

Performance Study of the NSIS QoS-NSLP Protocol

Mayutan Arumathurai^{*†}, Xiaoming Fu^{*}, Bernd Schloer^{*} and Hannes Tschofenig^{*†}

^{*}Institute of Computer Science, University of Goettingen, Germany,

Email : arumathurai, fu, bschloer@informatik.uni-goettingen.de

[†]Nokia Siemens Network, Helsinki, Finland, Email: hannes.tschofenig@nsn.com

Abstract—This paper presents an evaluation of the Quality of Service Signalling Layer Protocol (QoS-NSLP) of the NSIS (Next Steps In Signalling) protocol suite. The QoS-NSLP in combination with the NSIS Transport Layer Protocol (NTLP) is proposed by the Internet Engineering Task Force (IETF) as an alternative to the Resource reSerVation Protocol (RSVP). We describe our implementations of the software architecture, both on a network simulator and on a Linux implementation. Both implementations are used in a complimentary manner to illustrate the performance of the QoS-NSLP protocol. The results show the performance of QoS-NSLP with respect to resource consumption, packet processing time, session set up time, refresh interval and protocol overhead. Furthermore, we analyse the protocol performance during route change scenarios.

I. INTRODUCTION

The need to provide Quality of Service(QoS) guarantees in networks is imminent due to the increasing popularity of multimedia applications, like VoIP or Video on Demand. The IETF is currently developing the Next Steps in Signalling(NSIS) protocol suite, which consists of the Quality of Service NSIS Signalling Layer Protocol (QoS-NSLP) [1], the NAT/Firewall NSIS Signalling Layer Protocol (NAT-FW NSLP) [2] and the NSIS Transport Layer Protocol(NTLP) [3]. QoS-NSLP along with NTLP is expected to succeed the Resource reSerVation Protocol(RSVP) [4] to overcome its shortcomings [5] and also provide support for the use of transport layer security(TLS), sender and receiver initiation, QoS Models other than IntServ and DiffServ, incremental deployment, mobility among other things.

The QoS-NSLP Protocol is subjected to a comprehensive analysis with regard to its performance, thereby ensuring that it caters to its design objectives. Hence a detailed performance study of the QoS-NSLP protocol is presented in this paper. Though some of the results discussed in Section III-C are implementation dependent, we expect that the general behaviour of the protocol is proportional to the numeric values presented. Moreover, the implementation on the simulator has been used to get a thorough understanding of the protocol behaviour without being affected by hardware constraints.

This paper is structured as follows. Section II provides a short introduction to NSIS and the QoS-NSLP. Section III details upon the real-time implementation details and the results of the study on hardware constraints. Section IV discusses the implementation on a simulator and the results with respect to session setup time, refresh overhead, protocol overhead and adaptation to a route change scenario.

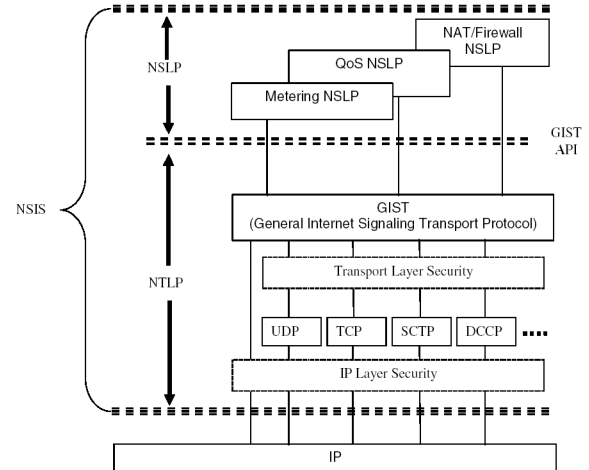


Fig. 1. NSIS framework [7]

II. AN INTRODUCTION TO THE NEXT STEPS IN SIGNALLING PROTOCOL SUITE(NSIS)

This section provides background information of the NSIS protocol suite [6] and explains the QoS-NSLP application in detail.

A. NSIS Framework

The IETF NSIS protocol suite [6] consists of the NSIS Transport Layer Protocol (NTLP) [3] and the NSIS Signalling Layer Protocol (NSLP) layers as shown in Fig. 1 [7]. The NSLP layer is placed above the NTLP layer and comprises of signalling applications, such as the QoS-NSLP, NSLP for metering [8] and the NSLP for NAT/firewall traversal (NatFW-NSLP) [2]. The NSIS protocol suite enables signalling applications to install, modify or remove control state in a network. This control state is related to the data flow that the application wants to deliver.

B. NSIS Transport Layer Protocol (NTLP)

The NTLP layer is commonly known as the General Internet Signalling Transport (GIST) [3]. GIST, the main building block of NSIS, is a soft state protocol. It is responsible for the discovery of signalling peers and delivery of signalling messages along the data path. As shown in Fig. 1, it reuses existing transport layer protocols, like TCP, UDP and SCTP. The NSIS Transport Layer Protocol operates between adjacent peers on a hop-by-hop basis whereas the end-to-end messaging

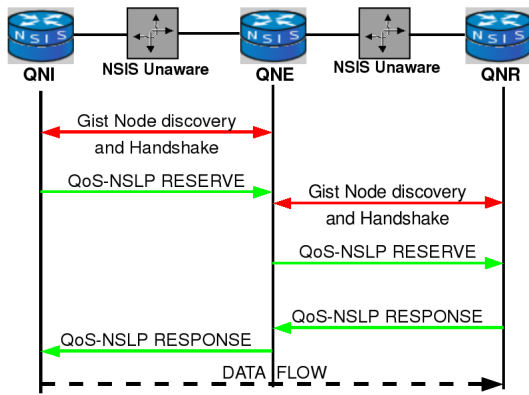


Fig. 2. Evaluation topology

is handled at the NSLP level. Intermediate nodes that are NSIS unaware or are not aware of the particular NSLP in question are ignored and hence the messages are forwarded without processing it further.

C. QoS-NSLP

The design concept of QoS-NSLP together with GIST is similar to the one of RSVP, but it provides additional functionality. Like RSVP, the QoS-NSLP also uses soft state for state management. The QoS-NSLP supports four Message Types: RESERVE, QUERY, RESPONSE and NOTIFY. The RESERVE message is used to create, refresh, modify or remove a QoS-NSLP reservation state. This is the only message that manipulates a reservation state. A QUERY message is used to collect information about available bandwidth on the reservation path. The RESPONSE message provides information about previously installed reservations. It could be used to signal the success of an installed reservation, to respond to a QUERY message, or to send error-specific information. A NOTIFY message is sent asynchronously, i.e., not in response to a previously received message, and typically indicates error conditions. It generally does not refer to any particular state or any received message.

Control information needed for state installation, removal or refreshes are carried in message-specific objects. Examples are sequence numbers, session IDs or objects requesting a response. The QSPEC object [9] describes information about the actual resource reservation, like bandwidth, token buckets or DiffServ Code Points (DSCPs). QSPEC parameters provide a pool of commonly used QoS parameters to define QoS models (QoSM) [10], such as IntServ, DiffServ, RMD etc. It is, however, possible for QoSMs to define additional QoS parameters if necessary.

Fig. 2 illustrates the main participants in a QoS-NSLP message exchange scenario. A QNE is an NSIS Entity (NE) that supports QoS-NSLP. The QNI is an end QNE node that initiates a reservation request whereas the QNR is the last QNE node on the data path. The QNR acts as the responder of a reservation request. To make a simple sender initiated reservation as shown in Fig. 2, the QNI creates a RESERVE

containing the required QoS parameters in the QSPEC object. The message is transported by GIST, in a hop by hop fashion to the QNR, the final destination of the message. In return, the QNR sends a RESPONSE message indicating the success of the reservation. For a receiver initiated reservation, as done by RSVP, three messages are exchanged: QUERY, RESERVE and RESPONSE.

III. REAL-TIME IMPLEMENTATION, TEST-BED SETUP AND EVALUATION

Our QoS-NSLP implementation is built on an implementation based on our existing GIST protocol implementation [11] [12] and is released as open source at [11]. One of the main objectives was to ensure that our implementation does not introduce any additional complexities to the efficient and scalable implementation of the GIST protocol [11]. In the following subsections we discuss the implementation overview and the quantitative performance of the QoS-NSLP protocol through benchmarks and show that the protocol scales well with the number of sessions.

A. Implementation Overview

During our implementation, we were satisfied with the ease with which we were able to design the QoS-NSLP layer and use the GIST-NSLP APIs to interact with the GIST layer. This was proof of the generic nature of the NSIS protocol suite, which is one of the main design objectives and also of the adherence of the GIST implementation [11] to the requirements of the protocol. The implementation is written in C++ and the main target platform is Linux and plugs into the GIST implementation [11]. The software design is object oriented and each protocol layer runs as a single-threaded process. The communication between the QoS-NSLP and NTLN level takes place via UNIX sockets.

The QoS-NSLP layer implementation consists of about 5,720 lines of code (1,360 for the core program including message parsing and socket operation, 860 for the Finite State Machine (FSM), 2,500 for the QoS-NSLP and QSPEC parsing, and about 1,000 lines for the Resource Management Function(RMF) [10]). The FSM is the main building block of the QoS-NSLP framework and performs almost all tasks concerning message processing, RMF and timer interaction. The QoS-NSLP state machine is implemented according to [13]. The states are maintained per session and contain information regarding the status of initiated signalling requests. Timers are used to maintain the lifetime of the states and to trigger the retransmissions of messages. Special attention was paid to the implementation of the timer handling and we used double-linked lists for fast processing of call-back functions while maintaining the consumption of system resources at a minimum. More details of this approach is given in [12]. The IntServ Controlled Load (CLS) QoS Model [14] is implemented for the RMF. CLS enabled traffic of a flow is comparable to an unloaded network offering best-effort service for that flow. The traffic characteristics are signalled as token buckets and the actual reservation is done using the Linux

Traffic Control interface. The test results shown in this paper are performed without the interaction of the Traffic Control interface that forms the bottleneck in our implementation. Testing the QoS-NSLP implementation together with the Traffic Control interface comes with a performance impact.

B. Test-bed Setup

The performance study was carried out on three low-end PCs performing the role of a QNI, QNE and QNR. They are connected as shown in Fig. 2. The low-end PCs were used to set up a test-bed similar to that described in [12] for the convenience of performance comparison. Testing was done with the following hardware-software configuration:

- VIA Samuel 2 CPU 533 MHz
- 256 MB RAM
- 100 Mbit Network Interface Card (NIC)
- Operating System: Ubuntu 4.0.3

The main goal was to study the scalability of the protocol and its impact on the three different role players. Comparisons of the performance of the QoS-NSLP protocol to the GIST implementation were also made. We assume that the QoS-NSLP signalling is assigned the highest priority even in the presence of background data traffic.

C. Performance Study

[12] presents a comparison between RSVP and GIST. Since QoS-NSLP simply adds to the GIST protocol, a comparison of the QoS-NSLP protocol to RSVP can be inferred between the results presented below and that of [12].

1) *Analysis of Control Packet processing time:* The processing time of the control packets in a router is measured by calculating the difference in timestamps from the time the packet arrives at a particular module till the time it leaves it. This analysis gives us an insight into how the processing time varies with increasing number of sessions. This measurement is useful to correlate the session set up time and also to model the protocol on a simulator accurately.

The processing time per control packet at the QNI, QNE and QNR was measured for processing a QoS-NSLP RESERVE and RESPONSE message for varying simultaneous sessions. It should be noted that the QNR does not receive any RESPONSE message and therefore does not have the results for the same. The session arrival rate was set at 100 sessions per second. We have performed this test under a low load scenario to avoid dropping of packets and to have a moderate waiting time in the queues.

From Fig. 3, it can be inferred that the processing time is independent of the number of sessions, except in the case of the QNI-RESERVE processing. This gives a good indication of the scalability of the implementation with increasing session numbers. The QNI requires more processing time for its RESERVE message (approximately 200-450 microseconds) because of the time it takes to process a new QoS request trigger. For this very reason we observe that the latency at the QNE and QNR for the RESERVE (approx 180-200 microseconds) is nearly the same, confirming that both these

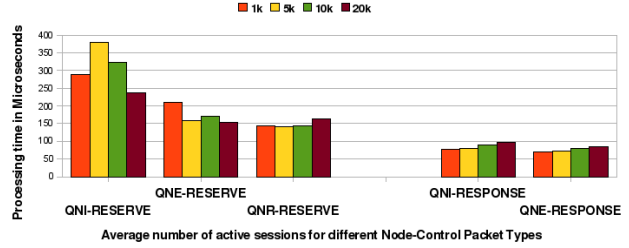


Fig. 3. Control Packet processing time at nodes for varying number of active sessions

nodes perform roughly the same operations. The processing time at the QNI and QNE for the RESPONSE (approx 80-100 microseconds) is roughly equivalent since these nodes process the RESPONSE in a similar fashion.

Interestingly, the processing time for control packets as shown in Fig. 3 was much higher when either the number of sessions setups (below 100) or the trigger for the session arrival (e.g. one session per second) were quite low. There are no concrete answers to this, but it could be due to the CPU entering and exiting idle state. The fact that the latency values are in microseconds should not be overlooked.

2) *CPU utilization and Memory consumption:* The number of active sessions a particular NSIS router can handle depends primarily on the CPU utilization and Memory consumption. We have evaluated the CPU utilization and memory consumption for a varying number of sessions at different nodes and the maximum number of sessions the nodes can support. In the experiments, the GIST refresh period was set to 60 seconds, the QoS-NSLP refresh period to 30 seconds and the new session trigger arrival rate at 100 sessions per second.

Fig. 4 and Fig. 5 illustrate a linear increase in total resource consumption with increasing number of sessions. They show that the QNE is the bottleneck due to its significantly higher resource consumption in comparison with the QNI and the QNR. This is due to the fact that the QNE has to play the roles of a sender and receiver simultaneously for the upstream and downstream signalling at the GIST layer.

At the QoS-NSLP layer, the QNE resource consumption is only slightly higher than that of the QNI and QNR. This is because of the similarity in the roles of these nodes at the QoS-NSLP level.

The GIST layer performs more work for session management in comparison to the QoS-NSLP layer. This is the reason for the comparatively high linear increase in resource consumption by the GIST layer to that of the QoS-NSLP layer.

X.Fu et. al [12] show that the resource consumption by the GIST layer is greater than that consumed by RSVP. The resource consumption of the QoS-NSLP layer together with the GIST is larger than the resource requirement for an equivalent RSVP session. But, this is justified since the QoS-NSLP provides more features. A maximum of 30,000 stable sessions was achieved with the addition of the QoS-NSLP layer, whereas Fu et. al [12] achieved a maximum of 45,000 stable sessions with only the GIST layer.

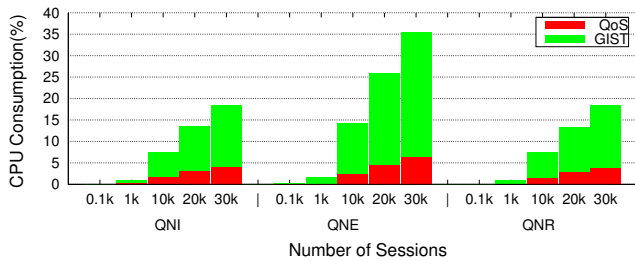


Fig. 4. CPU consumption

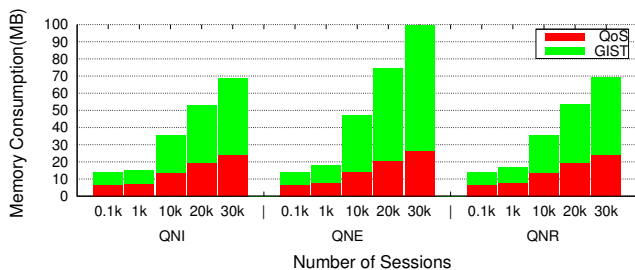


Fig. 5. Memory consumption

IV. IMPLEMENTATION AND EVALUATION ON A SIMULATOR

A. Implementation Overview

The implementation on the simulator involved the implementation of GIST and the QoS-NSLP. Having realised the benefits of using a modular and FSM based approach for the real-time implementation, we chose OMNeT++ 3.2 [15] along with INET(Version 20061020) [16] as our simulation tool. OMNeT++ is an object oriented discrete event network simulator. An OMNeT++ model consists of hierarchically nested modules that communicate by passing messages. It is highly modular, well structured, scalable and is based on C++. Another reason for choosing OMNeT++ as the simulator was that it has built in state machine support. It was therefore easier to implement the QoS-NSLP state machine and to perform debugging too. The implementation framework is similar to the framework used in the real-time implementation. The simulation implementation consists of approximately 5,000 lines of code (3,500 for Gist, 1,400 for QoS-NSLP). OMNeT++ provides support for debugging and obtaining the results.

B. Simulation setup

Our aim was to use the simulation tool to verify the scalability of the protocol, in terms of varying number of sessions and hops. For this purpose, we have used the backbone topology available in OMNeT++ (known as NTT) and have substituted all the routers with our NSIS capable routers. Fig. 6 gives an overview of the topology. The salient features of this topology are that:

- the number of nodes is 56
- the longest path is nearly 15 hops
- the link capacity is 0.6 Mbits/sec
- the delay is 15 ms

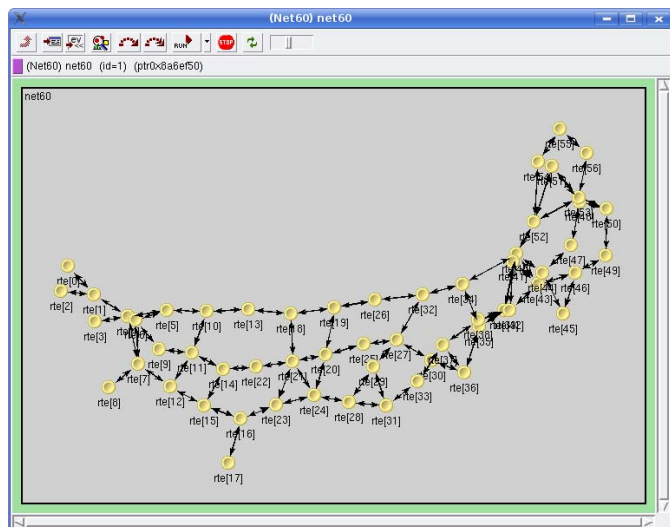


Fig. 6. Topology used in the simulator

- the arrival rate of request for reservation is based on a Poisson arrival rate with lambda of 0.7
- the life time of a session is based on a uniform distribution with a range varying between 30 and 60 seconds.

Our simulation-implementation was quite helpful in analysing the protocol, independent of its dependency on hardware. The simulator along with the test-topology had to be supported on one machine. Though we used a modern PC (DELL dual core P4) with 4 GB RAM, it was unable to take the load of simulating high number of sessions. It could support a maximum of about 5,000 sessions on a 2 hop network. Nevertheless, we were able to achieve the desired objective of using a simulator to evaluate the session setup time, refresh round-trip time, protocol overhead and route change scenario. Using this topology it was possible to study the impact of varying sessions and also varying hops.

C. Session set up time

The session set up time is the delay between the sending of the RESERVE to the arrival of the RESPONSE during the setting up of a session. It was measured at the QNI since the sessions originate from there. Theoretically this time would be the sum of the packet latency at the QNI, the QNE(s) and the QNR for the RESERVE and RESPONSE, GIST handshake, hops * Message association setup, and 2 * (no. of hops * link delay). We have used GIST in C-MODE (TCP based) and therefore the message association is done once per hop and is reused for other message exchanges.

Fig. 7 shows the average session setup time on the simulator for varying number of sessions and hops. On a per hop basis, it is observed that the connection setup time increases linearly with increase in the number of established sessions but only at lower values. At higher number of sessions it nearly remains a constant. The initial linear increase is due to the shift from a zero load to a low load network due to increasing session numbers. We can also observe that as the number of hops

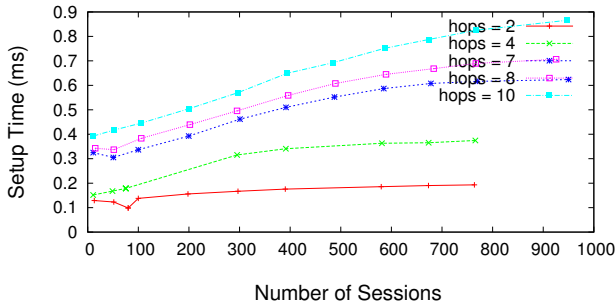


Fig. 7. Session setup time

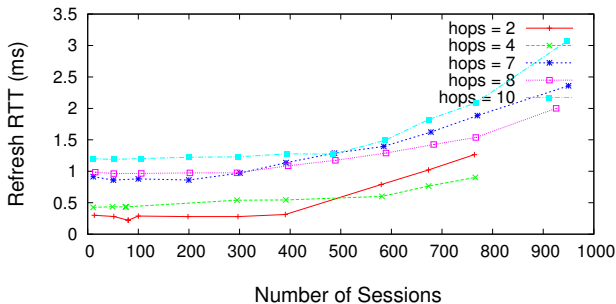


Fig. 8. Refresh interval

increase, the initial linear increase becomes longer. This is because the waiting time in queues increase as there are more messages in the network. this is due to a longer RTT and also higher load on the intermediate QNEs as shown in the hardware constraints that we mentioned in Section III-C.

D. Soft State Refresh Overhead

QoS-NSLP and GIST are soft state protocols. Refresh messages are necessary to keep the state alive. Larger refresh values incur less control traffic but result in longer recovery time from route failures. Smaller refresh values increase the amount of control traffic but decreases the recovery time from routing failures and reroute.

We have measured the refresh interval as the time it takes for the refresh RESERVE to traverse all nodes on a hop-by-hop manner till the QNR and the RESPONSE to arrive at the QNI. Fig. 8 shows the refresh-interval for varying hops and sessions. We can observe that the refresh-interval does not increase much at lower number of sessions and then there is a rapid increase as the number of sessions increase. This result also agrees with the per packet latency results in Section III-C where the processing time for packets is the same in a low load scenario.

The RESERVE messages sent as refreshes also introduce message overheads. The size of a refreshing RESERVE message is small compared to a RESERVE that is used to set up sessions, since it does not contain the full RESERVE message with the required QSPEC.

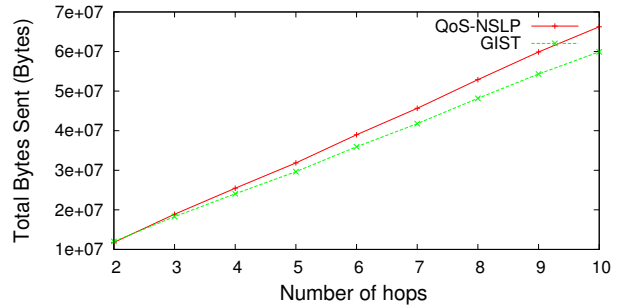


Fig. 9. Protocol overhead vs. Hops

E. Protocol Overhead

Messages are exchanged to setup sessions and to keep them alive. We have studied the amount of packets that are exchanged at the QoS-NSLP layer and at the GIST layer for C-MODE (TCP based) and presented them for an average of 300 sessions in Fig. 9.

It can be observed that as the hops increase the total message exchange in the network increases linearly. The bytes additionally required by GIST is not significant since there is only a single message association per hop. The message size at the GIST level is 116 bytes for QUERY, 152 bytes for RESPONSE, 116 bytes for CONFIRM and 72 bytes for DATA message overhead and at the QoS-NSLP level, it is 112 for RESERVE and 44 for RESPONSE. The GIST-DATA message is used to carry the QoS-NSLP messages.

F. Route Change

The NSIS protocol suite depends on the underlying routing protocol to select the data path and uses this selected path to send the signalling information. Route changes may occur due to a number of reasons, such as router failure, link failure, load balancing, ISPs switching to a different provider etc. The QoS-NSLP has to detect these changes and react efficiently to ensure that the new path the data traverses does not introduce a new hurdle to its own data or the existing data flows. The new path might require new states to be set up or some existing flows to be terminated. QoS-NSLP has to guarantee a certain resource allocation for every flow it admits. Therefore it becomes all the more important for it to adapt to route change scenarios and get back to a stable state. The data flow would be treated as Best-effort-Traffic till fresh states are set up on QNEs present in the new path.

Let us assume a scenario such as in Fig. 10 where the QNE (QNE-1) fails and the data traffic is redirected through a different QNE (QNE-2). We are not interested in a QNI or a QNR failure, since it would result in the expiration of the established states at other nodes due to a time-out caused by not receiving a REFRESH or a RESPONSE. According to our proposed scenario, the route failure at QNE-1, does not affect the number of active sessions at the QNI and the QNR, because the state lifetime is greater than the time it takes for our network to move from a transient state to a stable state.

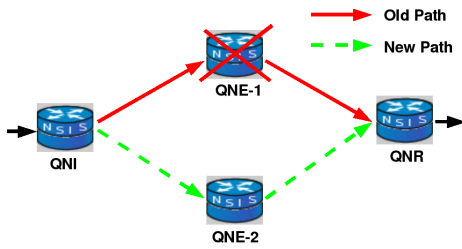


Fig. 10. Route failure topology

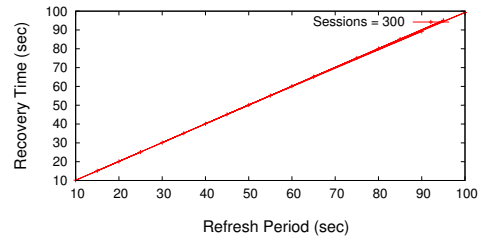


Fig. 11. Fault recovery time

Since these nodes are not affected by the route failure, data passing through these nodes still continue to receive the same treatment.

The recovery time after a route change depends on how long the underlying layer requires to reroute the data and on the time the GIST and QoS-NSLP take to recover. QoS-NSLP is either dependent on the GIST layer to notify it of a route change or on the change in the interface identifier of the messages it receives. GIST realises that the data path is traversing a different path when it sends a refresh QUERY for a session and receives a RESPONSE message containing a different next-hop node address. It then sends a network change notification to the QoS-NSLP layer. The QoS-NSLP layer on receiving this notification will proceed to send a full RESERVE to set up new states along the new data path.

Fig. 11 plots the time it takes for all sessions to be set up in the new QNE (QNE-2) for varying GIST refresh periods between 10 and 100. These values were obtained by varying the arrival rate, sessions dying rate etc and since the observed general behaviour was the same, we have showed one such result in Fig. 11. We can observe that the recovery time is proportional to the GIST refresh period. This is because the QoS-NSLP is dependent on the GIST refresh interval to recover from route change scenarios. Furthermore, though the GIST layer realises that there is a route change when it receives the first RESPONSE message (with a different next hop identifier), it reacts on a session to session basis. Thus every QoS-NSLP session will have to wait for its GIST refresh to take place to come to stable state behaviour again. This is not an ideal NSIS reaction to a route change scenario.

In case the data gets re-routed through nodes that are QoS-NSLP unaware (QNE-2 is NSIS unaware), the QoS-NSLP at QNI would still receive a network notification from the GIST layer and would send a non-refresh RESERVE message. This is not necessary since the next node is already maintaining states for the data flow.

V. CONCLUSION

We have performed an evaluation of the NSIS QoS-NSLP protocol and have measured the resource consumption, latency, session setup time, refresh overhead, protocol overhead and the reaction of QoS-NSLP to a route change scenario. The results show that the QoS-NSLP protocol meets its design requirements and is scalable with the number of sessions and hops.

Some concerns were raised regarding the reaction of the protocol to a route change scenario and its dependency on the GIST refresh interval. Moreover it was shown that the QNE tends to be a bottleneck. Further work on enhancements to enable the QoS-NSLP to react better to route changes is being investigated.

VI. ACKNOWLEDGEMENT

We would like to thank the anonymous reviewers for their helpful comments.

REFERENCES

- [1] J. Manner, G. Karagiannis, and A. McDonald. "NSLP for Quality-of-Service Signaling". draft-ietf-nsis-qos-nslp-16 (work in progress), 2008.
- [2] M. Stiernerling, H. Tschofenig, C. Aoun, and E. Davies. "NAT/Firewall NSIS Signaling Layer Protocol (NSLP)". draft-ietf-nsis-nslp-natfw-18 (work in progress), 2008.
- [3] H. Schulzrinne and R. Hancock. "GIST: General Internet Signalling Transport". draft-ietf-nsis-ntlp-16 (work in progress), 2008.
- [4] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. "Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification". RFC 2205 (Proposed Standard), September 1997.
- [5] J.Manner and X.Fu. "Analysis of Existing Quality-of-Service Signaling Protocols". RFC 4094 (Informational), May 2005.
- [6] R. Hancock, G. Karagiannis, J. Loughney, and S. Van den Bosch. "Next Steps in Signaling (NSIS): Framework". RFC 4080 (Informational), June 2005.
- [7] X. Fu, H. Tschofenig, H. Schulzrinne, A. Bader, C. Kappler, G. Karagiannis, and Sven Van den Bosch. "NSIS: A new Extensible IP Signaling Protocol Suite". IEEE Communications Magazine, 2005.
- [8] F.Dressler J. Quittek C. Kappler A. Fessi, G. Carle and H. Tschofenig. "NSLP for Metering Configuration Signaling". draft-dressler-nsis-metering-nslp-05 (work in progress), 2007.
- [9] G. Ash, A. Bader, C. Kappler, and D. Oran. "QoS NSLP QSPEC Template". draft-ietf-nsis-qspec-17 (work in progress), 2007.
- [10] A. Bader, Lars Westberg, G. Karagiannis, C. Kappler, and Tom Phelan. "RMD-QOSM - The Resource Management in Diffserv QOS Model". draft-ietf-nsis-rmd-13 (work in progress), 2008.
- [11] Open source NSIS implementation, University of Goettingen. <http://user.informatik.uni-goettingen.de/~nsis/>.
- [12] X.Fu et. al. "Overhead and Performance Study of the General Internet Signaling Transport (GIST) Protocol". IEEE INFOCOM, 2006.
- [13] X. Fu, B. Schloer, H. Tschofenig, and T. Tsenov. "QoS NSLP State Machine". draft-fu-nsis-qos-nslp-statemachine-06 (work in progress), 2007.
- [14] C. Kappler, X. Fu, and B. Schloer. "A QoS Model for Signaling IntServ Controlled-Load Service with NSIS". draft-kappler-nsis-qosmodel-controlledload-07 (work in progress), 2007.
- [15] OMNeT++, Discrete Event Simulation System. <http://www.omnetpp.org/>.
- [16] The INET Framework for OMNeT++. <http://ctieware.eng.monash.edu.au/twiki/bin/view/Simulation/INETFramework>.