take over the mapping of flows to QoS functions. This delegation of mapping decisions reduces the amount of required signaling messages.

Based on three use cases, we described the setup of gates in IntServ, DiffServ and mixed scenarios. Although the route length is dynamic, the protocol overhead remains low. A protocol simulation in a large-scale network with 5000 nodes showed that 87% of the routes are shorter than an IPv6 address. Simulations of a basic version of the routing component showed the flexibility of FoG by using partial routes. Depending on the knowledg a node and its routing component have, FoG can be used for a trade-off between hop-by-hop routing (with many routing requests) and source routing (with only one routing request). FoG's routing component implementation can exploit this flexibility and can use new address types or routing algorithms.

The results presented in this article show the flexibility of FoG in providing QoS in a scalable way. It indicates that the FoG architecture is a promising basis for a network layer architecture that can replace IP in a Future Internet. Since a complete replacement of IP seems to be unrealistic, Section V outlined several interoperability solutions for partial deployment scenarios. They are similar to the interoperability solutions for IPv4 and IPv6. Thus, running FoG in parallel to IP is an deployment option as well.

In the future, we plan to use route repair techniques known from MPLS to evaluate the robustness of FoG routes against link and node failures. Additionally, we will evaluate the FoG architecture in comparison to other existing architectures for selected Internet-like scenarios. Furthermore, we plan to extend the existing qualitative evaluation [15] of the routing component infrastructure "Hierarchical Routing Management" [14] by measuring its quantitative advantages and management overhead for selected complex network scenarios.

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 [33] research initiative.

REFERENCES

[1] F. Liers, T. Volkert, and A. Mitschele-Thiel, "The Forwarding on Gates architecture: Merging IntServ and DivServ," Proc. of International Conference on Advances in Future Internet (AFIN) 2012, pp. 7-13, August 2012.

[2] 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, last access: December 2013.

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

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

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

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

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

[8] 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.

[9] 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.

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

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

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

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

[14] T. Volkert, M. Osdoba, A. Mitschele-Thiel, and M. Becke, „Multipath Video Streaming Based on Hierarchical Routing Management," Proc. of 27th IEEE International Conference on Advanced Information Networking and Applications (AINA), Barcelona/Spain, March 2013.

[15] T. Volkert and A. Mitschele-Thiel, "Hierarchical routing management for improving multimedia transmissions and QoE," Proc. of IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks (WoWMoM), San Francisco/USA, June 2012.

[16] 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.

[17] "FoGSiEm" on GitHub, web site: https://github.com/ICS-TU-Ilmenau/fog/wiki, last access: December 2013.

[18] T. Volkert and F. Liers, "Video transcoding and rerouting in Forwarding on Gates networks," 12th Würzburg Workshop on IP: ITG Workshop "Visions of Future Generation Networks" (EuroView2012), Würzburg, Germany, August 2012.

[19] 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.

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

[21] CAIDA, "Comparative analysis of the Internet AS-level topologies extracted from different data sources," http://www.caida.org/~dima/pub/as-topo-comparisons.pdf, last access: December 2013.

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

[23] Cisco Systems, "Implementing Quality of Service Policies with DSCP," http://www.cisco.com/application/pdf/paws/10103/dscpvalues.pdf, last access: December 2013.

[24] I. 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.

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

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

[27] 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, last access: December 2013.

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

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

[30] L. Völker, D. Martin, C. Werle, M. Zitterbart, and I. El Khayat, "Selecting concurrent network architectures at runtime," Proc. of the 2009 IEEE international conference on Communications, ser. ICC'09. Piscataway, NJ, USA: IEEE Press, 2009, pp. 2124–2128.

[31] P. Müller and B. Reuther, "Future internet architecture - a service oriented approach," it - Information Technology, vol. 50, no. 6, pp. 383–389, 2008.

[32] R. Braden, T. Faber, and M. Handley, "From protocol stack to protocol heap: role-based architecture," SIGCOMM Comput. Commun. Rev., vol. 33, no. 1, pp. 17-22, January 2003.

[33] German Lab, web site: http://www.german-lab.de, last access: December 2013.