Universität Karlsruhe (TH)
**Institut für Telematik**

TELEMATICS TECHNICAL REPORTS

# Analysis and Design of Mobility Support for QoS NSLP

Max Laier
mlaier@freebsd.org

February, 24th 2009

TM-2009-1

ISSN 1613-849X
http://doc.tm.uka.de/tr/

Institute of Telematics, University of Karlsruhe
Zirkel 2, D-76128 Karlsruhe, Germany

Universität Karlsruhe (TH)
Forschungsuniversität · gegründet 1825

# Analysis and Design of Mobility Support for QoS NSLP

Studienarbeit am Institut für Telematik
Prof. Dr. M. Zitterbart
Fakultät für Informatik
Universität Karlsruhe (TH)

von

cand. inform.
**Max Laier**

Betreuer:
Prof. Dr. M. Zitterbart
Dr. R. Bless
Dipl.-Inform. M. Röhricht

Tag der Anmeldung: 1. Juli 2008
Tag der Abgabe: 9. Dezember 2008

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Karlsruhe, den 9. Dezember 2008

# Contents

# 1. Introduction

Advanced use cases in the current Internet and—even more so—future extensions require signaling to establish IP resources and negotiate related service parameters such as Quality-of-Service reservations. The *Next Steps in Signaling (NSIS)* working group at the IETF is creating a generic framework for signaling solutions. One of the goals for this framework is the support of current and future mobility solutions as an important use case. Mobility adds some interesting chalanges which are different from normal operation. For *Quality-of-Service* signaling as an example, reservation parameters have to be renegotiated along the new path whenever a movement occurs, established reservations along the obsolete paths should be torn down so that the allocated resources are available to other clients again, and—depending on the new location of the mobile endpoint—the reservation might have to adapt to the new conditions. In addition, the address information of a network flow does not necessarily serve as a unique identifier that is valid for the lifetime of the connection. Instead, the address information is variable and might change with every movement. In order to operate properly in a mobile environment a signaling application must be aware of mobility events—such as handovers—and be able to react with an appropriate action.

The signaling protocols under development in the NSIS working group are divided into two layers:

1. The *Transport Layer* that implements basic transport facilities. The transport layer protocol is responsible for discovering other signaling nodes along a given path, establishing state with a signaling enabled next hop and exchanging messages with neighboring nodes. It also provides functions to authenticate and authorize neiboring nodes. Currently there is only one protocol defined for this layer: the *General Internet Signalling Transport (GIST)* [14]. Protocols in this layer are referred to as *NSIS Transport Layer Protocols* or *NTLP*.

2. The application specific *Signaling Layer* protocols build upon the functionality provided by the transport layer. These protocols are called *NSIS Signaling Layer Protocols* or *NSLP*. At this time there are draft protocol specifications for Quality-of-Service [8] and NAT/Firewall [19] NSLPs.

The working group has also produced a mobility draft document [13] that discusses the interaction and compatibility of these proposed protocols with mobility. This particular draft is the basis for the work presented herein.

## 1.1 Objectives of this Thesis

The aim of this work is to extend an existing *Signaling Application* that implements GIST and the QoS-NSLP with mobility awareness in order to perform the signaling operations outlined in the mobility draft [13]. Based on the required modifications an evaluation of the proposed signaling stack and its applicability to mobile scenarios will be elaborated. In addition this will provide some numbers on the general signaling performance and delay of service negotiation in mobile scenarios.

Mobility management is provided by MobileIPv6. This technology provides transparent mobility support in IPv6 networks and exercises most—if not all—challenges that come up when dealing with general IP mobility.

## 1.2 Structure of this document

The remainder of this document will firstly—in Chapter 2— introduce and describe the basic principles of MobileIPv6 and Quality-of-Service signaling as well as review other related work. The Analysis chapter discusses the main problems that need solving in light of the current state of the examined software and protocols. Chapter 4 provides theoretical solutions to the identified problems and considers alternatives. In addition this chapter also describes the actual implementation changes and additions. Chapter 5 provides experimental proof that the chosen design and implementation are sufficient to provide signaling for mobility scenarios. In addition it outlines and summarizes how and if the required design and implementation decisions impact on the current state of the NSIS protocol stack drafts. Finally—in Chapter 6—a short summary and further directions are given.

# 2. Background and Related Work

This chapter provides a short introduction to the technologies used. It also establishes some common terms used throughout the rest of the document. Most of the terminology is shared with the cited Internet drafts and standards so that readers already familiar with these works can proceed to the end of this chapter in Section 2.3 that discusses other related work.

## 2.1 Mobility with MobileIPv6

The foundation for mobility for this work is provided by *MobileIPv6 (MIP6)* with colocated *Care-of-Addresses (CoA)* as defined in RFC 3775 [5]. Figure 2.1 provides a high level overview of the architecture with additional annotations that are required further below.

The problem any mobility management needs to solve is that once a mobile node changes its location it also changes its IP address as the address encodes an end-system identification and network location at the same time. After an address change all existing connections from or to the old address are no longer valid. Applications with open connections for the old address will stall and have to re-connect with their peers.

With MIP6 a *Mobile Node (MN)* is assigned a *Home Address (HoA)* located in the home network that serves as a unique identifier for all connections from and to the MN. In addition the MN acquires one ore more *Care-of-Address (CoA)* at its current location. MIP6 provides a transparent mapping between these addresses and thus hides the actual location from a communication peer—called *Correspondent Node (CN)* in terms of MIP6—as well as applications running on the MN itself.

In order to realize the mapping on the wire there are two basic modes of operation: Tunnel- and Route-optimization mode.

- *Tunnel mode* is used when a CN has no support for MIP6 and during the initial contact. When a CN wants to talk to a MN it sends traffic to the MN's HoA located in the MN's home network. The IP datagram reaches the home

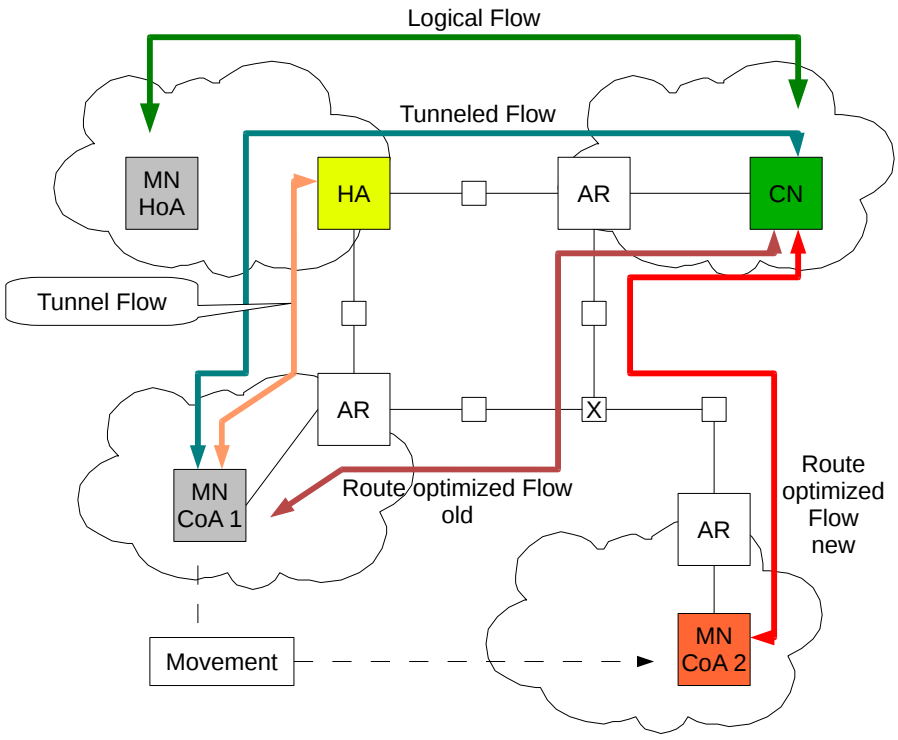


Figure 2.1: MobileIPv6—Basic operations

network by means of normal IP routing. If the MN is not available at the homelink a special service node within the home network—the *Home Agent (HA)*—intercepts the packet on behalf of the MN. Because the MN registers and updates its CoA with the HA, the latter knows the current location (i.e. the current CoA) of the MN and forwards the intercepted packet through a *tunnel* that is established between the HA and the MN at its CoA. In the opposite direction—if the MN wants to reply—it uses the same tunnel to send the packet to the HA that then forwards it on behalf of the MN using the HoA as the packet source. This is valid since the HA is located at the home link where the HoA is located. In addition, the MN can use the tunnel to the HA to initiate contact with yet unknown CNs.

- *Route-optimization* is used with MIP6 enabled CNs. The MN and CN establish a *Binding* between CoA and HoA. Once a Binding is in place the MN can send traffic directly from the CoA to the CN—using the optimized route between the current location and the CN. In turn, the CN will also send traffic directly to the MN's CoA. Special IPv6 options are used to differentiate route-optimized packets from normal traffic and both sides check for an active Binding before accepting such traffic. Section 2.1.1 provides a more detailed discussion of these operations.

As the MN moves to a new location and acquires a new CoA it updates existing Bindings and thus connected CNs will know where to send their traffic. There is also a Binding between the MN and its HA that ensures that the tunnel follows the MN to the new location.

Independent of the used mode of operation an application on either side of the connection always sees the HoA as local or peer address on the socket layer.

### 2.1.1 Logical and Actual Network Flows

In Figure 2.1 one can see that the use of MobileIP may result in creation of various flows that differ from the *logical flow* that is visible to the application. When we speak of a logical flow, we mean the flow originated from or destined to a MN's HoA. Only if the MN is at its home network the logical flow is identical to the *actual flow(s)* along which the traffic is sent. In tunnel mode there are two actual flows: The inner, *tunneled flow*, which can be seen as an extension of the logical flow after the HA has intercepted it, and the outer, *tunnel flow*, that carries the encapsulated tunnel packets. In route-optimization mode there is the route optimized flow where the HoA(s) in the logical flow are replaced by the MN's current CoA. In order to do signaling in these scenarios the signaling application (e.g. QoS NSLP) must be aware of the actual flow(s) to perform its task correctly.

### 2.1.2 MIP6 signaling and Datastructures

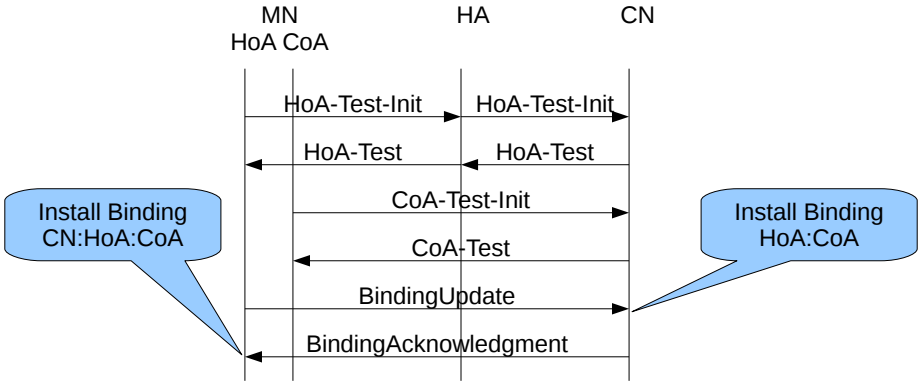


Figure 2.2: MobileIPv6—Binding Update Process

As mentioned earlier MIP6 performs signaling to inform the HA and MIP6-enabled CN of the MN's CoA. The process is outlined in Figure 2.2. It involves a series of tests to ensure that the MN is really in control of both the CoA and HoA to avoid the use of MIP6 in spoofing and amplification attacks. The MN takes tokens from those tests and generates a *BindingUpdate (BU)*. Once the BU is received at the HA or CN and the tokens match, the peer replies with a *BindingAcknowledgement (BA)*. The peer also installs a *Binding* in its *BindingCache*. The Binding stores the MN's HoA and CoA and allows the peer to perform route-optimized communication to the MN. After the MN has received the BA it installs its own Binding. This Binding is stored in the MN's *BindingUpdateList (BUL)* and contains the MN's HoA and CoA as well as the CN's address for which the Binding is valid. At this point the MN can also send route-optimized packets to the CN at this address.

Note from the above that the receipt of the BindingUpdate or BindingAck at the CN and MN respectively indicates the earliest time route-optimization to/from the new CoA is possible. Consequently, we will use these events as triggers for the QoS-Signaling to update any affected reservations. Also note that the BindingCache and BindingUpdateList contain all active Bindings at any time.

## 2.2 The NSIS architecture

Figure 2.3 gives an overview of the NSIS architecture. It shows the two layer architecture with the GIST NTLP at the bottom and various NSLPs on top. The NSLP

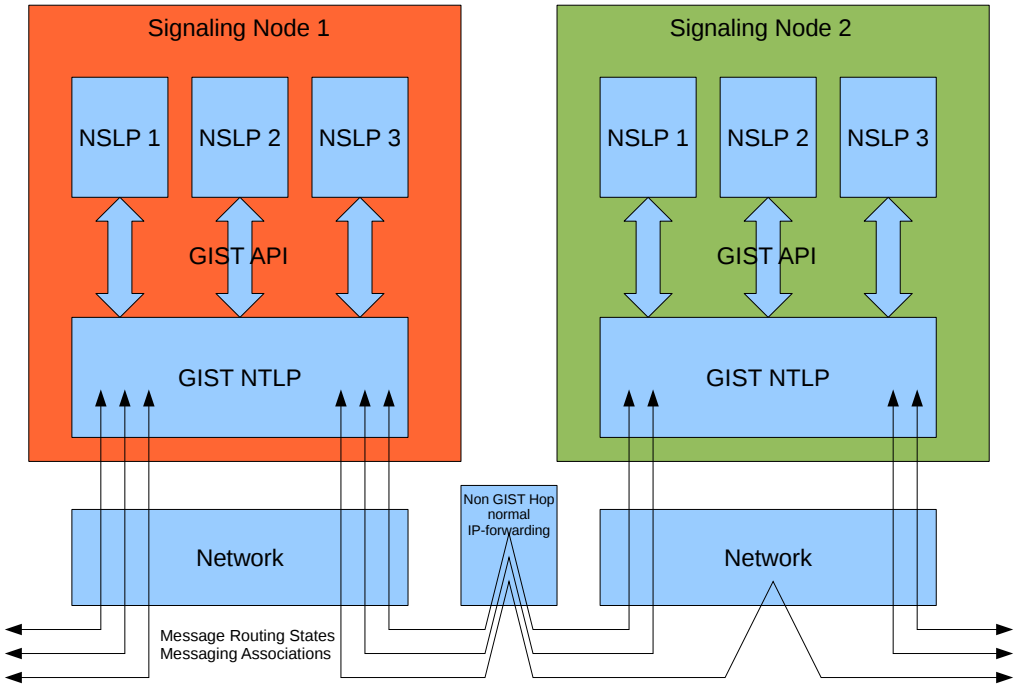


Figure 2.3: The NSIS architecture and layering

layer implements the actual signaling application, while GIST layer only provides common basic functionality required and used by all NSLPs. The NSLPs are responsible to manage end-to-end state, if required. Figure 2.4 depicts the process. GIST only holds state with its direct GIST neighbors. A direct GIST neighbor is not necessarily the next IP-hop, but can be several IP-hops away. Also, if the direct GIST-neighbor is not enabled for a certain NSLP it might be skipped for state installation for that NSLP.

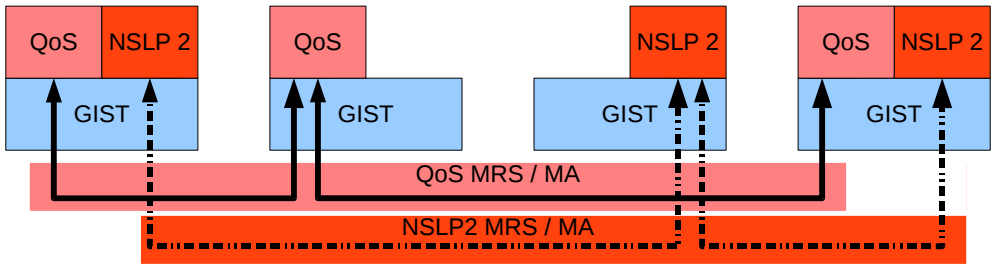


Figure 2.4: NTLP Hop-to-hop state

The GIST API, offered to the NSLPs, evolves around two basic functions to send and receive messages with some additional functions to manage state and receive information about network topology changes.

When an NSLP wants to send an initial signaling message for a flow it provides GIST with the following information:

**A *Session-ID*** to manage end-to-end state. GIST is oblivious of the meaning of the SID and forwards it verbatim inside the resulting messages.

**The *Message Routing Information (MRI)*** for the flow. This item contains a description of the flow: Source and destination address, protocol, source and

destination port, as well as additional information to classify the flow such as SPI or the IP6 Flow Label. We will call the sum of this information or a sufficient subset thereof the *Flow Identifier*. The MRI object that is used as the argument for the API call also selects a *Message Routing Method (MRM)*. Currently GIST defines three MRMs:

***Path-coupled (PC-MRM)*** where GIST attempts to send the signaling messages along the same path as the flow.

***Explicit-Signaling-Target (EST-MRM)*** where GIST will send the signaling message directly to the destination of the flow instead of establishing state along the path.

***Loose-End (LE-MRM)*** which is used for NAT discovery and proxy operations.

For the scope of this work we are only concerned with the PC-MRM.

**Transfer parameters** that tell GIST what kind of state to install, how many GIST- and IP-hops the message is allowed to travel and properties for the state GIST should install as a result of the message. The NSLP can select between unreliable, reliable and confidential transport and it is up to GIST how to best provide the requested mode selecting a suitable transport protocol (UDP, TCP, SCTP, TLS).

**The payload** for the signaling message itself.

Assuming PC-MRM and unreliable transfer, Figure 2.5 shows the API and signaling messages exchanged between the NSLP and GIST, and neighboring GIST nodes respectively. GIST starts operation by discovering its next neighbor on the path. It does so by sending a *GIST-Query* message to the flow destination. This Query is intercepted by the next GIST node that—in return—replies with a GIST-Response. The first GIST node completes the three-way-handshake with a GIST-Confirm. At this point both GIST nodes have installed a—so-called—*routing state* for the flow described in the MRI and the NSLP that initiated the exchange. Now the first GIST node can send the actual payload to the neighbor.

The receiving GIST instance informs the NSLP via a *ReceiveMessage* API call. The NSLP detects that it is not the final destination of the message by looking at the MRI and asks GIST to forward the message.

Again, the GIST node sends a Query to discover its next neighbor. This Query message should closely resemble packets of the flow and thus GIST attempts to send it with a spoofed IP-source address of the flow source. The GIST node can, however, choose to use one of its own addresses as IP-source address in the Query message. This *Signalling Source Mode* is instrumental in detecting legacy NAT hops. Otherwise, the address of the GIST node itself is also stored in the Query message's *Network Layer Information (NLI)* object so that the intercepting/receiving GIST node knows where to send the Response to. Once the routing state has been established the GIST node forwards the payload as requested.

The NSLP on the final hop decides to send back a reply on the NSLP level. In order to do so, GIST provides a *Source Identification Information Handle (SII-handle)*

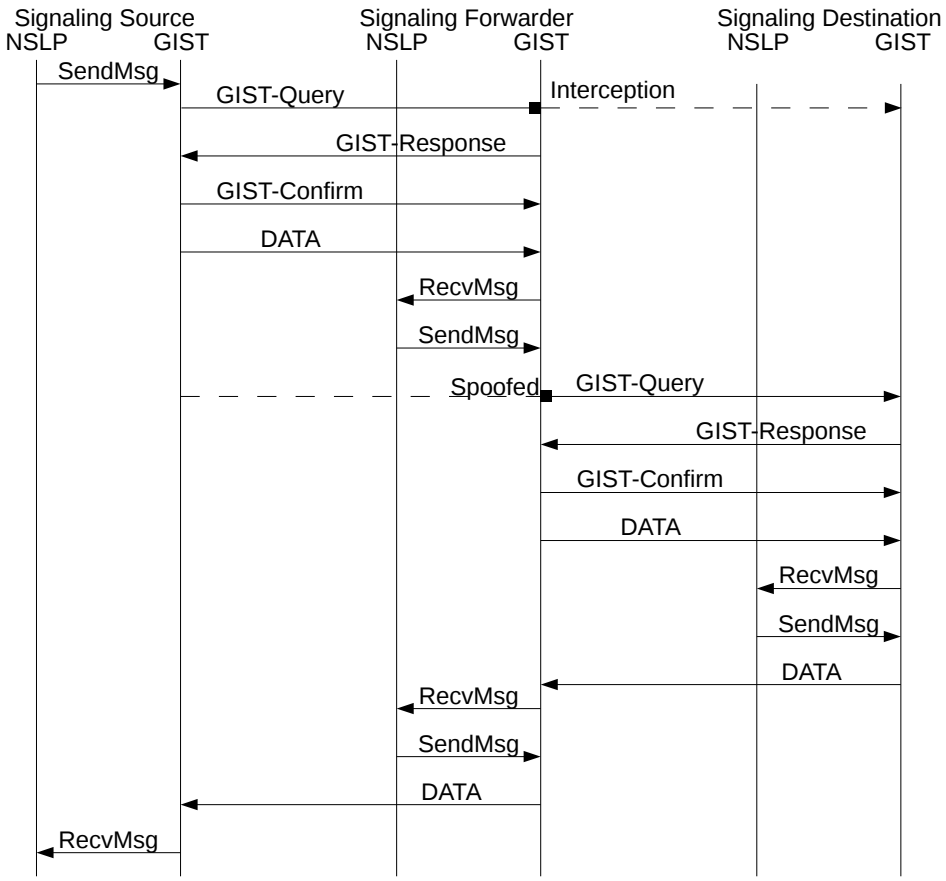


Figure 2.5: Basic signaling process

with the ReceiveMessage API call for the routing state over which the message was received. This handle can be used in the SendMessage API call to send a message over an existing routing state to a known peer.

GIST regularly retransmits Queries for existing routing states in order to detect topology changes. This process is called *GIST probing*. If a new peer is detected, the NSLP is informed of the new peer and receives a new handle to use the resulting routing state to (re-)send messages in order to update the end-to-end state as a result of the network topology change.

In addition to simple routing state, GIST can also establish a *Message Association (MA)* with a neighbor if reliable or confidential transport is requested. The QoS-NSLP works over unreliable transport by default as means of confirmation are built into the signaling protocol itself and MA-setup introduces additional, undesired delay. We do not work with Message Associations in our implementation due to implementation obstacles in the way MAs are currently set-up.

### 2.2.1 Quality-of-Service Signaling in NSIS

Quality-of-Service signaling is required whenever a node wants to reserve resources for traffic originated from or destined to itself. The QoS NSLP provides means to establish, manage and destroy such resource reservations. It is designed to allow for both sender- as well as receiver-initiated reservations—in contrast to the currently deployed *Resource ReSerVation Protocol (RSVP)* [3] that only supports the latter.

The design allows for using a plethora of different QoS-mechanisms including IntServ as well as DiffServ. It includes support for reservation sessions, allows to aggregate sessions as well as to bind related sessions together. This will be required to combine the reservations for a tunneled flow with the reservation for the tunnel flow itself (see Figure 2.1).

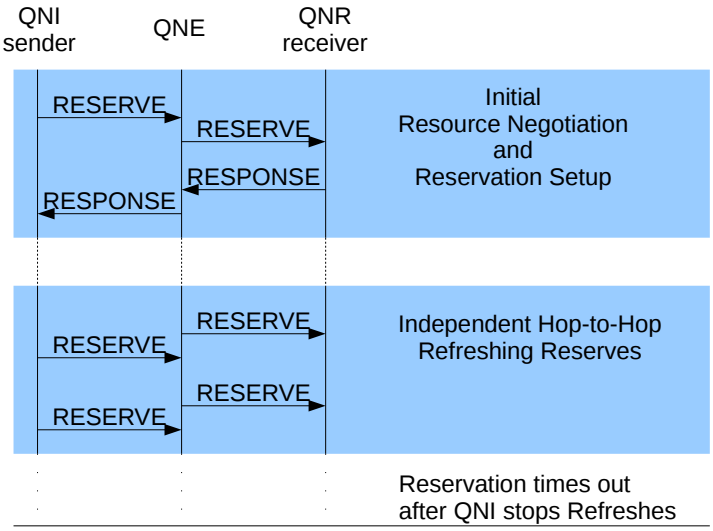


Figure 2.6: Basic QoS signaling messages – sender-initiated

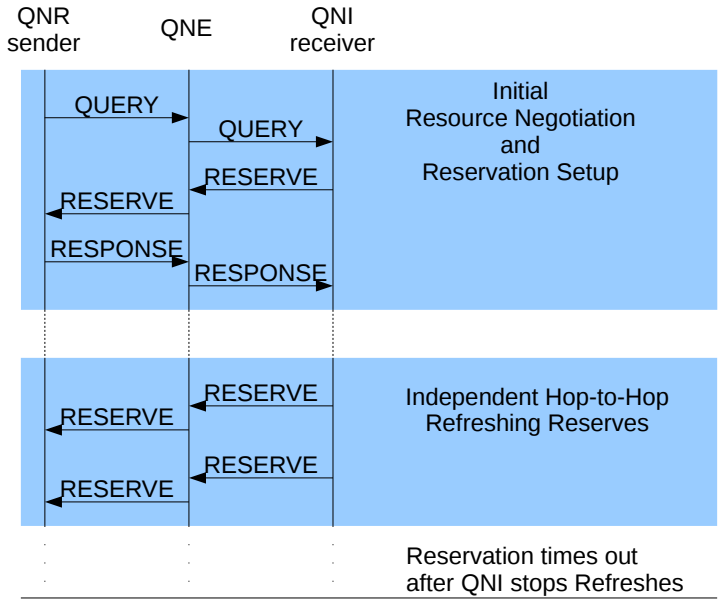


Figure 2.7: Basic QoS signaling messages – receiver-initiated

The basic message exchange performed by the QoS-NSLP is shown in Figure 2.6. In order to establish a reservation the *QoS-NSLP initiator* (QNI) sends a RESERVE message along the path of the flow the reservation is for. The QoS-nodes along the path (QNEs) forward the message until it hits the destination—the *QoS-NSLP responder (QNR)*—or one node detects that it can not satisfy the requested reservation. Once the destination receives a RESERVE it replies with a RESPONSE message that again travels back, hop-by-hop, until it reaches the initiator again. It should be mentioned that the RESPONSE is optional and has to be specifically

requested in the RESERVE message by attaching a RII-object. Setting up of the QoS reservation requires a RESPONSE most of the time, however. On the way the QoS-nodes set-up the necessary resources to the negotiated amount as they forward the RESERVE and later commit the resources as the RESPONSE is processed. Once the RESPONSE is received at the QNI the reservation is completely set up and the resources can be used. In order to preform receiver-initiated reserves (cf. Figure 2.7), the flow sender starts by sending a QUERY message to the receiver that then—in reply—sends a RESERVE and the mechanism continues as above. The QoS-NSLP uses a soft-state mechanism, so periodical refreshes are necessary to keep the reservation active.

## 2.3 Related Work

This section reviews documents and drafts from the NSIS working group that build the foundation for our work. We also review other approaches to the problem of QoS in mobile environments and highlight differences to our work. Finally, we give a brief overview of the software that is the foundation for our extensions.

### 2.3.1 From the NSIS Working Group

The mobility-draft [13] serves as the theoretical background for this work. It discusses the main problems with mobility and signaling and introduces some new concepts to deal with these problems. The main findings are:

1. Signaling must be preformed along the actual flow rather than the logical flow.
2. Movements are different from normal re-routing events and special care is required to avoid problems such as double reservations or accidental reservation tear down. In order to tackle this problem, the draft introduces the concept of a *Crossover node (CRN)* (in Figure 2.1, the CRN is marked with an "X"). The CRN is the first node that is on both the old path before a movement as well as on the new path after a movement. This node is responsible for tearing down the obsolete part of the old path. It also must filter accidental tear down messages coming from the old path after the last node has detected that the MN is no longer present at its end of the path.
3. Tunnel mode must be considered and needs special handling. For more detailed discussion of the operations required for tunnel mode, the mobility-draft falls back on the dedicated tunnel-draft [15].

Most of these points will be examined in more detail during our analysis. The draft [13] also provides some good examples which we will revisit in later chapters.

An early draft document [16], also produced in relation to the NSIS working group, provides an excellent discussion of the issues that arise from the different flows discussed in Section 2.1.1 and the consequences for the NSIS architecture. Most noteably, it includes an extensive discussion whether signaling should be done for the logical or actual flow, and at which layer to use which flow representation. We will reiterate some of the points in this draft in the next chapter as we discuss the different flows.

In addition to the mobility-draft, which focuses on the QoS-NSLP mostly, there is another article [18] that shows how the NAT-Firewall-NSLP can be used to allow MobileIP signaling, route-optimization and tunnel establishment in environments with restrictive firewall settings. Similar to the mobility-draft, this article is focused on the signaling process itself and does not provide details on the actual interaction between the signaling application and the mobility management.

### 2.3.2 Other approaches to QoS-signaling for mobility

New network developments these days, at least since 3G, always have mobility and some kind of Quality-of-Service support on the checklist of desired features. Many works were produced in this context ([9]). However, most of the approaches—originating from the context of classical, centrally managed telephone networks—rely on central management infrastructure and highly integrated solutions. With the move towards all-IP based solutions—in 4G networks and beyond—a slight shift of focus has happend and newer approaches move away from the centralization ([4]). Still, most currently proposed or deployed solutions tightly integrate mobility- and QoS-Management. This limits the implementation freedom serverly and often precludes future extension and adaptation.

On the other hand there are some proposals to add QoS-functionality to specific IP-mobility management systems. For instance [6] that suggests to piggyback QoS-information on the MobileIPv6 signaling messages. Again, these approaches lack in flexibility and extensibility as they focus on a specific environment.

The approach presented herein is different from that as it depends on a generic signaling framework that is only loosely coupled with mobility management. This allows changes in the mobility management as well as the QoS-signaling independent of each other, facilitating future improvements and adaption to new environments. At the same time it does not preclude a more tightly coupled implementation in order to facilitate environment specific optimizations. The signaling framework provides consistent semantics for the requested QoS-parameters and it is up to the local implementation how to best provide them.

Several approaches ([4]) utilize RSVP for QoS-signaling in the same way we use the NSIS framework. While this has similar benefits as our approach, RSVP has not been designed with mobility in mind and depends—for example—on the address information to correlate messages from the same session. As we mentioned above and will discuss in more detail later, this is a serious problem in mobile environments. The works in this category usually propose extensions to RSVP to include the HomeAddress as a session identifier ([17]) to work around this limitation.

### 2.3.3 Implementation Environment

The implementation of this work is based on the NSIS protocol implementation [2] provided by the *Universität Karlsruhe (TH)*. This provides a complete implementation of GIST as well as the QoS-NSLP. An overview of this implementation environment—including the MobileIPv6 daemon—is provided as Figure 2.8. In contrast to the draft it uses a three layer approach. A library provides basic data types and a high-level wrapper to transport protocols. GIST is implemented as either a standalone daemon that provides the NSLP API calls over a unix domain socket, or

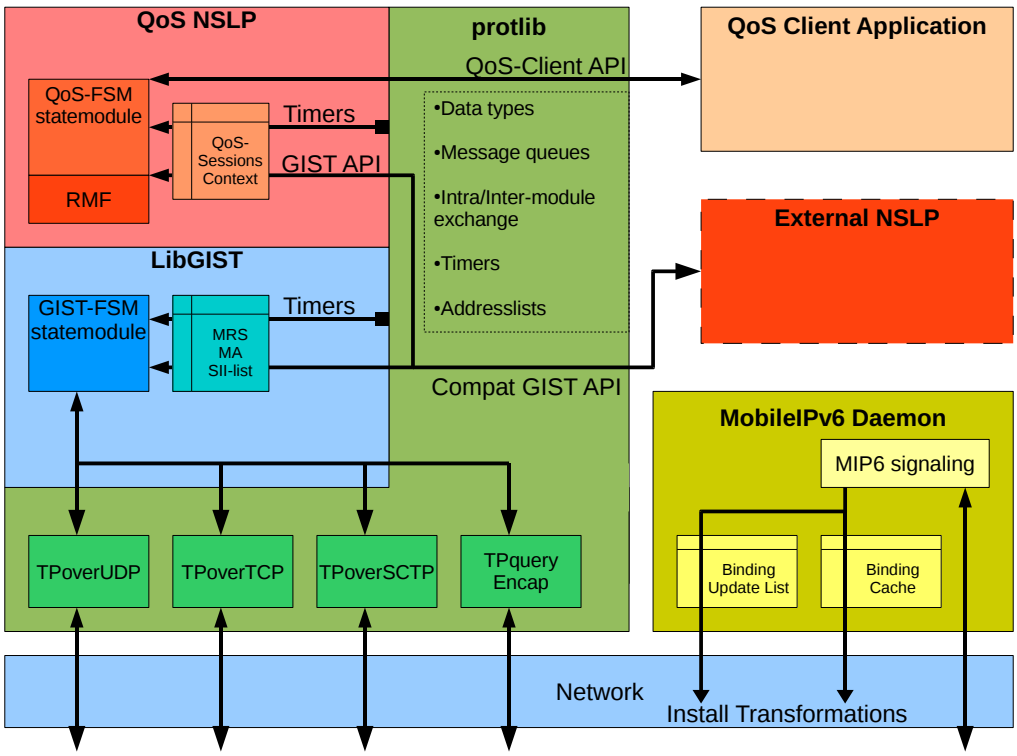


Figure 2.8: Architectural Overview of the Implementation Environment

it can also be used as a library linked to the NSLP application directly. The whole implementation is written in C++ and multi-threaded. All communication is done over internal message exchange between the threads making it easy to compartmentalize NSLPs into different address spaces. The QoS-NSLP application we use for our experiments is statically linked to the GIST library. It also links to a separate library that provides the QSPEC handling used to describe actual QoS reservations. This is not show in the figure.

Mobility support is provided by the Linux kernel with the *MobileIPv6 daemon* from the USAGI project[1]. The kernel provides a unified interface to apply transformations to the IPv6 header of incoming and outgoing datagrams. The MIP6-daemon uses this interface and realizes the MIP6 signaling described in some detail above. It also watches the interfaces for new Care-of-Addresses and other significant events.

# 3. Analysis

This chapter will identify the main, structural problems and challenges for Quality-of-Service signaling in a mobile environment. We do so by walking through several examples identifying the problems in each of them until we summarize the findings and come up with what needs to be implemented to resolve these issues.

## 3.1 QoS-signaling in Different Mobility States

In order to analyze what kind of special handling is required in the face of mobility we look at the MobileIP6 architecture step-by-step and try to identify what kind of signaling is required in the different scenarios.

### 3.1.1 At Home

While the MN is at the home network and communicating from the HoA directly, there is no special handling required as the actual and logical flows are identical.

### 3.1.2 Tunnel Mode

In tunnel mode (see Figure 3.1) there are two flows: the tunneled flow—an extension of the logical flow that is transparent to the CN—and the tunnel flow that carries the encapsulated packets between the HA and the MN. In order to establish Quality-of-Service for traffic passed between the MN and CN we must establish QoS reservations for both these flows. As described in the tunnel-draft [15] special handling is required at the endpoints of the tunnel. Initially either the CN or the MN—depending on which side the flow sender is—tries to establish a reservation for the logical flow and sends a RESERVE message along that path. If the MN is the initiator it must immediately also establish a reservation along the tunnel flow (or reuse/update an existing reservation). If the CN is the flow sender the RESERVE will travel towards the home network as normal, once the HA intercepts the packet it has to take care of a respective reservation for the tunnel flow. Once both reservations are established successfully the HA and MN should *bind* the two sessions together in order to notify the other if one changes or goes down. The QoS-NSLP has support for all the

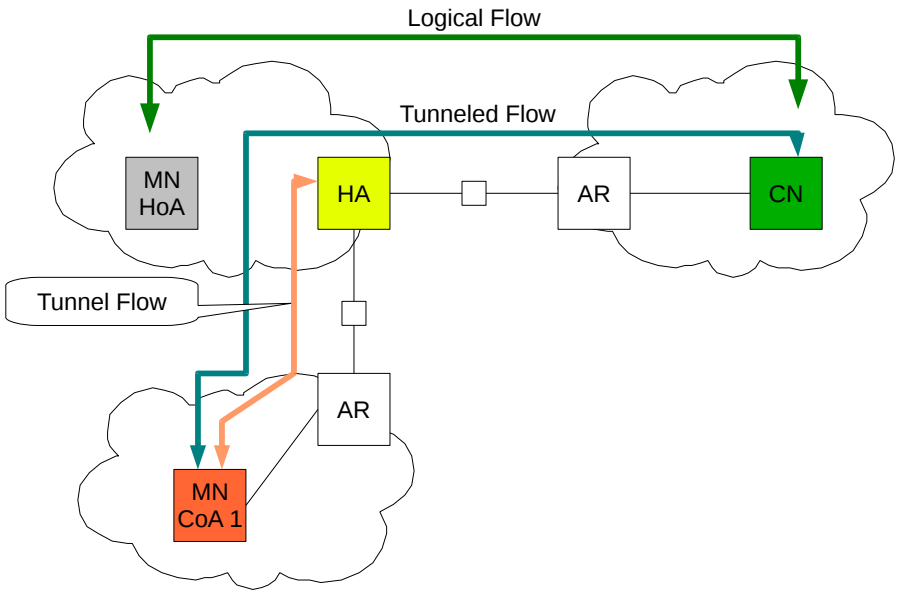


Figure 3.1: MobileIPv6 in tunnel mode

required mechanisms to cater for these operations and the only challenge in this scenario is the interaction with the actual tunneling mechanism. Figure 3.2 shows the signaling messages to establish a reservation initiated at the CN. Note that the reservations for the inner and outer tunnel flow could happen in parallel.

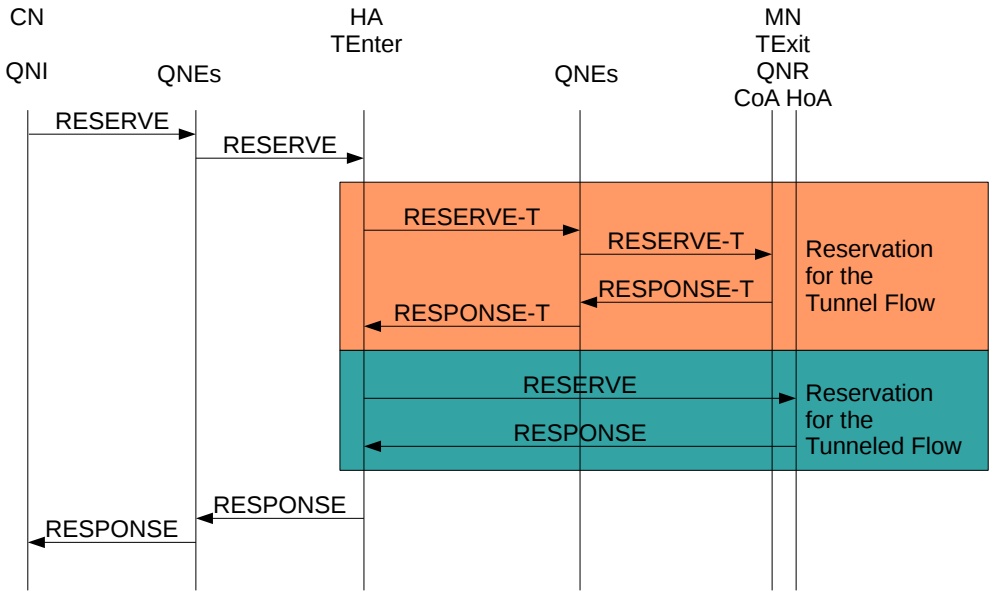


Figure 3.2: QoS signaling in tunnel mode with CN as QNI

This figure also shows that the inner and outer flow terminate at different addresses at the MN. The tunnel flow is between the HA and the MN's CoA, whereas the tunneled flow terminates at the MN's HoA. This has consequences for any Routing State that is used to carry the corresponding signaling messages.

Figure 3.3 shows a serious problem with tunnel mode when the HA uses signaling source mode for the outgoing GIST-Query. As the HA always has an active Binding with the MN, IP-datagrams from the HA-Address to the MN's HoA will always be

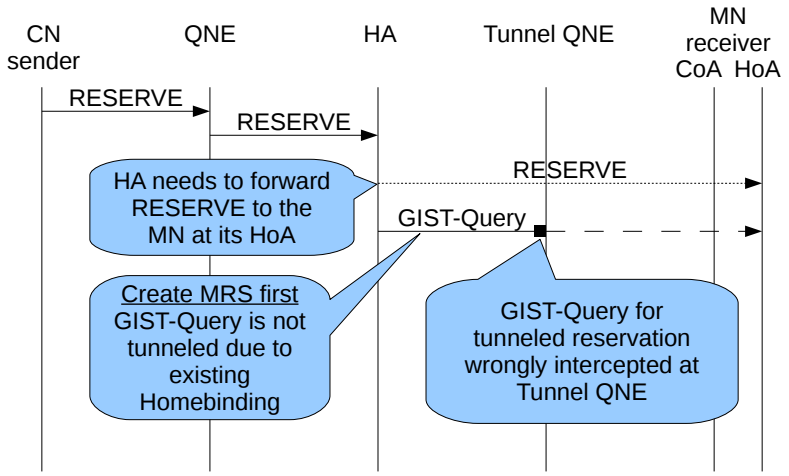


Figure 3.3: Problems with GIST-Queries in tunnel mode

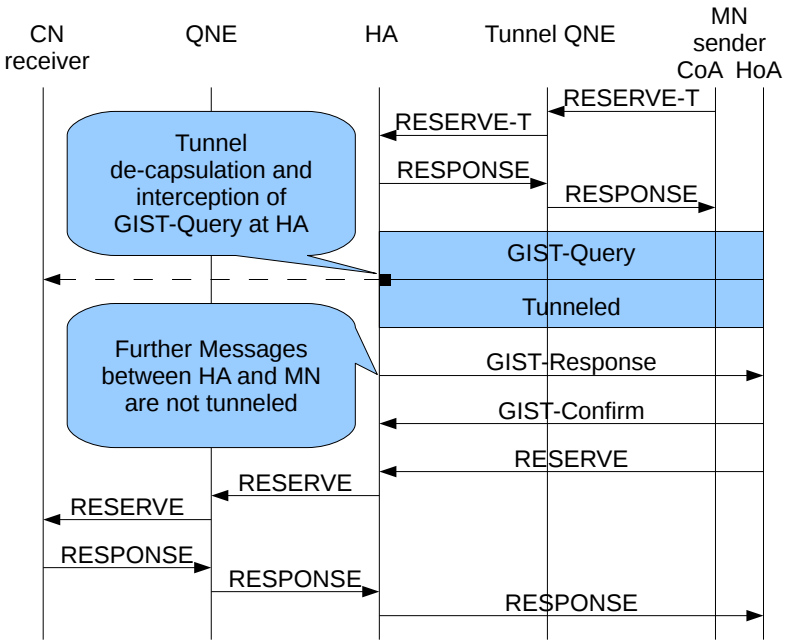


Figure 3.4: Same situation as in Figure 3.3 with MN as sender

route-optimized. This also applies to the GIST-Query that is supposed to establish routing state between the HA and the MN at its HoA. As a result the Query is not encapsulated in the tunnel and will be intercepted by any intermediate NSLP nodes. The intercepting node will then process the RESERVE message from the HA and try to forward it to the MN. In order to do so it sends a GIST-Query towards the MN's HoA that ultimately arrives at the HA again creating a loop (unless every intermediate hop also has an active Binding with the MN).

In order to avoid such problems there are three possible solutions:

1. Avoid GIST-Query interception for route-optimized packets.

2. Avoid signaling source mode for GIST-Queries from the HA to the MN.

3. Force GIST-Queries in signaling source mode from the HA destined to a connected MN's HoA into the tunnel encapsulation, regardless of any active Bindings.

Fortunately, this is not a problem when the MN is the QNI as one can see from Figure 3.4. In this scenario, the GIST-Query datagram will always go into the tunnel as it is destined for the CN. If there exists an active Binding with the CN, the MN would not try to establish a tunnel mode reservation in the first place. Also note that further messages (after the GIST-Query) between HA and MN need not necessarily be tunneled. This is not a problem as these messages will not be intercepted at intermediate nodes but are treated as normal datagrams and forwarded to the destination as such.

### 3.1.3 Route-Optimization

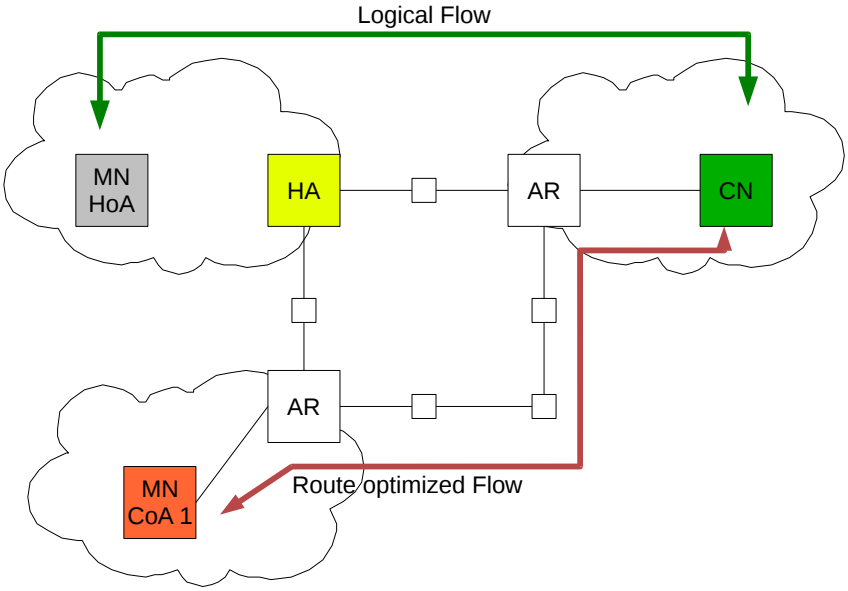


Figure 3.5: MobileIP6 in route-optimization mode

Once in route-optimization mode no traffic flows along the logical path anymore and consequently any QoS-reservation must be made along the actual path. The signaling operations required are not at all different from the normal operation, but care must be taken to the Message Routing State that is formed to and from the MN. All MRSs that carry route-optimized flows should be terminated at the MN's CoA, otherwise—if the HoA was used—the MRS would be tunneled over the HA or get route-optimized itself. This would impose serious latency to the signaling due to the additional HA detour. This is discussed in more detail in section 3.4.

Intermediate nodes (QNEs) on the (route-optimized) path between the CN and MN will not see any difference between a route-optimized flow and a normal flow. No special handling or additional features are required. Earlier draft documents [16] were considering to keep the logical flow source and destination in the MRI for route-optimized flows. This, however, would mean that all nodes on the path would have to parse the mobility options that are placed in the IP packets in order to

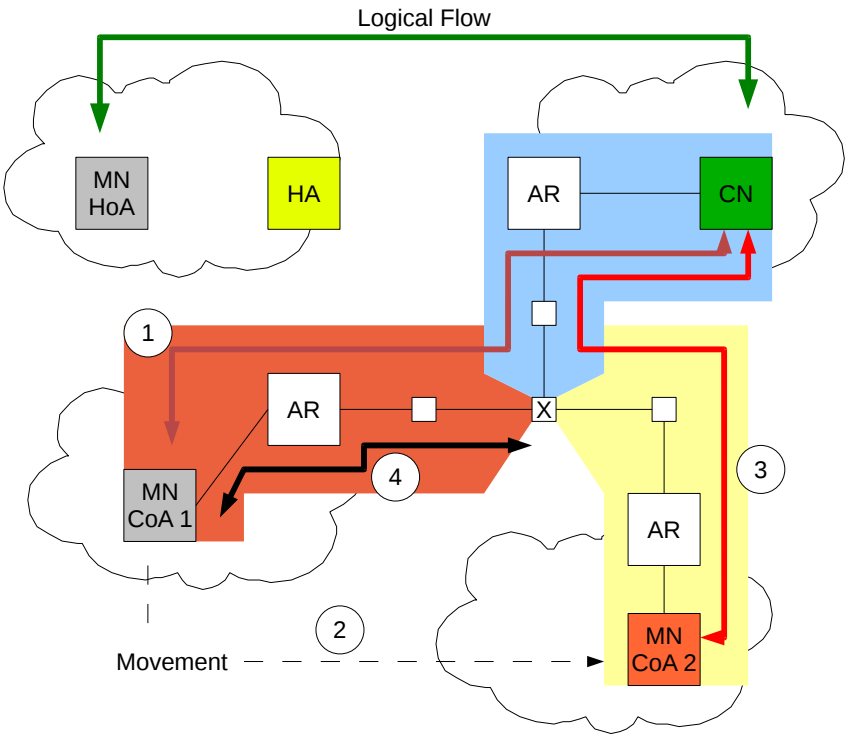


Figure 3.6: MobileIP6 in route-optimization mode after a movement

classify the packets that belong to the route-optimized flow. Using the actual flow in the MRI for the route-optimized reservation makes this much easier.

Using the route-optimized flow identifier, however, raises another problem illustrated in Figure 3.6. After a movement (cf. 2 in the figure) the MN acquires a new CoA and the flow changes (cf. 3 in the figure). This means that the flow identifier alone cannot be used at intermediate nodes to detect a mobility event. Fortunately, the QoS-NSLP uses another, flow identifier independent *SESSION_ID* to track reservations and end-to-end state. This allows intermediate nodes to correlate messages before and after a movement to the same session. The first node that already has an active reservation with the same SESSION_ID but different flow identifier will note this fact and become the *Crossover Node (CRN)* as described in the mobility-draft [13]. The CRN is denoted by an "X" in the figure. It is the job of the CRN to initiate release of the—no longer required—resources on the now obsolete path (cf. 4 in the figure). In some situations—discussed in detail in the following section—the CRN can not directly issue a tear and can only inform the old path that it has detected a route change. It is then the job of the last node on the old path to detect that it is a dead-end and to send a tear towards the CRN. The CRN then has to block further propagation of this tear on the still valid part of the reservation.

## 3.2 A Detailed Look at Signaling Operations

Taking a closer look at the required signaling operations we identify four different cases in a mobile environment: Sender- and Receiver-initiated reservations with either the MN or the CN as flow sender. The flowing message flow figures show the process for route-optimization mode in detail.

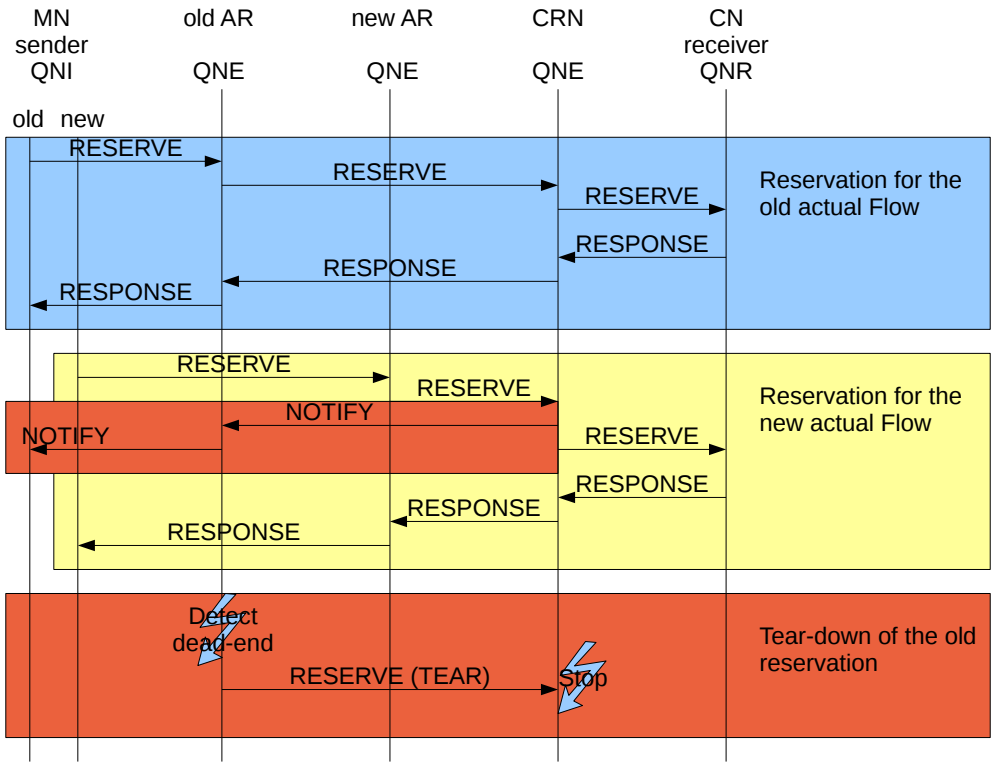


Figure 3.7: QoS-Signaling Messages: MN sender, sender-initiated

In the case where the MN is sender and QNI (cf. Figure 3.7) the Crossover Node detects a change of its upstream peer when receiving the new RESERVE message. Thus, it can not tear down the old path on its own, but can only inform the old path that it has detected a routing change by sending a NOTIFY message. From that point on it will silently drop refreshing RESERVEs from the old path. At some point the last QNE on the old path (usually the old Access Router) will detect that its upstream peer is no longer available, together with the earlier NOTIFY it now can assume that it is the dead-end of the old path and sends a RESERVE with the TEAR-flag set in order to free the resources. The CRN stops further propagation of the TEAR. Note that there can be other QNEs between the CRN and the dead-end on the old path. The resources on these nodes will also be freed as the TEAR from the dead end is processed.

When the CN is sender and QNI at the same time (cf. Figure 3.8) the situation is different. The CRN detects a change of its downstream peer when forwarding the new RESERVE message. At this point it has to store the old MRI in order to later send a RESERVE with TEAR-flag. It does so as soon as it receives the RESPONSE for the new reservation and is thus sure that the old reservation is no longer required. The TEAR is forwarded on the old path—again along as many QNEs as there are between the CRN and the dead-end. The dead-end QNE (again, usually the old AR) tries to forward the message to the MN at the old CoA that is no longer available and the process stops.

The receiver-initiated cases (see Figures 3.9 and 3.10) look very similar to the respective receiver-initiated case. The only difference is the initial QUERY message to start the process.

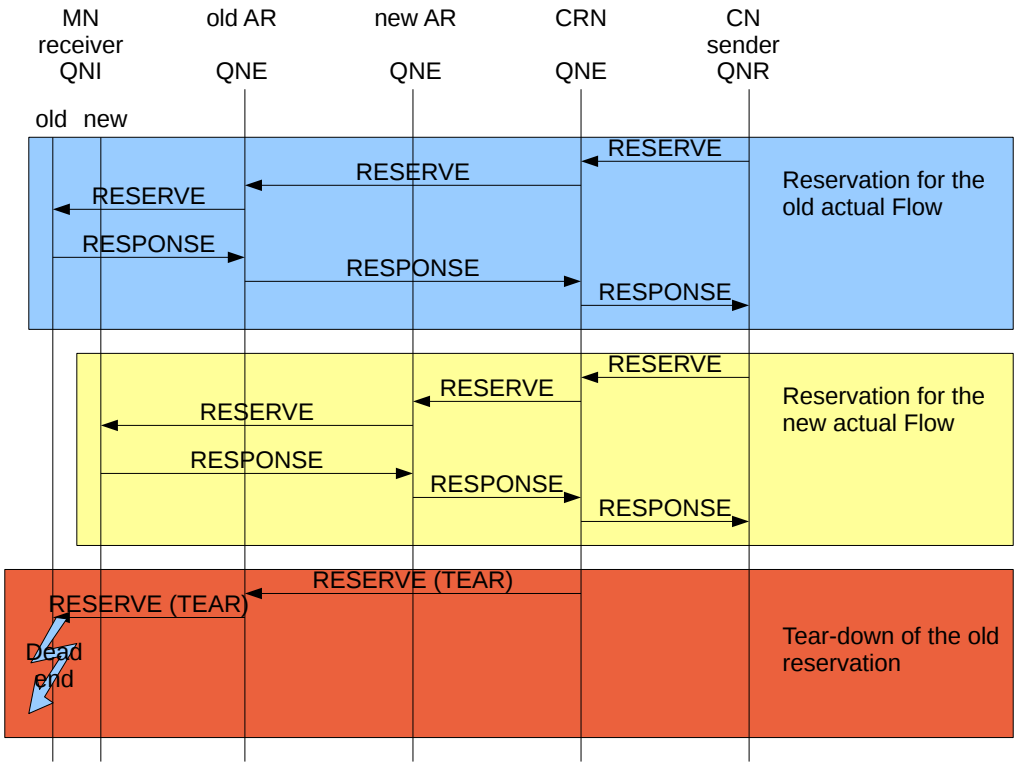


Figure 3.8: QoS-Signaling Messages: CN sender, sender-initiated

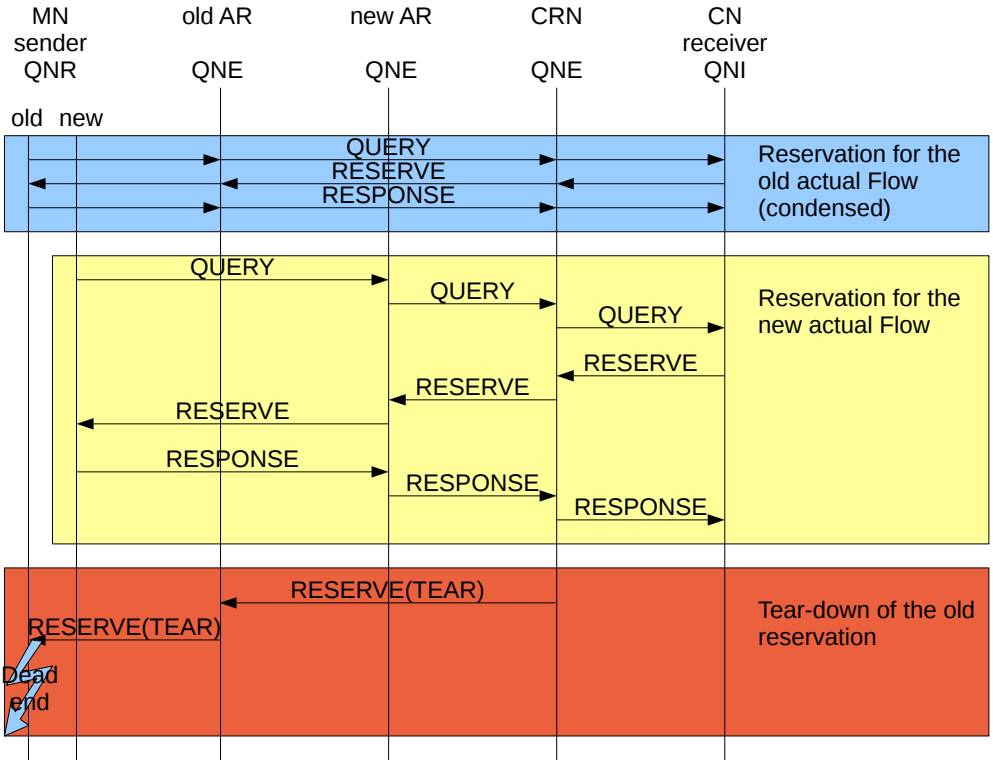


Figure 3.9: QoS-Signaling Messages: MN sender, receiver-initiated

In addition, it should be noted that the QNR can also be the Crossover Node if there are no other common QNEs between the old and new path. In this case the QNR will perform the same tasks as the CRN as it has the same information available.

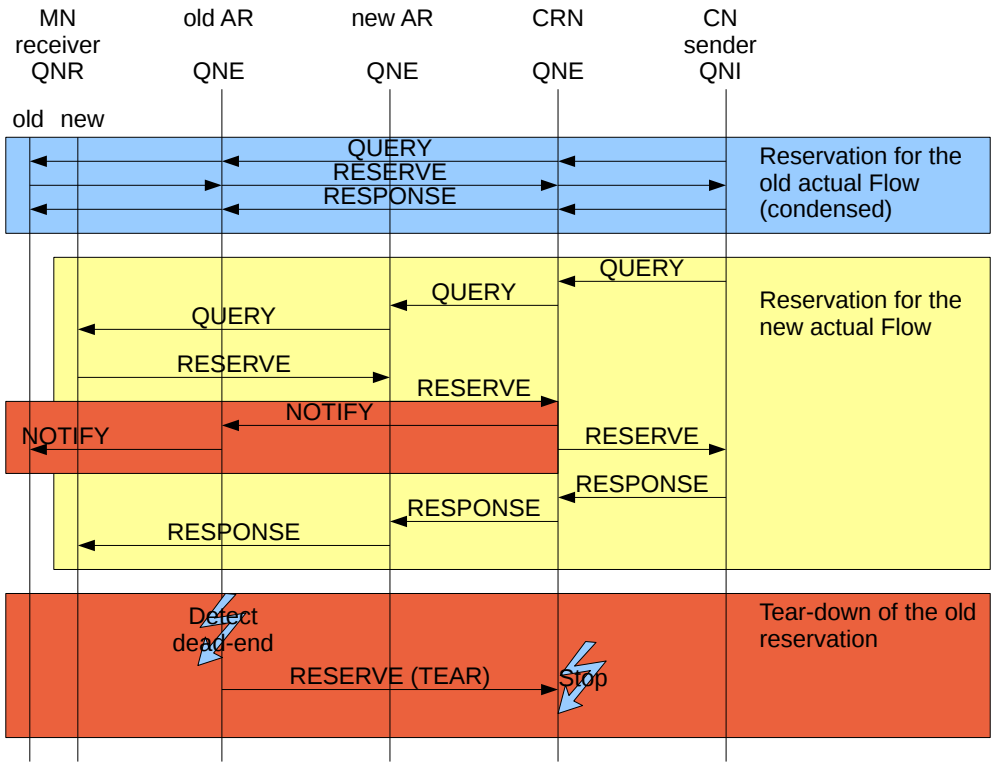


Figure 3.10: QoS-Signaling Messages: CN sender, receiver-initiated

Also of note is that in cases where the QoS-NSLP cannot send a TEAR from the CRN directly, because the direction of the reservation does not allow so and we fall back on sending a NOTIFY, the QNEs on the old path will be issuing refreshing RESERVEs for some time before the dead-end is detected and the path is properly torn down. It might even happen that the reservation on the old path times out before the dead-end is detected at all. Neither of this is a problem. We only have to ensure that the CRN properly filters the refreshing RESERVEs.

Finally, it is of note that the Mobile Node is almost always a Crossover Node as well, because the up- or downstream peer of the MN changes after the handover unless the movement is localized enough that the next GIST hop is the same as before. Most of the time this doesn't matter as the MN can not communicate from the old CoA and thusly can not issue any messages on the old path. The only exception from this is after the MN leaves its home network. In this scenario the routing state between the MN and HA has been established from/to the HoA and the MN can still use it to send a message to the HA, and the old path, over it. After this message has been sent the MN must make sure that the routing state is torn down, otherwise subsequent GIST probing attempts will be route optimized and confuse the next hop on the new path, as it will interpret the newly established routing state as a routing change and start CRN processing.

### 3.2.1 Tunnel Mode Operation

The basic signaling operations for tunnel mode are already described in Section 3.1.2. In essence the problem can be reduced to a route-optimized reservation between the MN and the HA. As we will show in the implementation there is no other special handling for tunnel reservations required once it has been created. The

only issues that need to be addressed is the creation and—after switching to route-optimization—the destruction of the tunnel reservation.

Another issue to consider is the fact that after a handover there is a short time where the MN and CN may fall back to tunnel mode before a new binding has been established. This tunnel is usually very short-lived and does not justify an additional tunnel reservation. In order to avoid creating a tunnel reservation just to tear it down again a few seconds later, we will introduce a hold-off timer that is started as soon as we detect a tunnel and establishes the required tunnel reservation after a short wait period unless the flow has switched to route-optimization in the meantime.

## 3.3 Mobility unaware NSLP Implementations

From the analysis so far it is obvious that all nodes that are active components in the MobileIP setup (the Mobile Node, the Home Agent and MobileIP enabled Correspondent Nodes while not in tunnel mode) need also mobility aware signaling applications. The impact on operational reliability differs. While in tunnel mode the impact is limited to the tunneled section. If the signaling application on either MN or HA are not mobility aware, this might result in a missing reservation for the tunnel section—depending on the direction of the reservation. Once route-optimization is enabled it is mandatory that the signaling implementation on the flow sender is mobility aware, otherwise it will—as discussed above—try to establish a reservation for the logical flow. If this attempt reaches the receiver and the implementation on that node is mobility aware it is possible to detect the error and ask the mobility management to fall back to tunnel mode. Otherwise the signaling will fail.

Table 3.3 summarizes possible scenarios and confirms the assumption that the Mobile Node is key in the process. As long as the signaling application on the MN is aware of its own mobility everything fails gracefully. The missing tunnel reservation can not be worked around. Eventhough the MN is able to detect the case where the signaling application on the HA forgets to create a tunnel reservation it can not issue a reservation itself if it is not the flow sender. A possible sollution would be to send a special NOTIFY or QUERY message to the HA via the EST-MRM that would then make the HA issue a reservation. Currently the QoS-NSLP specification does not offer such a message, however. It should be noted that a receiver-initiated reservation does not offer what is needed. Because the signaling path is always built from sender to receiver, it is most likely different from the path in the opposite direction due to asymmetric routing.

## 3.4 Message Routing State from/to the MN

In order to transmit actual traffic, GIST establishes so-called Message Routing States (MRSs) along the discovered path, basically the state of a signaling connection between neighboring GIST nodes. On the MN much care has to be taken concerning the decision whether to use the HoA or the CoA as source/destination for an MRS. In order to establish the MAs as closely as possible along the path of the actual flow, the CoA is the right choice in most cases. Figure 3.11 illustrates the additional problem already mentioned in section 3.1.3. If the message routing state is established from

| Mobility awareness | Operation mode | Impact |
|---|---|---|
| MN ✓, HA ✓, CN † | Tunnel mode | none |
| | Route optimization, MN is sender | none |
| | Route optimization, CN is sender | fall back |
| MN ✓, HA †, CN ✓ | Tunnel mode, MN is sender | none |
| | Tunnel mode, CN is sender | no tunnel reservation |
| | route optimization | none |
| MN ✓, HA †, CN † | Tunnel mode, MN is sender | none |
| | Tunnel mode, CN is sender | no tunnel reservation |
| | route optimization, MN is sender | none |
| | route optimization, CN is sender | fall back |
| MN †, HA ✓, CN ✓ | Tunnel mode, MN is sender | no tunnel reservation |
| | Tunnel mode, CN is sender | none |
| | route optimization, MN is sender | fall back |
| | route optimization, CN is sender | none |
| MN †, HA ✓, CN † | Tunnel mode, MN is sender | no tunnel reservation |
| | Tunnel mode, CN is sender | none |
| | route optimization | fail |
| MN †, HA †, CN ✓ | Tunnel mode | no tunnel reservation |
| | route optimization, MN is sender | fall back |
| | route optimization, CN is sender | none |
| MN †, HA †, CN † | Tunnel mode | no tunnel reservation |
| | route optimization | fail |

Table 3.1: Different cases of mobility-aware QoS NSLP nodes

or to the HoA the resulting messages are tunneled to the HA and back incurring a serious delay on the way. Only in tunnel mode and for the inner tunnel reservation should the HoA be used as the source of the MRS with the HA. Depending on the mobility service implementation, special operations may be required to force communication from the CoA directly. This is discussed in more detail in Chapter 4.3.

## 3.5 Overhead due to MobileIPv6

We mentioned that MIP6 uses special IP6 extension headers to carry information about the HoA in route-optimized flows. These options impose a serious overhead on packets that are route-optimized. The overhead can amount to as much as 48 byte if both sides of the flow are mobile nodes. The same is true for tunnel-mode where an entire new IP-header is appended for the tunnel encapsulation. If the tunnel is protected by IPSEC there is even more overhead. For typical applications that need QoS—such as real-time video and audio exchange—a per-packet overhead of 40+ bytes can mean a bandwidth increase of 50% (40 byte overhead / (40 byte IPv6 header + 20 byte UDP header + 20 byte codec payload[1])). These applications tend

[1] e.g. G.729 (8 Kbps)

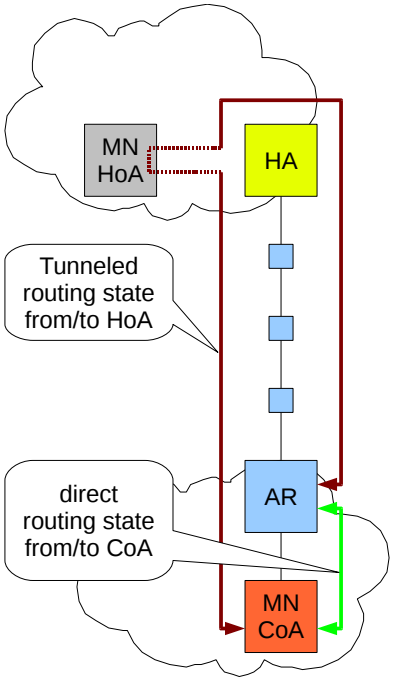


Figure 3.11: Tunneled and Direct Routing State

to use a steady stream of relatively small packets in order to minimize the delay. As the client application is oblivious of this overhead it will ask for a reservation based on the unmodified packet size. It is then the job of the signaling application to adjust the bandwidth according to the overhead. This is not trivial as the client application usually does not specify a packet size and rate for the data it intends to send, but instead provides other, more abstract configuration parameters—such as bandwidth and bucket size. In order to calculate modified values to account for the overhead, the signaling application has to deduct the mean packet size and rate from these abstract parameters.

## 3.6 Requirements

In this chapter we have identified a number of problems that need solving in order to provide functional QoS signaling in a mobile environment. The following requirements for our implementation must be resolved:

1. The signaling application needs to be aware of active MobileIP bindings and the resulting flow transformations.

2. The signaling stack (NTLP and NSLPs) must be aware of and react to mobility events.

3. The QoS-application must be aware of additional overhead induced by MobileIP specific extension headers or the tunnel encapsulation.

4. The signaling application must signal for the actual flow instead of for the logical flow as provided by the user applications requesting signaling from the signaling application.

5. The signaling stack must be able to communicate from the CoA directly.

Most of these depend on information available to the MIP6-daemon and thus we need an efficient way to obtain this information from it. In addition we need to decide where and in what form this information is required and design the interface accordingly. The next chapter will discuss this in detail.

## 3.7 Summary

The analysis so far shows that the QoS-NSLP is already well equipped to deal with mobility. The existence of a static SESSION-ID that is independent of the flow address information allows for easy correlation of messages for the same session before and after a movement. The existing re-routing handling provides a good abstraction and can be reused without much modification to handle mobility events as well. In route-optimization the required modifications and interactions with the mobility management are limited to the flow sender. For tunnel-mode additional modifications and interaction is required at the HomeAgent. In addition, the modifications are limited to the initial setup of a reservation and handling of mobility events. No modifications of the normal processing are otherwise required.

# 4. Design and Implementation

As mentioned before, most of the theoretical details about what to signal are already described in the mobility draft [13] and presented in Chapter 3.2. In our design we can focus on the problem of how to realize and implement the outlined signaling processes. The analysis in Chapter 3.6 summarizes the problems that need to be solved. This chapter provides a more detailed look at each problem in the specific environment and a high level solution. Further down herein we also discuss the actual implementation of the required services and changes.

## 4.1 Flow Info Service

One of the most fundamental problems we have identified during our analysis is the fact that we need to break the transparency of MobileIP and to provide means for the signaling application to query state from the MobileIP management service. In addition we require the MobileIP management service to send notifications to the signaling application for significant events, such as handovers, new CoAs and a change in the binding cache and binding update list. While there is a specification for a MobileIP6 management information base in [7] that would allow implementation independent access to the required information, the MIB is not implemented in any of the MIP6 services available to us at the time. Instead it was decided to roll an implementation specific service that provides only the required functionality. We call this *Flow Info Service* for its primary functionality.

The basic mode of operation is a request-response mechanism, as show in Figure 4.1. The signaling application sends a request indicating that it wishes to retrieve information about a certain flow. The Flow Info Service replies with the current state of that flow. In addition the Flow Info Service sends notifications whenever the state of an active flow changes. This provides mobility event triggers required by QoS NSLP to issue updating RESERVE or QUERY messages. This way the consumer (the signaling application) is able to cache the results provided by the Flow Info Service, but does not need to bootstrap and mirror the complete state in the MobileIP management. A lazy implementation can fall back to querying the state over and over, i.e. a polling mode.

The request needs to describe a flow, the information required for that are two IP-addresses (source and destination). No other information is required. The reply needs to describe three possible flow states:

1. No MobileIP flow – because the source is not a HoA of the node or there is no active binding for the destination. It might turn out later on that the peer is a mobile node, but for the moment the flow is sent unmodified.

2. Tunnel mode – the flow will enter or exit a tunnel at the current node, on the MN or the HA respectively. This happens when the flow source is a HoA and the peer is either MobileIP unaware or there is no binding established for it, yet. In addition to the state itself that response has to include the tunnel source and destination in order to enable the signaling application to establish a bound session for the tunnel section or update an existing session accordingly. Future implementations might want to provide a tunnel id that would make it possible to demultiplex the flow inside the tunnel on intermediate tunnel nodes, e.g by copying the IPv6 flow label or DSCP from the inner packet header. Currently there is no possibility to distinguish flows inside the tunnel. The tunnel-draft [15] has more details about tunnel-ids.

3. Route optimization – there is an active binding for this flow and the flow source and/or flow destination will be rewritten. The information required in the reply consists of the new flow addresses.

Figure 4.1 gives a summary of the contents of the messages exchanged in the Flow Info Service.

Every reply should additionally indicate the per-packet overhead that is incurred by the translation. Even though this information can be deducted from the transformation itself, parts of it might be implementation specific. Finally, the reply should carry enough of the original request to enable parallel operation. The simplest solution is to quote the complete flow that is concerned. Requests are handled in the same order in which they arrive, consecutive requests for the same flow result in several replies with the requested information. This allows the client to issue parallel requests for different flows and from more than one consumer (in our case NSLP and NTLP) over a single Flow Info Service connection.

The notifications also use the reply message format and simply inform the client (the signaling application) about the new state of a flow. If the client cares about the state that was in effect before the event it will have to deduct that from its own state.

On the signaling application side we have to consider where and how to make the Flow Info Service available to the different application parts. The general spirit of separation of a network aware transport layer and a somewhat network unaware signaling layer would suggest that this service should only be required in the transport layer, which would then notify the signaling layer with information that is of interest to it. With the current interface design between transport and signaling layer, where the signaling layer has to provide the flow information (the MRI) for any message it wants delivered, this is not easily possible. It was considered to let the transport layer translate the MRI on the fly with the information available from the Flow Info

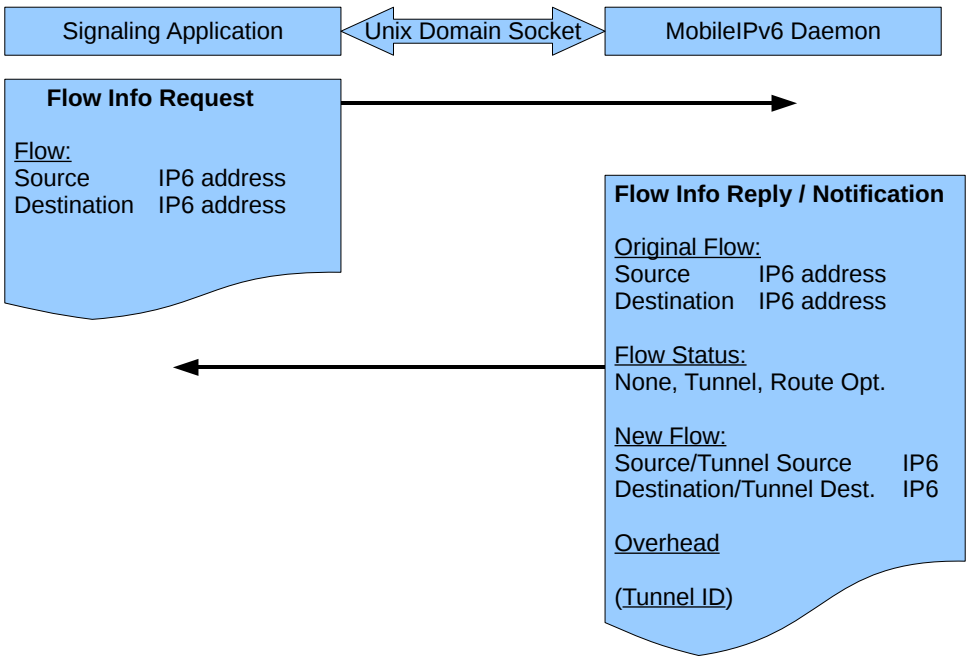


Figure 4.1: Messages for the Flow Info Service

Service. This would keep the separation. A similar mechanism was proposed to deal with network address translation (which is a similar problem) in the NAT-traversal draft [10, section 5.3]. It was decided, however, that this approach is not applicable when dealing with mobility as the signaling layer requires very specific knowledge of the flow status to function properly. While the MRI translation service could hide the addressing details, the transport layer would still have to provide a large amount of other information about the flows. It was decided that the interface extensions required to convey this information were too extensive. Instead the signaling layer is given direct access to the Flow Info Service and must provide correct flow information to the transport layer. This has the downside that every signaling application has to be made mobility aware, eventhough it might not require special handling. Considering the currently specified applications: Quality-of-Service and NAT-firewall—which both require rather extensive changes—it seems unlikely that future signaling applications could work without extensive knowledge of the mobility status. In addition, this design gives the most flexibility in implementing the mobility changes in the QoS-NSLP.

Figure 4.2 shows the placement and partitioning of the Flow Info Service and how the different components communicate with each other. The following sections describe the individual components in detail and specify the protocols for the various interfaces.

### 4.1.1 Flow Info Service – Provider

In order to export BindingCache and other mobility state information from the MIP6-daemon to the signaling application we added a Unix Domain Socket interface to the daemon code. The unix domain socket interface allows for inter-process communication. It uses a fixed packet size and allows for four types of packets that closely follow the previously described findings. The implementation allows for a

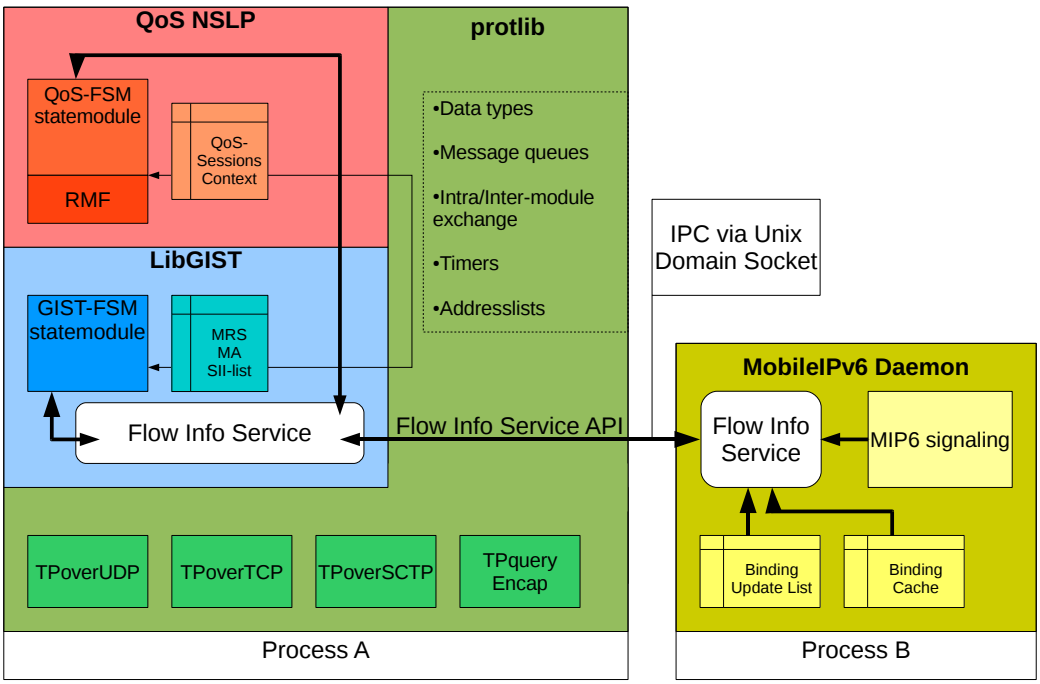


Figure 4.2: The Flow Info Service Placement

fixed number of concurrent clients that can be configured at compile time. For every client a new thread is spawned that accepts requests, looks up the information and sends a reply or error in return. If a mobility event occurs, the main thread iterates through all connected clients and sends an event datagram with the information relating to the mobility event.

### 4.1.2 Flow Info Service – Consumer

On the client side of the Flow Info Service (in the QoS-signaling application) both the NTLP and the QoS-NSLP layer need direct access to the service. In the current form, both layers reside in the same binary and address space so that they can share a common Unix Domain Socket connection with the provider. In the future it may be possible to have the NTLP and NSLP reside in different address spaces where each component would need to open a Flow Info Service connection with the MIP6-daemon on its own. The handling of the UDS messaging is hidden behind a class that provides a single, convenient interface to lookup information for a flow. As the NTLP already provides a data structure that describes a flow—the Message Routing Information—this data structure is reused for the Flow Info Service interface as well. This is particularly convenient as most of the operations required by the NSLP evolve around the translation from a logical MRI to the actual MRI.

The interface looks as follows:

```
Flowstatus *get_flowinfo(mri_pathcoupled &orig_mri);
```

with "Flowstatus" resembling the same information as available in a Flow Info Service reply or event in terms of the data types available to the NTLP:

```
struct Flowstatus {
    enum flowstatus_t {
        fs_nothing,      /* no active binding -> no transformation  */
        fs_normal,       /* active RO binding -> route-optimization */
        fs_tunnel,       /* active non-MIP binding -> tunnel mode   */
        fs_home          /* home binding -> route-optimization      */
    };

    flowstatus_t type;
    mri_pathcoupled orig_mri;
    mri_pathcoupled *new_mri;
    mri_pathcoupled *tunnel_id;
    uint32 pp_overhead;
};
```

The interface is part of a class object. At start-up the QoS-Application spawns a single thread of this class that will try to connect to the MIP6-daemon over the UDS and manage the socket. All interested parties are given access to the thread object and through that to the `get_flowinfo` interface. At the moment requests over the high-level interface are translated verbatim into a Flow Info Request and sent to the MIP6d and vice versa with the reply. Future improvements can easily implement caching at that place to avoid unnecessary UDS traffic, context switches, and other overhead. Casual measurement of the request delay suggests, however, that there is little reason for concern at the moment.

In addition to the `get_flowinfo` interface, the socket managing thread will also listen for event datagrams. On receipt of a notification, the thread sends an internal message to the NTLP layer. After NTLP has completed any actions that are required from its point of view (at the moment there is nothing in this category) the NTLP layer sends a special NetworkNotification to all locally connected NSLPs with the original MRI as reported in the event. The message must be delivered to all NSLPs as the NTLP does not have access to information regarding the logical flow and it can not deduct the old flow (prior to movement) from the information available at the time of the event.

## 4.2 Quality-of-Service NSLP changes

The changes to the operation of the QoS-NSLP required to work in a mobile environment have been outlined in Chapter 3.1. To recap, the client application asks the QoS-NSLP to establish a reservation for a flow. The flow provided is the logical flow as seen by the client application and it is the QoS-NSLP's job to translate this request to a reservation for the actual flow as seen by the MIP6 management. Once the reservation has been established, the QoS-NSLP must listen to MIP6 events that change the actual flow for this reservation and renew it accordingly.

In the NSIS implementation we use, this process works as follows: The client application sends a preliminary RESERVE or QUERY message to the QoS-NSLP

over a client API. The QoS-NSLP transforms this request into a real RESERVE or QUERY message by assigning a SESSION-ID and turns it into a valid NSIS PDU. The message is then handed off to the *QoS-statemodule* that implements the *Finite State Machine (FSM)* described in the QoS-NSLP draft. The statemodule creates a context for the reservation, installs required state at the local node and hands the message off to the NTLP layer for delivery. Processing of incoming messages is also implemented inside the statemodule, as well as processing of GIST NetworkNotification messages. The statemodule is thus the natural point to perform any mobility transformations to the messages as well as the hook point to process mobility events.

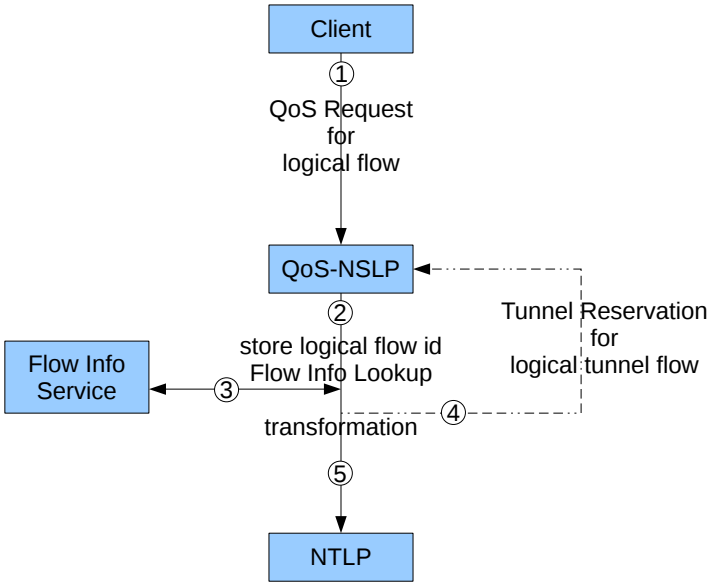


Figure 4.3: Mobility processing of initial QoS-client requests

Upon receiving the first RESERVE or QUERY message from the client API the statemodule queries the Flow Info Service to check if mobility processing is required. If this is the case, the logical flow information is stored in the created context and the message is modified for the actual flow. If tunnel mode is in effect a second RESERVE or QUERY message is generated for the tunnel flow and sent through the normal processing. The process is depicted in Figure 4.3. The message for the tunnel reservation is generated with the logical tunnel flow id—between the MN's HoA and the primary HA address—initially, and thus allows the same processing for the tunnel reservation as for a normal reservation. This approach simplifies the code a great deal as no special processing is required to react to mobility events for tunneled reservations. Instead the tunnel reservation is updated automatically by the mobility event for the logical tunnel flow.

The processing of mobility events is shown in Figure 4.4. Mobility events from the Flow Info Service are sent to the NTLP that then transforms the event into a NetworkNotification API call to the NSLPs. We introduce two new NetworkNotification types to convey mobility events: *Home Binding Update* and *Binding Update*. Both carry the MRI for logical flow. As with other NetworkNotification events, it is the job of the NSLP to figure out which, if any, of its active sessions are affected. The QoS-NSLP uses the stored logical flow id to lookup any such context. We had to add a new access structure to handle this lookup. Previously the QoS-NSLP was

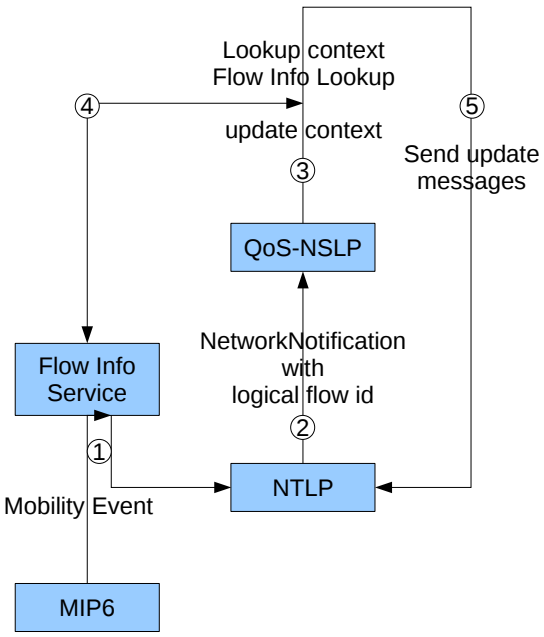


Figure 4.4: Processing of mobility events

only able to look up a context by its SESSION-ID as our NTLP always provided this information as well—in contrast to the definition in the GIST draft. This is no longer the case, as the NSLP has no means of informing the NTLP of the logical flow identification after the transformation has been applied and thus the NTLP can not look up the corresponding session. After identifying an affected context the QoS-NSLP queries the Flow Info Service for the new MIP6 state, compares it to the state of the reservation and—if required—sends messages to update the reservation accordingly.

As described in Chapter 3.3, the receiver of a reservation has limited possibilities to react if it receives a reservation for a logical flow for which the local Flow Info Service indicates a different actual flow. For the moment, our implementation simply assumes that the sender has the correct information and no mobility processing is applied for received messages. Even for the receiver-initiated case, the flow receiver depends on the flow sender to issue a new QUERY with the updated flow information. This again greatly simplifies the code as only the flow sender is aware of the logical flow and only it must react to mobility events. All other QoS nodes simply react to QoS-NSLP messages in the same way they react to ordinary, non-mobile messages.

The only changes to the normal processing of received messages concerns the handling of the old path after a new reservation has been established. In order to send a TEAR message on this path, the *Cross-Over-Nodes (CRN)* need to—in some situations—store the old flow MRI. Other than that, the normal re-routing processing readily provides the needed operations that are required after a re-routing caused by mobility. Even detecting the CRN can be done in the same way as detecting the branching or merging node after a normal re-routing event—by watching for changes of the SII-handle for the connected up- or downstream peer. It turned out that the provided code had some defects in this area as the draft documents have been in

flux about this in recent months. The required fixes for mobility, however, directly fixed the same issues for normal re-routing events.

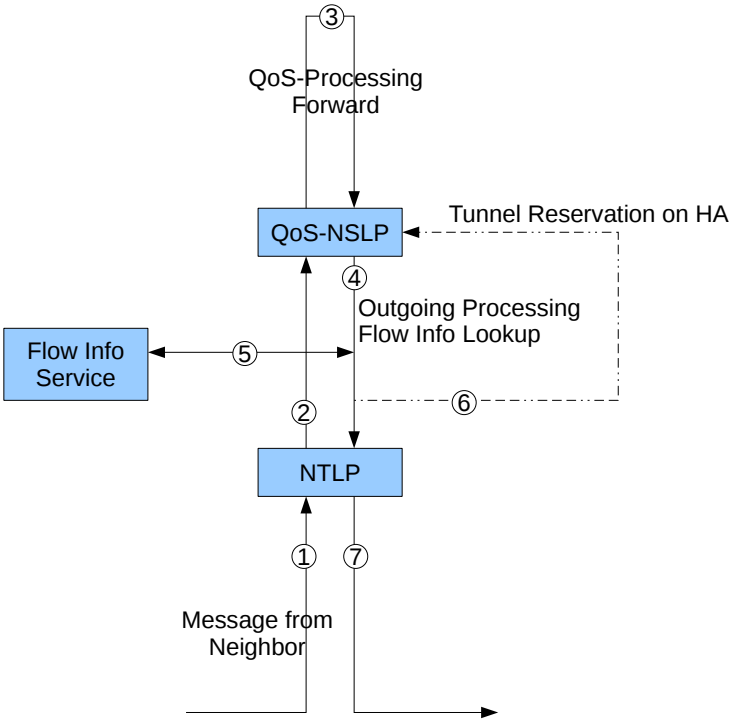


Figure 4.5: Processing of incoming messages on the HA

The one exception to the rule that there is no special mobility processing required upon receiving a message, is the Home Agent, in the case where the CN is the flow sender and tunnel mode is in effect. The processing is shown in Figure 4.5. In this scenario the HA is the entry point for the tunnel and has to take care of the tunnel reservation. The required processing, however, can be done after the HA forwards the initial RESERVE or QUERY message. Because of the way the statemodule is currently implemented, every forwarded message goes through the same processing as a locally created message. This way the code to create a tunnel reservation from the HA or the MN is the same, hooked into the processing of outgoing messages.

## 4.3 Source Address Selection

As discussed in the Analysis (Chapter 3.4) we have to take special provisions in order to select the correct address when building Message Routing State from or to the Mobile Node. When MIP6 is in use a normal socket will always bind to the HoA in order to keep the mobility transparent to the application. If we want to use the CoA instead, we have to manually bind the socket to it after creation. In order to do so, we have to look up the current CoA first. This problem is solved with the advanced API described in RFC 5040 [11]. Unfortunately, the current implementation available for the Linux kernel is broken, so we had to introduce a local alternative. Our implementation receives the interface that carries the CoA as a configuration parameter and uses standard API calls to get the address(es) configured on that interface. After checking that the address is neither the HoA nor a link-local address we have a good guess for the CoA. This approach is limited in

many ways, but good enough for our test environment. As long as no working RFC 5040 implementation is available, this is the only approach available. This lookup work-around is hidden behind a generic interface that will allow to replace it with a proper RFC 5040 API call once available.

We use the CoA, whenever we issue a GIST-Query or -Response, as the IP-source of the datagram as well as for the Interface Address in the Network Layer Information object that is used by the receiver of the message to build the Message Routing State or Messaging Association later on. Only if the Message Routing Information indicates that we really want to use the MRS/MA to signal from or to the HoA we use the HoA as source address directly. This makes sure that signaling messages for the tunneled flow enter the tunnel as they are supposed to. In addition, while the MN is at its home network there is not necessarily a CoA available so we have to build the MRS/MA from the HoA directly.

### 4.3.1 The Home Agent to Mobile Node GIST-Query Issue

For the issue described in Section 3.1.2 and depicted in Figure 3.3 where the GIST-Query from the HA to the MN does not enter the tunnel in some situations, we had to implement a local, environment specific solution. In order to avoid the resulting problems, we assign a secondary address to the Home Agent. This address is excluded from route optimization on every Mobile Node and used by the HA as the IP-source for GIST-Queries that are supposed to enter the tunnel. This requires additional configuration on the HA and MNs, but allows for a quick and efficient work around. Currently there is no other general approach known to tackle this issue otherwise.

# 5. Evaluation

To develop and evaluate the changes for mobility we are using an environment consisting of six virtual hosts connected to a "smart switch". This setup allows us to easily perform all possible mobility events. This chapter describes the setup in some detail and provides measurements of the signaling performance and delay.

## 5.1 The Testing Environment

Our testing environment consists of six virtual hosts: The Mobile Node, Home Agent and Correspondent Node as well as three Access Routers. The Home Agent and Access Routers are connected as shown in Figure 5.1 to build a topology that allows us to exercise various movement patterns. The Mobile Node moves between the Home Network and the three access networks, while the Correspondent Node is located in the access network provided by Access Router 1. Each Access Router as well as the Home Agent has installed static routes to the other routers and to the various networks provided by them. Each AR acts as default router in its network and sends fast (minimum delay as per RFC3775 [5], 0.03-0.07s) Router Advertisements—indicated by the circles in the figure. The MN and CN obtain an address in the network they are located in at the moment—indicated by the squares. In addition the MN has a HomeAddress in the Home Network that stays with it and is used for all reservations as logical source or destination. All virtual hosts run a slightly modified version of the Linux kernel with patches from the USAGI project in order to provide mobility functionality.

The topology is setup on a smart switch by bridging VLAN interfaces exported from the virtual environment. The smart switch runs the FreeBSD operating system and Wireshark to obtain packet capture dumps of the complete network traffic in the whole topology. This provides easy access to the complete signaling message exchange between the nodes and has proven to be a very powerful tool during initial prototyping and debugging. It also allows us to obtain accurate and easy measurements of the delay between a handover event and the resulting signaling exchange. In contrast to solutions where packet dumps are captured at each individual node, we do not have to correlate the individual dumps and need not account for clock

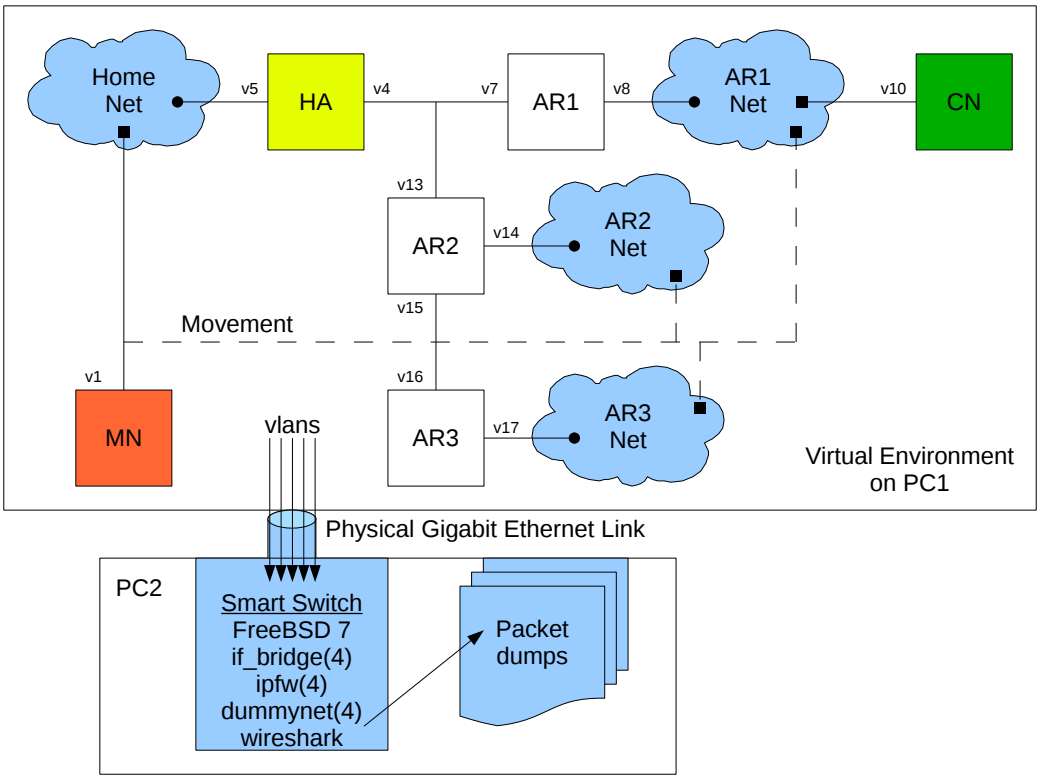


Figure 5.1: Test environment network layout

skew on the various nodes. As the smart switch is running on real hardware and all packets have to travel over a physical wire (twice), we are also not in danger of beneficial virtualization effects on the measurements and can assume that we obtain an upper bound for measurements performed on real hardware. We will, however, only be able to measure the time difference as observed on the smart switch. This is not a problem as the delay between the smart switch and the virtual nodes is well within the sub-microsecond range, while our measurements—as we will see later in the chapter—are up to four magnitudes larger.

## 5.2 Functional Evaluation

To evaluate if our implementation does perform the signaling as we have designed, we look at packet capture dumps from the smart switch and compare them with what we have drawn up before. The following figures are a visualization of the packet dump quoted—in full—in the appendix.

### 5.2.1 MN Sender, Sender-Initiated Reservation

The scenario we walk through in its entirety is the following: The MN is the flow sender creating a sender-initiated reservation from the HoA to the CN. Initially the MN is at its home network. The creation of the initial reservation is shown in Figure 5.2 (cf. Appendices A.1.1 and A.1.2 for the packet dump).

After a while the MN moves to AR3, obtains a new CoA, updates the mobility bindings with the HA and the CN and the reservation. This is shown in Figure 5.3 (cf. Appendix A.1.3-A.1.4). QoS-NSLP messages with the old MRI are drawn

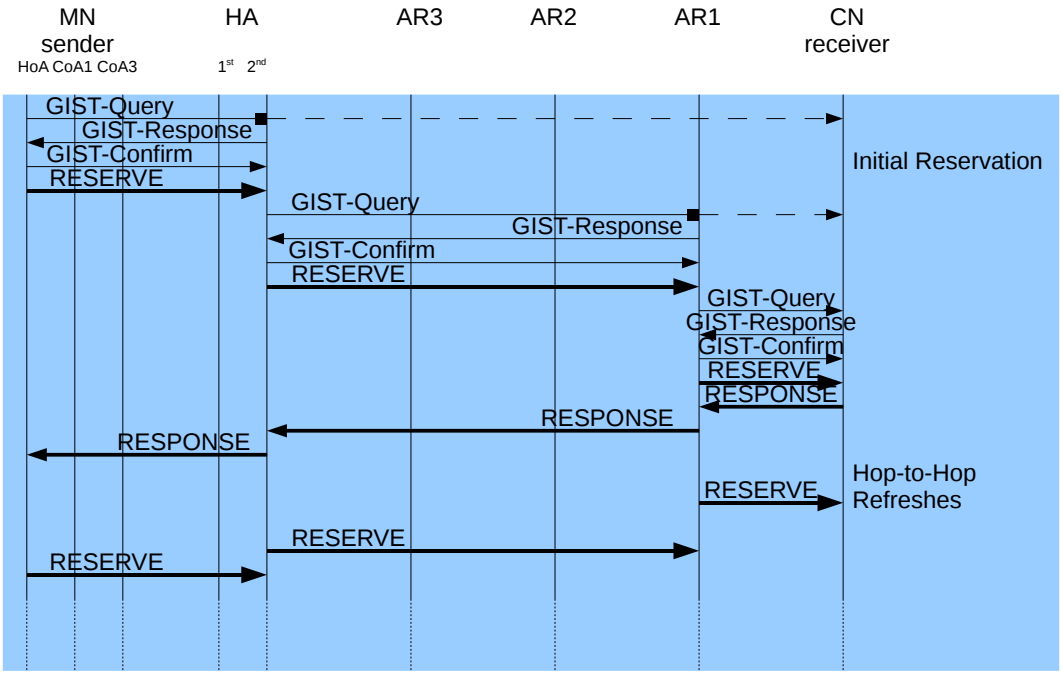


Figure 5.2: Initial Reservation Setup

dashed and marked with a "*" in the packet dump in the appendix. Note that the actual packet dump shows a short period of tunnel mode operation while the MN has not yet collected the tokens required to update the binding with the CN. As discussed before, we are not creating a reservation for the tunnel flow immediately and route-optimization is established quickly enough.

Figure 5.3 also illustrates our understanding of the signaling delay. We regard the time between receiving the *Binding Acknowledgement* (BA) from the CN and the final RESPONSE at the MN as the setup delay. Respectively, the time between receiving the BA at the MN and the RESERVE(TEAR) at the CRN is the tear down delay. This measurement points gives the time that is used by only the signaling application to negotiate. We purposely neglect the additional delay that is incurred by the MobileIPv6 signaling as well as the time the MIP6d needs to detect the new CoA. It is out of the scope of this work to optimize these times. It should be noted, however, that using the binding acknowledgement as trigger for the reservation update is a quite pessimistic approach. While this gives the earliest point in time when the new CoA can be used for route-optimized communication, there is room for optimization by taking a more optimistic approach. For instance, we could use sending of the binding update to the CN as the trigger, assuming that the CN will indeed acknowledge the binding. This way we would save a full round trip time. As our focus lies on constructing a functional prototype we leave these optimizations for further work in this area.

In the example dump we are looking at the following packets for the setup delay:

```
212  23.720472     1      CN(ar1)      MN(ar3)    MIPv6 Binding Acknowledgement
260  23.768795     1     AR3(ar3)      MN(ar3)     GIST Data, RESPONSE
```

We calculate a setup delay of under 50ms to update a reservation spanning five NSLP hops. During this test we did not add artificial delay (hop-to-hop round trip

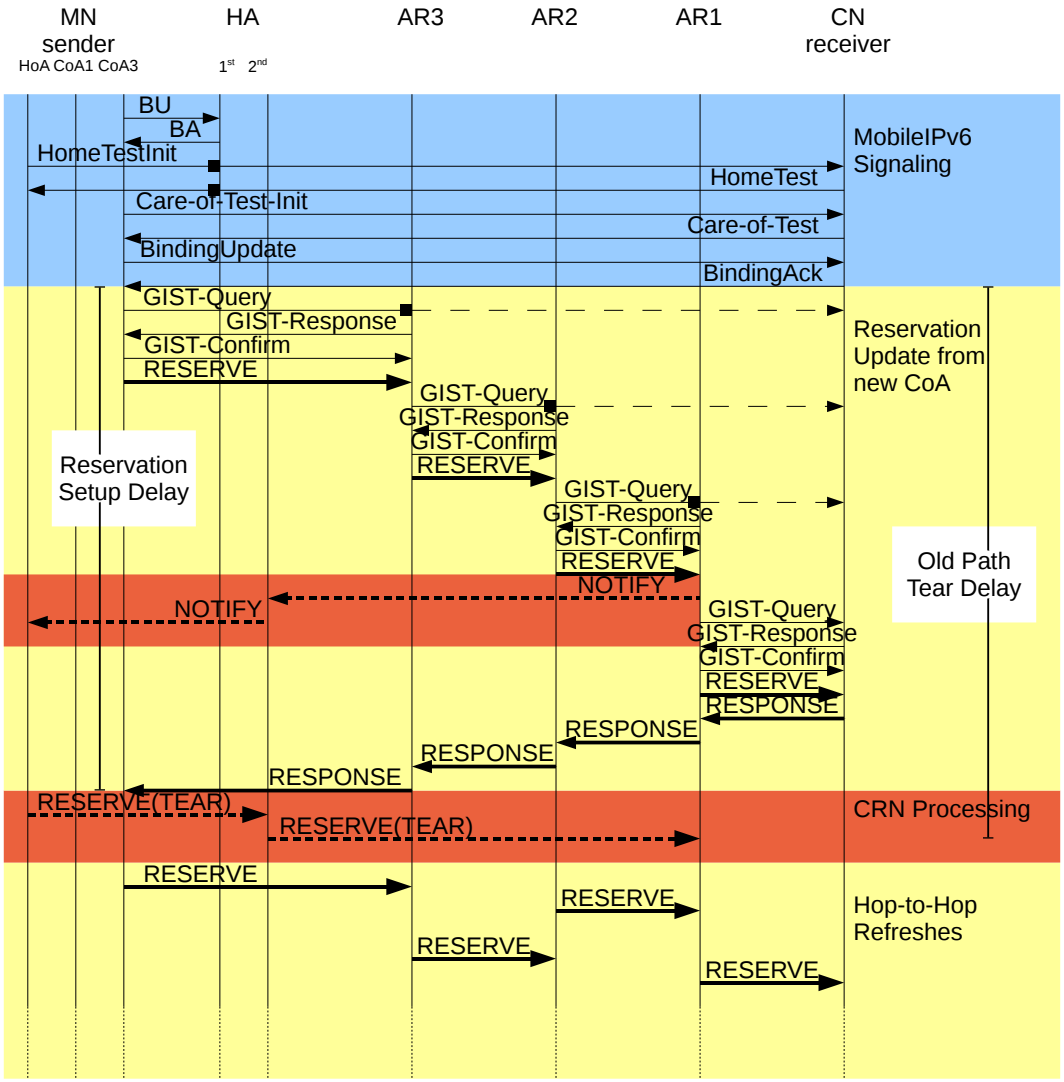


Figure 5.3: Handover to AR3 and Reservation Update

times are below 1ms) at the smart switch, so this number gives a good idea of the processing overhead alone. We should note that the times given by the packet dump are the times the respective packets are observed at the smart switch and not at the final destination, but since we are looking for time differences between two packets this does not matter. The time between the smart switch and the final destination can assumed to be uniform with the tiny absolute delay between smart switch and the virtual nodes as compared to the measured difference.

For the tear down delay we have a special case after leaving home. The MN can still communicate with the HA from the HoA and thusly issue an immediate tear. We measure roughly 52ms to tear a reservation spanning three NSLP hops:

```
212  23.720472     1      CN(ar1)      MN(ar3)    MIPv6 Binding Acknowledgement
268  23.772840     7     HA(inet)    AR1(inet)     GIST Data, RESERVE(TEAR)*
```

After some time settling at AR3 the MN moves to AR1 from where it can communicate with the CN directly. We choose this movement pattern to evaluate the longest and shortest reservation in terms of NSLP hops in our environment. The resulting exchange is shown in Figure 5.4 (cf. Appendix A.1.6).

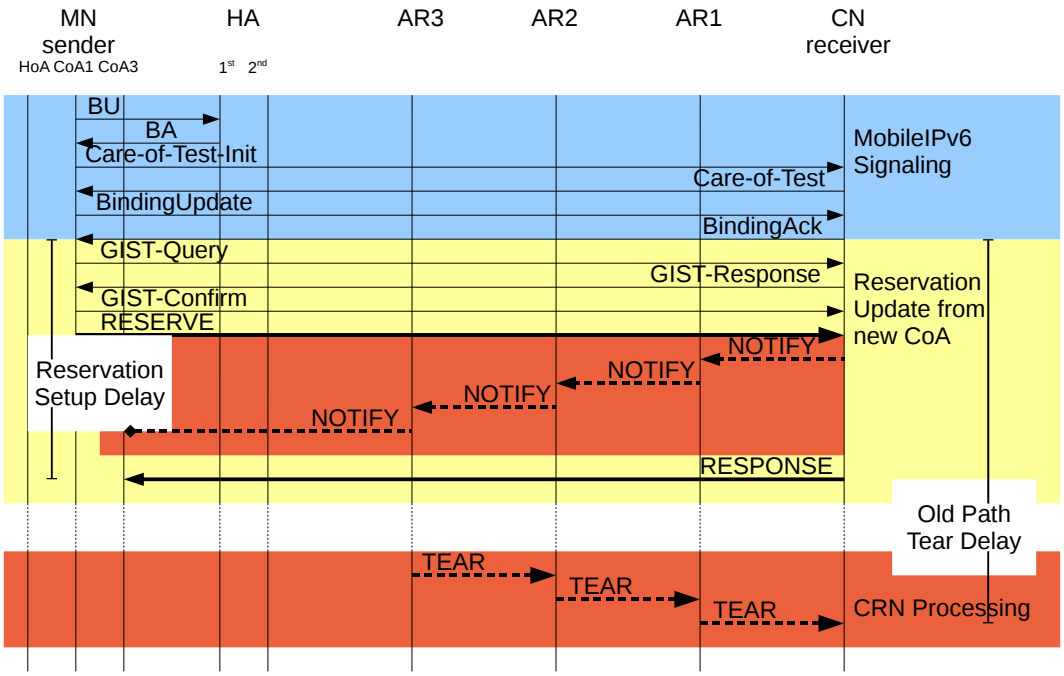


Figure 5.4: Handover to AR1 and Reservation Update

This time we observe the normal CRN processing in case of a sender-initiated reservation after movement of the flow sender. AR3 is trying to deliver the NOTIFY to the MN at the old CoA and fails. After some time during which refreshing RESERVEs are sent for the new and the old path at the same time (cf. Appendix A.1.7, not show in the figure), AR3 realizes that it is the dead-end and sends a TEAR down the old path (cf. Appendix A.1.8). The TEAR is forwarded all the way down to the CN which is also the CRN and stops forwarding of the TEAR from the old path silently.

Calculating the delays for this case we have to look at the following packets:

```
392   60.753212     1      CN(ar1)      MN(ar1)    MIPv6 Binding Acknowledgement
404   60.760651     1      CN(ar1)      MN(ar1)     GIST Data, RESPONSE
455   71.547837    10     AR1(ar1)      CN(ar1)     GIST Data, RESERVE(TEAR)*
```

We measure 7.4ms for the setup delay for a one hop reservation and almost 11s for the tear delay. The latter only depends on the lifetime of the routing state between the MN at the old CoA and AR3. The process could be sped up by requesting an acknowledgement at the QoS-NSLP level for the NOTIFY message. This way the QoS-NSLP at AR3 would realize much faster that the MN is no longer available and could free the resources on the old path much earlier. The QoS-NSLP draft includes a mechanism to request hop-to-hop acknowledgments so this is another optimization that my be tackled in future work.

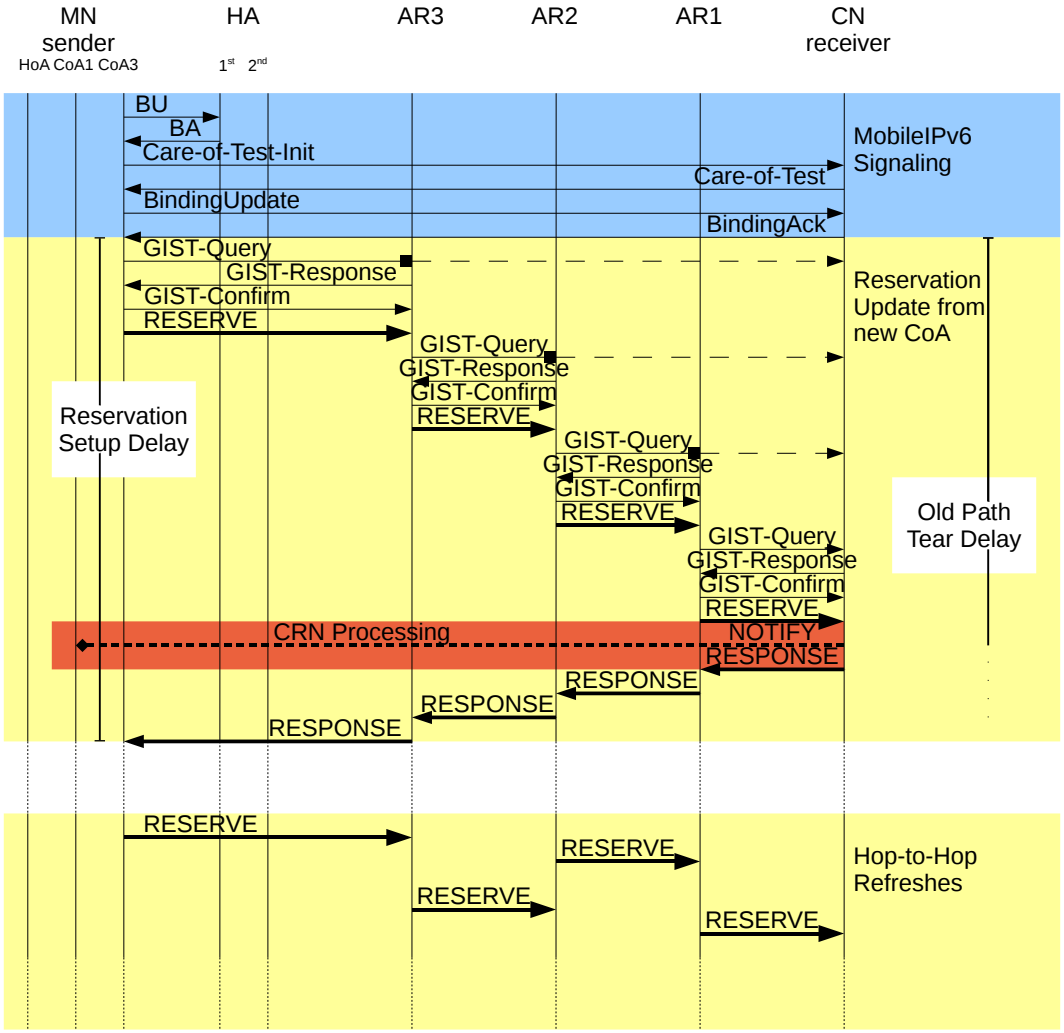


Figure 5.5: Handover AR1 back to AR3 and Reservation Update

Now we have a handover back to AR3. This is interesting in order to look at the CRN processing at the receiver (the CN in this scenario). As can be seen in Figure 5.5 (cf. Appendix A.1.10) the receiving node performs the normal CRN processing by sending a NOTIFY on the old path as soon as it receives the RESERVE from a new upstream node (change from MN to AR1). This NOTIFY message does not reach the MN as it is no longer available on CoA1. There is no further tear, as there

is in fact no old path left. The new reservation is otherwise updated and refreshed as normal.

Finally, in the packet dump shown in the appendix (A.1.13 through A.1.15) we have a handover back to the home network. There is nothing special about this except that the new CoA is the HoA and all nodes react appropriately. Please refer to the textual packet dump for details.

### 5.2.2 CN Sender, Sender-Initiated Reservation

Now we invert the direction of the flow. The CN becomes the flow sender and the MN the flow receiver, still using sender-initiated reservation. This is interesting because in this case a movement will result in a change of the downstream peer of the CRN allowing for a direct tear down of the old path. Figure 5.6 (cf. A.2) shows process of a handover for the MN moving from AR3 to AR1. This is an extract from the benchmark dump discussed in detail further down.

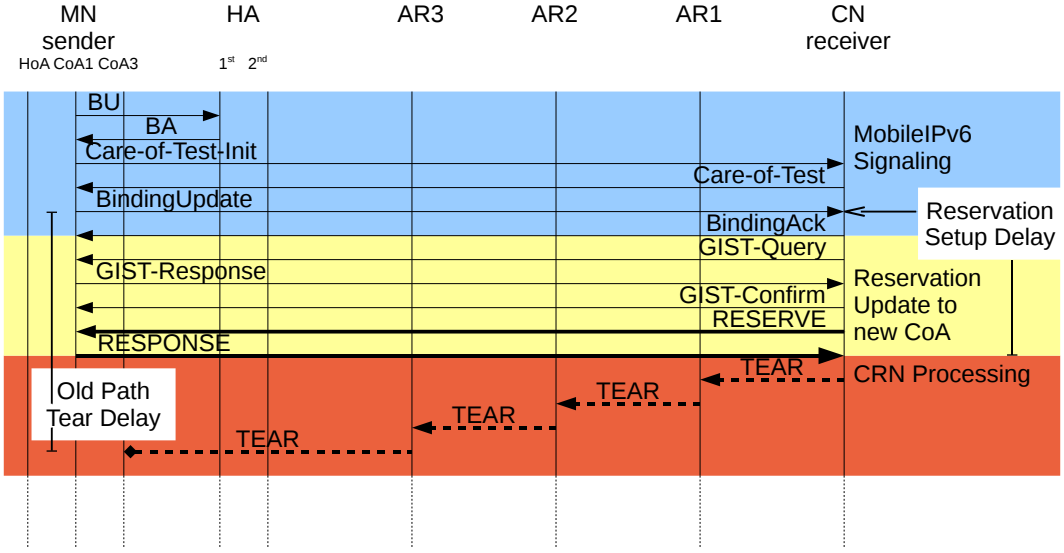


Figure 5.6: Handover AR1 back to AR3 and Reservation Update

As shown in the figure, the MIP6 update procedure works the same as before from the MN to the CN, but the reservation is done in the opposite direction. In order to obtain the same measurement points for the setup and tear down delay, we now use the time when the CN receives the binding update message as the start of the measured time. The CN is also a CRN—it sees a change in its downstream peer from AR1 to MN as it receives the GIST-Response. At that time, the CRN saves the old MRI (CN to CoA3) and sends a TEAR message down the old path as soon as the RESPONSE is processed and thus the new reservation is in place. The processing for a movement from AR1 to AR3 is very similar, only this time the TEAR message is sent to the MN at CoA1 directly (where it is no longer available).

To calculate the delay we have to look at the following packets:

```
 12   0.009932    10      MN(ar1)      CN(ar1)    MIPv6 Binding Update
 25   0.020173     1      MN(ar1)      CN(ar1)     GIST Data, RESPONSE
 32   0.025687    16   AR2(inet2)   AR3(inet2)     GIST Data, RESERVE(TEAR)*
```

We measure 10ms for the setup over two NSLP hops and 16ms for the tear down over four hops.

### 5.2.3 Receiver-Initiated Reservations

The receiver-initiated case is not that much different compared to the sender-initiated case, only that the first message after the GIST handshake is a QUERY instead of a RESERVE. Unfortunately, this causes severe confusion in the SII-handle change processing and thus in the CRN detection and processing. We were not able to make CRN processing work on the flow sender or receiver without profoundly reworking the complete handling of receiver-initiated reservations as a whole. It was decided to accept the missing tear-down of the old path in this case. In cases where the CRN is a QNE (and not QNI/QNR) the CRN processing is working as expected.

## 5.3 Signaling Performance Benchmarks

We evaluate the signaling performance based on the time between receiving the BA/BU on the MN/CN respectively and the time when the final RESPONSE for the new reservation is received. These are the same numbers that are mentioned in the functional evaluation above.

We sample this time over 50 consecutive movements of the MN between AR3 to AR2 to AR1 and back. The resulting reservations span either a single hop (MN to CN), four (MN, AR2, AR1, CN) or five hops (MN, AR3, AR2, AR1, CN). We also measure the time it takes before the old path is properly torn down. This is only measured for the old path between AR2 and AR3 after a movement from AR3 to AR2 as there is not necessarily a tear down for the short path.

The following Table gives the median[1] over the 50 runs[2]. Figure 5.7 provides a visual representation of these numbers.

| Testcase | AR1 setup | AR2 setup | AR3 setup | tear |
|---|---|---|---|---|
| MN sender, sender-initiated | 11.8ms | 26.9ms | 37.5ms | 20.6s |
| CN sender, sender-initiated | 13.3ms | 27.4ms | 40.3ms | 26.8ms |
| MN sender, receiver-initiated | 11.3ms | 29.0ms | 43.1ms | 28.0ms |
| CN sender, receiver-initiated | 12.5ms | 33.4ms | 42.2ms | 31.9ms |

Table 5.1: Reservation Setup and Old Path Tear Delay After Movement

Again these tests have been performed without adding artificial delay at the smart switch and a resulting round trip time between the virtual hosts of under 1ms. During these tests we identified the problem in our implementation regarding the CRN operation on the flow sender or receiver in receiver-initiated mode. Due to problems with the current state of the SII-handle change processing—already mentioned at the end of Section 4.2—we were unable to fix this particular problem. As a result we do not issue explicit tear down messages on the old path in these scenarios. The

[1] We prefer the median over the arithmetic average to be more resilient to outliers
[2] Please refer to Appendix B for the raw numbers

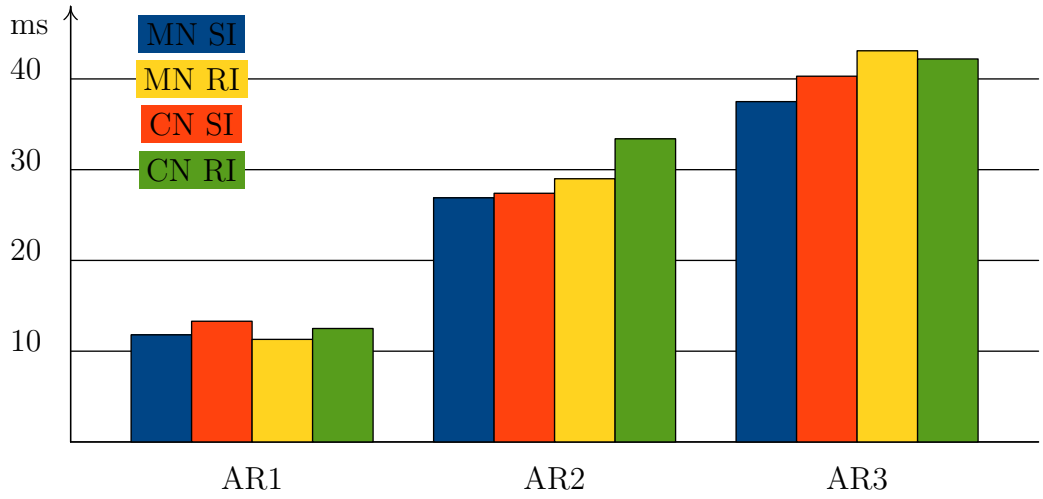


Figure 5.7: Setup Delay for Different Hop-Counts and Modes

impact of this is limited, however, as the old path will time out eventually. As mentioned earlier, this is not a problem when the CRN is a QNE.

The fact that there is little to no difference between the sender- and receiver-initiated case while at AR1 is also due to the missing CRN processing on the receiver. The additional overhead and half round trip for the QUERY seems to be roughly the same as the overhead to generate the NOTIFY or RESERVE(TEAR) message that is sent before the RESPONSE in sender-initiated mode.

The relative high number for the tear down delay when the MN is the flow sender and QNI is again due to the way the CRN processing works. As discussed in during the example above, the process could be sped up using hop-to-hop acknowledgments for the NOTIFY message at the QoS-NSLP level.

Now we introduce additional delay on the smart switch to see how much of the processing overhead can be amortized in typical Internet scenarios. We configured 50ms symmetric delay between AR1 and AR2 and another 25ms between AR2 and AR3. The access networks remain unchanged. We use the IPFW2 packet filter and dummynet[12] to simulate these link properties. With these settings we measure 104ms round trip time between the MN and the CN while the MN is at the access network of AR2, and 160ms while at AR3. The slight difference from the configured values (100ms/150ms respectively) is due to scheduling granularity and additional overhead for the queueing on the smart host. The numbers were obtained using the `ping6(8)` utility with 600 samples over a one minute interval, reporting a standard deviation of under 2ms in both cases.

These numbers allow us to project the theoretical, optimal delay for the reservation setup. For sender-initiated reservations we count: GIST-Query and GIST-Response (one round trip), GIST-Confirm/RESERVE and RESPONSE (one round trip): Two round trips total. In sender-initiated mode there is an additional half round trip for the QUERY i.e. two and a half round trips total:

Our measurements in face of the delay give the following numbers, this time over 50 movements between AR2 and AR3—again the median is given:

| Testcase | AR2 optimal delay | AR3 optimal delay |
|---|---|---|
| sender-initiated | 208ms (2RTT) | 320ms (2RTT) |
| receiver-initiated | 260ms (2.5RTT) | 400ms (2.5RTT) |

Table 5.2: Theoretical Optimal Reservation Setup Delay

| Testcase | AR2 setup delay | AR3 setup delay |
|---|---|---|
| MN sender, sender-initiated | 228.8ms | 348.9ms |
| CN sender, sender-initiated | 231.5ms | 353.1ms |
| MN sender, receiver-initiated | 285.5ms | 429.3ms |
| CN sender, receiver-initiated | 287.8ms | 429.1ms |

Table 5.3: Measured Reservation Setup Delay

Or in terms of overhead compared to the theoretical, optimal performance see Table 5.4.

| Testcase | AR2 setup overhead | AR3 setup overhead |
|---|---|---|
| MN sender, sender-initiated | 20.8ms | 28.9ms |
| CN sender, sender-initiated | 23.5ms | 33.1ms |
| MN sender, receiver-initiated | 25.5ms | 29.3ms |
| CN sender, receiver-initiated | 27.8ms | 29.1ms |

Table 5.4: Difference Between Measurements and Theoretic Optimum

Figure 5.8 visualizes the difference between the optimal and measured setup delay.

These numbers are slightly better than what we obtained without delay (cf. Table 5.1) where we did not attempt to remove the 1-2ms round trip times. The fact that they are otherwise very similar suggests that these numbers are non-dependent on the hop-to-hop delay and represent the static processing overhead.

Finally we look at the time it takes to process the mobility trigger event by measuring the time between the BindingUpdate/BindingAcknowledgement and GIST-Query that starts the signaling process. The following numbers are collected over all movement events from all benchmarks performed. There is no difference between sender- or receiver-initiated mode for these numbers.

| Testcase | Mobility trigger delay |
|---|---|
| MN | 1.93ms |
| CN | 2.54ms |

Table 5.5: Mobility Trigger Processing Overhead

The absolute time is very small and—as we are only looking at externally observable events—includes not only the trigger event processing itself, but also the generation

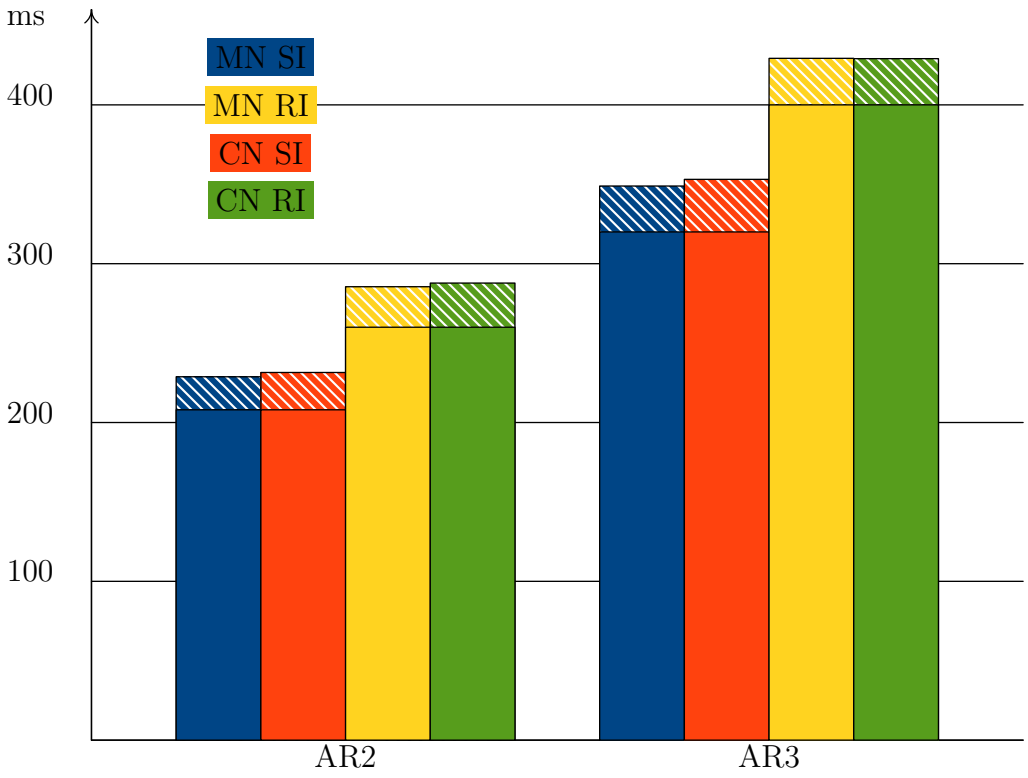


Figure 5.8: Setup Delay and Overhead

of the Query message that has to be done regardless. The slightly longer delay on the CN is due to the fact that the MIP6d has to prepare and send the BindingAcknowledgement message before it issues the trigger.

## 5.4 Summary

We ran an extensive testsuite to verify other test cases such as the switch from tunnel mode to route-optimization and vice versa, leaving and returning home as well as different CRN scenarios. With the exception of the aforementioned issue with CRN processing in receiver-initiated mode all these tests were successfull. The signaling performance numbers from the previous section show that even our proof-of-concept implementation is able to provide sensible performance and that the suggested interface with the mobility management does not pose a bottleneck.

# 6. Summary and further directions

In this work we were able to show that the NSIS protocol drafts for the QoS-NSLP and GIST are well-equipped to deal with mobility. No changes at protocol level were required to implement functional mobility support. The only change from the drafts is the introduction of a new type for the NetworkNotification-API call to propagate mobility events from GIST to NSLPs. Most other issues identified during this work revolve around implementation problems and can only be solved with regard to the specific implementation environment. We also identified one serious issue in relation to tunnel mode (see Section 3.1.2 for details). There is currently no satisfying solution to this problem available. We introduce a possible work-around in Section 4.3.1 that resolves the problem at the cost of additional configuration and administrative overhead. The mobility-draft [13] gives a good overview of the general problem of signaling in mobile environments, but—in our opinion—does not give sufficient detail in some areas where implementation issues are concerned. It would help if some of the points presented in the early draft [16] where reiterated in the current document—especially the discussion of different flow identifiers.

Our implementation currently serves as a proof-of-concept and there are a number of possible optimizations for further work in this area: A more optimistic trigger point could be used for the mobility events. Instead of using full message routing state setup, the QoS-NSLP data could be transported in the GIST-Query. Experiments with Message Associations should be conducted, too. Tear-down of the old path can be sped up. In addition, as our work is only concerned with inter-domain handovers, there are a number of possible optimizations when looking at intra-domain handovers, too.

We were able to provide a fully mobility-aware QoS-Signaling Application with relatively little changes to a pre-existing, not-mobility-aware implementation, showing that the work in the NSIS working group was indeed conducted with mobility in mind.

# A. Evaluation Packet Dumps

## A.1 MN Sender in Sender-Initiated Mode

The following packet dump shows the complete packet exchange for a sender-initiated reservation in the topology described in Chapter 5.1 and shown in Figure 5.1. The VLAN numbers quoted below are also shown in that figure. The dump is split up into logical parts as described in Chapter 5.2. Source and destination addresses are replaced with symbolic names of the node and network they belong to. The "inet"-network refers to the network that connects the HA, AR1 and AR2. "inet2" is the interconnect between AR2 and AR3. As described earlier we had to assign two addresses to the HA in order to avoid problems with the creation of tunneled routing state from the HA to the MN. The primary address is "HA(home:1)". "HA(home:2)" is the address used to establish routing state. The MN's HoA is "MN(home)".

### A.1.1 Initial Reservation

```
No.       Time  VLAN       Source  Destination Protocol Info
  1   0.000000     1     MN(home)      CN(ar1)     GIST Query
    Message routing information object
        Object Header
        MRI type: 0
        Flags: 0x00
        IP version: 6
        Flags: 0x800 (P)
        Source address: MN(home)
        Destination address: CN(ar1)
        Source address prefix: 0
        Destination address prefix: 0
        Protocol: UDP (0x11)
  2   0.000007     5     MN(home)      CN(ar1)     GIST Query
  3   0.003753     5   HA(home:2)     MN(home)     GIST Response
  4   0.003757     1   HA(home:2)     MN(home)     GIST Response
  5   0.009123     1     MN(home)   HA(home:2)     GIST Confirm
  6   0.009127     5     MN(home)   HA(home:2)     GIST Confirm
  7   0.009130     1     MN(home)   HA(home:2)     GIST Data, RESERVE
  8   0.009134     5     MN(home)   HA(home:2)     GIST Data, RESERVE
  9   0.013703     4     HA(inet)      CN(ar1)     GIST Query
```

| | | | | | |
|---|---|---|---|---|---|
| 10 | 0.013712 | 7 | HA(inet) | CN(ar1) | GIST Query |
| 11 | 0.017781 | 7 | AR1(inet) | HA(inet) | GIST Response |
| 12 | 0.017790 | 4 | AR1(inet) | HA(inet) | GIST Response |
| 13 | 0.022205 | 4 | HA(inet) | AR1(inet) | GIST Confirm |
| 14 | 0.022210 | 7 | HA(inet) | AR1(inet) | GIST Confirm |
| 15 | 0.023403 | 4 | HA(inet) | AR1(inet) | GIST Data, RESERVE |
| 16 | 0.023408 | 7 | HA(inet) | AR1(inet) | GIST Data, RESERVE |
| 17 | 0.029141 | 8 | AR1(ar1) | CN(ar1) | GIST Query |
| 18 | 0.029149 | 10 | AR1(ar1) | CN(ar1) | GIST Query |
| 19 | 0.041884 | 10 | CN(ar1) | AR1(ar1) | GIST Response |
| 20 | 0.041888 | 8 | CN(ar1) | AR1(ar1) | GIST Response |
| 21 | 0.042912 | 8 | AR1(ar1) | CN(ar1) | GIST Confirm |
| 22 | 0.042917 | 10 | AR1(ar1) | CN(ar1) | GIST Confirm |
| 23 | 0.043415 | 8 | AR1(ar1) | CN(ar1) | GIST Data, RESERVE |
| 24 | 0.043420 | 10 | AR1(ar1) | CN(ar1) | GIST Data, RESERVE |
| 25 | 0.047139 | 10 | CN(ar1) | AR1(ar1) | GIST Data, RESPONSE |
| 26 | 0.047144 | 8 | CN(ar1) | AR1(ar1) | GIST Data, RESPONSE |
| 27 | 0.048494 | 7 | AR1(inet) | HA(inet) | GIST Data, RESPONSE |
| 28 | 0.048500 | 4 | AR1(inet) | HA(inet) | GIST Data, RESPONSE |
| 29 | 0.050877 | 5 | HA(home:2) | MN(home) | GIST Data, RESPONSE |
| 30 | 0.050884 | 1 | HA(home:2) | MN(home) | GIST Data, RESPONSE |

### A.1.2 Hop-to-Hop Refreshes

| | | | | | |
|---|---|---|---|---|---|
| 31 | 4.854890 | 1 | MN(home) | HA(home:2) | GIST Data, RESERVE |
| 32 | 4.854898 | 5 | MN(home) | HA(home:2) | GIST Data, RESERVE |
| 33 | 4.916383 | 4 | HA(inet) | AR1(inet) | GIST Data, RESERVE |
| 34 | 4.916416 | 7 | HA(inet) | AR1(inet) | GIST Data, RESERVE |
| 35 | 6.499409 | 8 | AR1(ar1) | CN(ar1) | GIST Data, RESERVE |
| 36 | 6.499416 | 10 | AR1(ar1) | CN(ar1) | GIST Data, RESERVE |
| 37 | 8.500395 | 8 | AR1(ar1) | CN(ar1) | GIST Data, RESERVE |
| 38 | 8.500405 | 10 | AR1(ar1) | CN(ar1) | GIST Data, RESERVE |
| 39 | 9.854521 | 1 | MN(home) | HA(home:2) | GIST Data, RESERVE |
| 40 | 9.854530 | 5 | MN(home) | HA(home:2) | GIST Data, RESERVE |
| 41 | 9.918490 | 4 | HA(inet) | AR1(inet) | GIST Data, RESERVE |
| 42 | 9.918501 | 7 | HA(inet) | AR1(inet) | GIST Data, RESERVE |
| 43 | 11.500673 | 8 | AR1(ar1) | CN(ar1) | GIST Data, RESERVE |
| 44 | 11.500682 | 10 | AR1(ar1) | CN(ar1) | GIST Data, RESERVE |
| 45 | 13.855628 | 1 | MN(home) | HA(home:2) | GIST Data, RESERVE |
| 46 | 13.855636 | 5 | MN(home) | HA(home:2) | GIST Data, RESERVE |
| 47 | 14.530320 | 8 | AR1(ar1) | CN(ar1) | GIST Query |
| 48 | 14.530330 | 10 | AR1(ar1) | CN(ar1) | GIST Query |
| 49 | 14.531332 | 10 | CN(ar1) | AR1(ar1) | GIST Response |
| 50 | 14.531338 | 8 | CN(ar1) | AR1(ar1) | GIST Response |
| 51 | 14.532047 | 8 | AR1(ar1) | CN(ar1) | GIST Confirm |
| 52 | 14.532051 | 10 | AR1(ar1) | CN(ar1) | GIST Confirm |
| 53 | 14.919215 | 4 | HA(inet) | AR1(inet) | GIST Data, RESERVE |
| 54 | 14.919225 | 7 | HA(inet) | AR1(inet) | GIST Data, RESERVE |
| 55 | 15.500903 | 8 | AR1(ar1) | CN(ar1) | GIST Data, RESERVE |
| 56 | 15.500912 | 10 | AR1(ar1) | CN(ar1) | GIST Data, RESERVE |
| 57 | 17.918821 | 4 | HA(inet) | AR1(inet) | GIST Data, RESERVE |
| 58 | 17.918828 | 7 | HA(inet) | AR1(inet) | GIST Data, RESERVE |
| 59 | 18.501543 | 8 | AR1(ar1) | CN(ar1) | GIST Data, RESERVE |
| 60 | 18.501548 | 10 | AR1(ar1) | CN(ar1) | GIST Data, RESERVE |

### A.1.3 Handover to AR3 - Tunnel Mode

| | | | | | |
|---|---|---|---|---|---|
| 61 | 18.621585 | 1 | MN(ar3) | HA(home:1) | MIPv6 Binding Update |
| 62 | 18.621590 | 17 | MN(ar3) | HA(home:1) | MIPv6 Binding Update |

```
 63  18.621809    16      MN(ar3)   HA(home:1)    MIPv6 Binding Update
 64  18.621815    15      MN(ar3)   HA(home:1)    MIPv6 Binding Update
 65  18.622065    13      MN(ar3)   HA(home:1)    MIPv6 Binding Update
 66  18.622071     4      MN(ar3)   HA(home:1)    MIPv6 Binding Update
 67  18.710294     1    MN(home)      CN(ar1)     GIST Query
 68  18.710301    17    MN(home)      CN(ar1)     GIST Query
 69  18.710579    16    MN(home)      CN(ar1)     GIST Query
 70  18.710584    15    MN(home)      CN(ar1)     GIST Query
 71  18.710977    13    MN(home)      CN(ar1)     GIST Query
 72  18.710983     4    MN(home)      CN(ar1)     GIST Query
 73  18.711700     1    MN(home)      CN(ar1)    MIPv6 Home Test Init
 74  18.711704    17    MN(home)      CN(ar1)    MIPv6 Home Test Init
 75  18.711879    16    MN(home)      CN(ar1)    MIPv6 Home Test Init
 76  18.711883    15    MN(home)      CN(ar1)    MIPv6 Home Test Init
 77  18.711991     1      MN(ar3)     CN(ar1)    MIPv6 Care-of Test Init
 78  18.711994    17      MN(ar3)     CN(ar1)    MIPv6 Care-of Test Init
 79  18.712158    16      MN(ar3)     CN(ar1)    MIPv6 Care-of Test Init
 80  18.712161    15      MN(ar3)     CN(ar1)    MIPv6 Care-of Test Init
 81  18.712425    13    MN(home)      CN(ar1)    MIPv6 Home Test Init
 82  18.712428     4    MN(home)      CN(ar1)    MIPv6 Home Test Init
 83  18.712535    13      MN(ar3)     CN(ar1)    MIPv6 Care-of Test Init
 84  18.712538     7      MN(ar3)     CN(ar1)    MIPv6 Care-of Test Init
 85  18.713657     8      MN(ar3)     CN(ar1)    MIPv6 Care-of Test Init
 86  18.713661    10      MN(ar3)     CN(ar1)    MIPv6 Care-of Test Init
 87  18.714244    10      CN(ar1)     MN(ar3)    MIPv6 Care-of Test
 88  18.714247     8      CN(ar1)     MN(ar3)    MIPv6 Care-of Test
 89  18.715486     7      CN(ar1)     MN(ar3)    MIPv6 Care-of Test
 90  18.715491    13      CN(ar1)     MN(ar3)    MIPv6 Care-of Test
 91  18.715834    15      CN(ar1)     MN(ar3)    MIPv6 Care-of Test
 92  18.715840    16      CN(ar1)     MN(ar3)    MIPv6 Care-of Test
 93  18.716047    17      CN(ar1)     MN(ar3)    MIPv6 Care-of Test
 94  18.716051     1      CN(ar1)     MN(ar3)    MIPv6 Care-of Test
 95  18.831495     4   HA(home:1)     MN(ar3)   ICMPv6 Unreachable (Port unreachable)
 96  18.831502    13   HA(home:1)     MN(ar3)   ICMPv6 Unreachable (Port unreachable)
 97  18.831602     4   HA(home:1)     MN(ar3)   ICMPv6 Unreachable (Port unreachable)
 98  18.831606    13   HA(home:1)     MN(ar3)   ICMPv6 Unreachable (Port unreachable)
 99  18.831790    15   HA(home:1)     MN(ar3)   ICMPv6 Unreachable (Port unreachable)
100  18.831795    16   HA(home:1)     MN(ar3)   ICMPv6 Unreachable (Port unreachable)
101  18.832054    17   HA(home:1)     MN(ar3)   ICMPv6 Unreachable (Port unreachable)
102  18.832059     1   HA(home:1)     MN(ar3)   ICMPv6 Unreachable (Port unreachable)
103  18.832321    15   HA(home:1)     MN(ar3)   ICMPv6 Unreachable (Port unreachable)
104  18.832325    16   HA(home:1)     MN(ar3)   ICMPv6 Unreachable (Port unreachable)
105  18.832500    17   HA(home:1)     MN(ar3)   ICMPv6 Unreachable (Port unreachable)
106  18.832505     1   HA(home:1)     MN(ar3)   ICMPv6 Unreachable (Port unreachable)
107  19.710184     1     MN(home)     CN(ar1)     GIST Query
108  19.710193    17     MN(home)     CN(ar1)     GIST Query
109  19.710408    16     MN(home)     CN(ar1)     GIST Query
110  19.710414    15     MN(home)     CN(ar1)     GIST Query
111  19.710658    13     MN(home)     CN(ar1)     GIST Query
112  19.710664     4     MN(home)     CN(ar1)     GIST Query
113  19.712659     4   HA(home:1)     MN(ar3)   ICMPv6 Unreachable (Port unreachable)
114  19.712664    13   HA(home:1)     MN(ar3)   ICMPv6 Unreachable (Port unreachable)
115  19.713315    15   HA(home:1)     MN(ar3)   ICMPv6 Unreachable (Port unreachable)
116  19.713320    16   HA(home:1)     MN(ar3)   ICMPv6 Unreachable (Port unreachable)
117  19.713523    17   HA(home:1)     MN(ar3)   ICMPv6 Unreachable (Port unreachable)
118  19.713527     1   HA(home:1)     MN(ar3)   ICMPv6 Unreachable (Port unreachable)
119  19.855937     1     MN(home)   HA(home:2)     GIST Data, RESERVE
120  19.855946    17     MN(home)   HA(home:2)     GIST Data, RESERVE
121  19.856194    16     MN(home)   HA(home:2)     GIST Data, RESERVE
```

```
122  19.856200    15     MN(home)   HA(home:2)     GIST Data, RESERVE
123  19.856489    13     MN(home)   HA(home:2)     GIST Data, RESERVE
124  19.856495     4     MN(home)   HA(home:2)     GIST Data, RESERVE
125  20.039062     4   HA(home:1)      MN(ar3)   ICMPv6 Unreachable (Port unreachable)
126  20.039070    13   HA(home:1)      MN(ar3)   ICMPv6 Unreachable (Port unreachable)
127  20.039372    15   HA(home:1)      MN(ar3)   ICMPv6 Unreachable (Port unreachable)
128  20.039379    16   HA(home:1)      MN(ar3)   ICMPv6 Unreachable (Port unreachable)
129  20.039611    17   HA(home:1)      MN(ar3)   ICMPv6 Unreachable (Port unreachable)
130  20.039616     1   HA(home:1)      MN(ar3)   ICMPv6 Unreachable (Port unreachable)
131  20.122215     1      MN(ar3)   HA(home:1)    MIPv6 Binding Update
132  20.122221    17      MN(ar3)   HA(home:1)    MIPv6 Binding Update
133  20.122444    16      MN(ar3)   HA(home:1)    MIPv6 Binding Update
134  20.122450    15      MN(ar3)   HA(home:1)    MIPv6 Binding Update
135  20.122760    13      MN(ar3)   HA(home:1)    MIPv6 Binding Update
136  20.122767     4      MN(ar3)   HA(home:1)    MIPv6 Binding Update
137  20.266539     4     HA(inet)      CN(ar1)     GIST Query
138  20.266547     7     HA(inet)      CN(ar1)     GIST Query
139  20.267871     7    AR1(inet)     HA(inet)     GIST Response
140  20.267877     4    AR1(inet)     HA(inet)     GIST Response
141  20.268995     4     HA(inet)    AR1(inet)     GIST Confirm
142  20.269000     7     HA(inet)    AR1(inet)     GIST Confirm
143  20.270258     4   HA(home:1)      MN(ar3)    MIPv6 Binding Acknowledgement
144  20.270263    13   HA(home:1)      MN(ar3)    MIPv6 Binding Acknowledgement
145  20.270489    15   HA(home:1)      MN(ar3)    MIPv6 Binding Acknowledgement
146  20.270495    16   HA(home:1)      MN(ar3)    MIPv6 Binding Acknowledgement
147  20.270672     4   HA(home:1)      MN(ar3)    MIPv6 Binding Acknowledgement
148  20.270677    13   HA(home:1)      MN(ar3)    MIPv6 Binding Acknowledgement
149  20.270709    17   HA(home:1)      MN(ar3)    MIPv6 Binding Acknowledgement
150  20.270714     1   HA(home:1)      MN(ar3)    MIPv6 Binding Acknowledgement
151  20.270967    15   HA(home:1)      MN(ar3)    MIPv6 Binding Acknowledgement
152  20.270972    16   HA(home:1)      MN(ar3)    MIPv6 Binding Acknowledgement
153  20.271195    17   HA(home:1)      MN(ar3)    MIPv6 Binding Acknowledgement
154  20.271200     1   HA(home:1)      MN(ar3)    MIPv6 Binding Acknowledgement
155  21.710926     1     MN(home)      CN(ar1)     GIST Query
156  21.710936    17     MN(home)      CN(ar1)     GIST Query
157  21.711167    16     MN(home)      CN(ar1)     GIST Query
158  21.711173    15     MN(home)      CN(ar1)     GIST Query
159  21.711459    13     MN(home)      CN(ar1)     GIST Query
160  21.711467     4     MN(home)      CN(ar1)     GIST Query
161  21.712898     4   HA(home:2)     MN(home)     GIST Response
162  21.712905    13   HA(home:2)     MN(home)     GIST Response
163  21.713084    15   HA(home:2)     MN(home)     GIST Response
164  21.713090    16   HA(home:2)     MN(home)     GIST Response
165  21.713266    17   HA(home:2)     MN(home)     GIST Response
166  21.713272     1   HA(home:2)     MN(home)     GIST Response
167  21.714195     1     MN(home)   HA(home:2)     GIST Confirm
168  21.714201    17     MN(home)   HA(home:2)     GIST Confirm
169  21.714368    16     MN(home)   HA(home:2)     GIST Confirm
170  21.714373    15     MN(home)   HA(home:2)     GIST Confirm
171  21.714619    13     MN(home)   HA(home:2)     GIST Confirm
172  21.714624     4     MN(home)   HA(home:2)     GIST Confirm
173  21.919103     4     HA(inet)    AR1(inet)     GIST Data, RESERVE
174  21.919115     7     HA(inet)    AR1(inet)     GIST Data, RESERVE
175  23.502293     8     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
176  23.502297    10     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
177  23.713281     1     MN(home)      CN(ar1)    MIPv6 Home Test Init
178  23.713286    17     MN(home)      CN(ar1)    MIPv6 Home Test Init
179  23.713563    16     MN(home)      CN(ar1)    MIPv6 Home Test Init
180  23.713569    15     MN(home)      CN(ar1)    MIPv6 Home Test Init
```

```
181  23.713784    13     MN(home)      CN(ar1)    MIPv6 Home Test Init
182  23.713788     4     MN(home)      CN(ar1)    MIPv6 Home Test Init
183  23.714116     4     MN(home)      CN(ar1)    MIPv6 Home Test Init
184  23.714120     7     MN(home)      CN(ar1)    MIPv6 Home Test Init
185  23.714290     8     MN(home)      CN(ar1)    MIPv6 Home Test Init
186  23.714295    10     MN(home)      CN(ar1)    MIPv6 Home Test Init
187  23.714779    10      CN(ar1)     MN(home)    MIPv6 Home Test
188  23.714784     8      CN(ar1)     MN(home)    MIPv6 Home Test
189  23.714950     7      CN(ar1)     MN(home)    MIPv6 Home Test
190  23.714956     4      CN(ar1)     MN(home)    MIPv6 Home Test
191  23.715488     4      CN(ar1)     MN(home)    MIPv6 Home Test
192  23.715493    13      CN(ar1)     MN(home)    MIPv6 Home Test
193  23.715676    15      CN(ar1)     MN(home)    MIPv6 Home Test
194  23.715682    16      CN(ar1)     MN(home)    MIPv6 Home Test
195  23.716097    17      CN(ar1)     MN(home)    MIPv6 Home Test
196  23.716102     1      CN(ar1)     MN(home)    MIPv6 Home Test
```

### A.1.4 Handover to AR3 - Route-optimized

```
197  23.716784     1      MN(ar3)      CN(ar1)    MIPv6 Binding Update
198  23.716789    17      MN(ar3)      CN(ar1)    MIPv6 Binding Update
199  23.717418    16      MN(ar3)      CN(ar1)    MIPv6 Binding Update
200  23.717422    15      MN(ar3)      CN(ar1)    MIPv6 Binding Update
201  23.717735    13      MN(ar3)      CN(ar1)    MIPv6 Binding Update
202  23.717739     7      MN(ar3)      CN(ar1)    MIPv6 Binding Update
203  23.718019     8      MN(ar3)      CN(ar1)    MIPv6 Binding Update
204  23.718024    10      MN(ar3)      CN(ar1)    MIPv6 Binding Update
205  23.718810    10      CN(ar1)      MN(ar3)    MIPv6 Binding Acknowledgement
206  23.718813     8      CN(ar1)      MN(ar3)    MIPv6 Binding Acknowledgement
207  23.719011     7      CN(ar1)      MN(ar3)    MIPv6 Binding Acknowledgement
208  23.719015    13      CN(ar1)      MN(ar3)    MIPv6 Binding Acknowledgement
209  23.720266    15      CN(ar1)      MN(ar3)    MIPv6 Binding Acknowledgement
210  23.720270    16      CN(ar1)      MN(ar3)    MIPv6 Binding Acknowledgement
211  23.720467    17      CN(ar1)      MN(ar3)    MIPv6 Binding Acknowledgement
212  23.720472     1      CN(ar1)      MN(ar3)    MIPv6 Binding Acknowledgement

213  23.722132     1      MN(ar3)      CN(ar1)     GIST Query
    Message routing information object
        Object Header
        MRI type: 0
        Flags: 0x00
        IP version: 6
        Flags: 0x800 (P)
        Source address: MN(ar3)
        Destination address: CN(ar1)
        Source address prefix: 128
        Destination address prefix: 0
        Protocol: UDP (0x11)
214  23.722137    17      MN(ar3)      CN(ar1)     GIST Query
215  23.726605    17     AR3(ar3)      MN(ar3)     GIST Response
216  23.726610     1     AR3(ar3)      MN(ar3)     GIST Response
217  23.732232     1      MN(ar3)     AR3(ar3)     GIST Confirm
218  23.732236    17      MN(ar3)     AR3(ar3)     GIST Confirm
219  23.732239     1      MN(ar3)     AR3(ar3)     GIST Data, RESERVE
220  23.732242    17      MN(ar3)     AR3(ar3)     GIST Data, RESERVE
221  23.737756    16   AR3(inet2)      CN(ar1)     GIST Query
222  23.737764    15   AR3(inet2)      CN(ar1)     GIST Query
223  23.743691    15   AR2(inet2)   AR3(inet2)     GIST Response
```

```
224  23.743700    16   AR2(inet2)   AR3(inet2)     GIST Response
225  23.745032    16   AR3(inet2)   AR2(inet2)     GIST Confirm
226  23.745039    15   AR3(inet2)   AR2(inet2)     GIST Confirm
227  23.745557    16   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE
228  23.745561    15   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE
229  23.752856    13    AR2(inet)      CN(ar1)     GIST Query
230  23.752865     7    AR2(inet)      CN(ar1)     GIST Query
231  23.754184     7    AR1(inet)    AR2(inet)     GIST Response
232  23.754189    13    AR1(inet)    AR2(inet)     GIST Response
233  23.755226    13    AR2(inet)    AR1(inet)     GIST Confirm
234  23.755232     7    AR2(inet)    AR1(inet)     GIST Confirm
235  23.755694    13    AR2(inet)    AR1(inet)     GIST Data, RESERVE
236  23.755698     7    AR2(inet)    AR1(inet)     GIST Data, RESERVE
237  23.756939     7    AR1(inet)     HA(inet)     GIST Data, NOTIFY*
238  23.756944     4    AR1(inet)     HA(inet)     GIST Data, NOTIFY*
239  23.757809     8     AR1(ar1)      CN(ar1)     GIST Query
240  23.757814    10     AR1(ar1)      CN(ar1)     GIST Query
241  23.759482    10      CN(ar1)     AR1(ar1)     GIST Response
242  23.759489     8      CN(ar1)     AR1(ar1)     GIST Response
243  23.759645     4   HA(home:2)     MN(home)     GIST Data, NOTIFY*
244  23.759650    13   HA(home:2)     MN(home)     GIST Data, NOTIFY*
245  23.759926    15   HA(home:2)     MN(home)     GIST Data, NOTIFY*
246  23.759932    16   HA(home:2)     MN(home)     GIST Data, NOTIFY*
247  23.760210    17   HA(home:2)     MN(home)     GIST Data, NOTIFY*
248  23.760216     1   HA(home:2)     MN(home)     GIST Data, NOTIFY*
249  23.760658     8     AR1(ar1)      CN(ar1)     GIST Confirm
250  23.760663    10     AR1(ar1)      CN(ar1)     GIST Confirm
251  23.761174     8     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
252  23.761178    10     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
253  23.763173    10      CN(ar1)     AR1(ar1)     GIST Data, RESPONSE
254  23.763179     8      CN(ar1)     AR1(ar1)     GIST Data, RESPONSE
255  23.764478     7    AR1(inet)    AR2(inet)     GIST Data, RESPONSE
256  23.764485    13    AR1(inet)    AR2(inet)     GIST Data, RESPONSE
257  23.765935    15   AR2(inet2)   AR3(inet2)     GIST Data, RESPONSE
258  23.765944    16   AR2(inet2)   AR3(inet2)     GIST Data, RESPONSE
259  23.768787    17     AR3(ar3)      MN(ar3)     GIST Data, RESPONSE
260  23.768795     1     AR3(ar3)      MN(ar3)     GIST Data, RESPONSE
261  23.770635     1     MN(home)   HA(home:2)     GIST Data, RESERVE(TEAR)*
262  23.770662    17     MN(home)   HA(home:2)     GIST Data, RESERVE(TEAR)*
263  23.770896    16     MN(home)   HA(home:2)     GIST Data, RESERVE(TEAR)*
264  23.770902    15     MN(home)   HA(home:2)     GIST Data, RESERVE(TEAR)*
265  23.771172    13     MN(home)   HA(home:2)     GIST Data, RESERVE(TEAR)*
266  23.771179     4     MN(home)   HA(home:2)     GIST Data, RESERVE(TEAR)*
267  23.772832     4     HA(inet)    AR1(inet)     GIST Data, RESERVE(TEAR)*
268  23.772840     7     HA(inet)    AR1(inet)     GIST Data, RESERVE(TEAR)*
```

### A.1.5 At AR3: Hop-to-Hop Refreshes

```
269  23.855794     1      MN(ar3)     AR3(ar3)     GIST Data, RESERVE
270  23.855804    17      MN(ar3)     AR3(ar3)     GIST Data, RESERVE
271  27.181178    16   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE
272  27.181186    15   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE
273  27.416567    13    AR2(inet)    AR1(inet)     GIST Data, RESERVE
274  27.416579     7    AR2(inet)    AR1(inet)     GIST Data, RESERVE
275  27.502663     8     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
276  27.502670    10     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
277  28.857577     1      MN(ar3)     AR3(ar3)     GIST Data, RESERVE
278  28.857583    17      MN(ar3)     AR3(ar3)     GIST Data, RESERVE
```

```
279  29.104851     8     AR1(ar1)      CN(ar1)     GIST Query
280  29.104859    10     AR1(ar1)      CN(ar1)     GIST Query
281  29.105766    10      CN(ar1)     AR1(ar1)     GIST Response
282  29.105771     8      CN(ar1)     AR1(ar1)     GIST Response
283  29.106521     8     AR1(ar1)      CN(ar1)     GIST Confirm
284  29.106526    10     AR1(ar1)      CN(ar1)     GIST Confirm
285  29.857511     1      MN(ar3)     AR3(ar3)     GIST Data, RESERVE
286  29.857516    17      MN(ar3)     AR3(ar3)     GIST Data, RESERVE
287  31.182990    16   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE
288  31.182997    15   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE
289  32.416355    13    AR2(inet)    AR1(inet)     GIST Data, RESERVE
290  32.416363     7    AR2(inet)    AR1(inet)     GIST Data, RESERVE
291  32.907731    16   AR3(inet2)      CN(ar1)     GIST Query
292  32.907738    15   AR3(inet2)      CN(ar1)     GIST Query
293  32.908838    15   AR2(inet2)   AR3(inet2)     GIST Response
294  32.908843    16   AR2(inet2)   AR3(inet2)     GIST Response
295  32.909773    16   AR3(inet2)   AR2(inet2)     GIST Confirm
296  32.909777    15   AR3(inet2)   AR2(inet2)     GIST Confirm
297  33.503413     8     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
298  33.503420    10     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
299  34.858226     1      MN(ar3)     AR3(ar3)     GIST Data, RESERVE
300  34.858233    17      MN(ar3)     AR3(ar3)     GIST Data, RESERVE
301  37.182174    16   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE
302  37.182182    15   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE
303  37.417825    13    AR2(inet)    AR1(inet)     GIST Data, RESERVE
304  37.417835     7    AR2(inet)    AR1(inet)     GIST Data, RESERVE
305  37.859685     1      MN(ar3)     AR3(ar3)     GIST Data, RESERVE
306  37.859692    17      MN(ar3)     AR3(ar3)     GIST Data, RESERVE
307  38.503569     8     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
308  38.503576    10     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
309  39.814209    13    AR2(inet)      CN(ar1)     GIST Query
310  39.814218     7    AR2(inet)      CN(ar1)     GIST Query
311  39.815113     7    AR1(inet)    AR2(inet)     GIST Response
312  39.815118    13    AR1(inet)    AR2(inet)     GIST Response
313  39.815848    13    AR2(inet)    AR1(inet)     GIST Confirm
314  39.815852     7    AR2(inet)    AR1(inet)     GIST Confirm
315  39.834480     8     AR1(ar1)      CN(ar1)     GIST Query
316  39.834489    10     AR1(ar1)      CN(ar1)     GIST Query
317  39.835647    10      CN(ar1)     AR1(ar1)     GIST Response
318  39.835652     8      CN(ar1)     AR1(ar1)     GIST Response
319  39.836308     8     AR1(ar1)      CN(ar1)     GIST Confirm
320  39.836313    10     AR1(ar1)      CN(ar1)     GIST Confirm
321  39.875436     8     AR1(ar1)      CN(ar1)     GIST Query
322  39.875448    10     AR1(ar1)      CN(ar1)     GIST Query
323  39.876365    10      CN(ar1)     AR1(ar1)     GIST Response
324  39.876370     8      CN(ar1)     AR1(ar1)     GIST Response
325  39.877171     8     AR1(ar1)      CN(ar1)     GIST Confirm
326  39.877175    10     AR1(ar1)      CN(ar1)     GIST Confirm
327  41.183187    16   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE
328  41.183196    15   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE
329  41.534393     1      MN(ar3)      CN(ar1)     GIST Query
330  41.534402    17      MN(ar3)      CN(ar1)     GIST Query
331  41.535462    17     AR3(ar3)      MN(ar3)     GIST Response
332  41.535467     1     AR3(ar3)      MN(ar3)     GIST Response
333  41.536355     1      MN(ar3)     AR3(ar3)     GIST Confirm
334  41.536360    17      MN(ar3)     AR3(ar3)     GIST Confirm
335  42.504126     8     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
336  42.504134    10     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
337  42.859812     1      MN(ar3)     AR3(ar3)     GIST Data, RESERVE
```

```
338  42.859822    17      MN(ar3)     AR3(ar3)     GIST Data, RESERVE
339  44.417723    13    AR2(inet)    AR1(inet)     GIST Data, RESERVE
340  44.417733     7    AR2(inet)    AR1(inet)     GIST Data, RESERVE
341  46.184568    16   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE
342  46.184575    15   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE
343  47.859425     1      MN(ar3)     AR3(ar3)     GIST Data, RESERVE
344  47.859435    17      MN(ar3)     AR3(ar3)     GIST Data, RESERVE
345  48.817323    13    AR2(inet)      CN(ar1)     GIST Query
346  48.817335     7    AR2(inet)      CN(ar1)     GIST Query
347  48.818355     7    AR1(inet)    AR2(inet)     GIST Response
348  48.818362    13    AR1(inet)    AR2(inet)     GIST Response
349  48.819167    13    AR2(inet)    AR1(inet)     GIST Confirm
350  48.819173     7    AR2(inet)    AR1(inet)     GIST Confirm
351  49.505237     8     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
352  49.505247    10     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
353  50.859442     1      MN(ar3)     AR3(ar3)     GIST Data, RESERVE
354  50.859451    17      MN(ar3)     AR3(ar3)     GIST Data, RESERVE
355  51.418648    13    AR2(inet)    AR1(inet)     GIST Data, RESERVE
356  51.418656     7    AR2(inet)    AR1(inet)     GIST Data, RESERVE
357  52.185063    16   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE
358  52.185069    15   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE
359  52.506058     8     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
360  52.506064    10     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
361  55.233612    16   AR3(inet2)      CN(ar1)     GIST Query
362  55.233620    15   AR3(inet2)      CN(ar1)     GIST Query
363  55.234717    15   AR2(inet2)   AR3(inet2)     GIST Response
364  55.234724    16   AR2(inet2)   AR3(inet2)     GIST Response
365  55.235441    16   AR3(inet2)   AR2(inet2)     GIST Confirm
366  55.235446    15   AR3(inet2)   AR2(inet2)     GIST Confirm
367  55.861016     1      MN(ar3)     AR3(ar3)     GIST Data, RESERVE
368  55.861025    17      MN(ar3)     AR3(ar3)     GIST Data, RESERVE
369  56.184588    16   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE
370  56.184599    15   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE
371  56.420297    13    AR2(inet)    AR1(inet)     GIST Data, RESERVE
372  56.420307     7    AR2(inet)    AR1(inet)     GIST Data, RESERVE
373  57.506474     8     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
374  57.506478    10     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
375  59.420254    13    AR2(inet)    AR1(inet)     GIST Data, RESERVE
376  59.420262     7    AR2(inet)    AR1(inet)     GIST Data, RESERVE
```

### A.1.6 Handover to AR1

```
377  60.742141     1      MN(ar1)   HA(home:1)    MIPv6 Binding Update
378  60.742147     8      MN(ar1)   HA(home:1)    MIPv6 Binding Update
379  60.742389     7      MN(ar1)   HA(home:1)    MIPv6 Binding Update
380  60.742395     4      MN(ar1)   HA(home:1)    MIPv6 Binding Update
381  60.743437     4   HA(home:1)      MN(ar1)    MIPv6 Binding Acknowledgement
382  60.743441     7   HA(home:1)      MN(ar1)    MIPv6 Binding Acknowledgement
383  60.749343     8   HA(home:1)      MN(ar1)    MIPv6 Binding Acknowledgement
384  60.749348     1   HA(home:1)      MN(ar1)    MIPv6 Binding Acknowledgement
385  60.751322     1      MN(ar1)      CN(ar1)    MIPv6 Care-of Test Init
386  60.751325    10      MN(ar1)      CN(ar1)    MIPv6 Care-of Test Init
387  60.751756    10      CN(ar1)      MN(ar1)    MIPv6 Care-of Test
388  60.751761     1      CN(ar1)      MN(ar1)    MIPv6 Care-of Test
389  60.752335     1      MN(ar1)      CN(ar1)    MIPv6 Binding Update
390  60.752340    10      MN(ar1)      CN(ar1)    MIPv6 Binding Update
391  60.753206    10      CN(ar1)      MN(ar1)    MIPv6 Binding Acknowledgement
392  60.753212     1      CN(ar1)      MN(ar1)    MIPv6 Binding Acknowledgement
```

```
393  60.754686     1      MN(ar1)      CN(ar1)     GIST Query
    Message routing information object
        Object Header
        MRI type: 0
        Flags: 0x00
        IP version: 6
        Flags: 0x800 (P)
        Source address: MN(ar1)
        Destination address: CN(ar1)
        Source address prefix: 128
        Destination address prefix: 0
        Protocol: UDP (0x11)
394  60.754693    10      MN(ar1)      CN(ar1)     GIST Query
395  60.755788    10      CN(ar1)      MN(ar1)     GIST Response
396  60.755796     1      CN(ar1)      MN(ar1)     GIST Response
397  60.757484     1      MN(ar1)      CN(ar1)     GIST Confirm
398  60.757490    10      MN(ar1)      CN(ar1)     GIST Confirm
399  60.758839     1      MN(ar1)      CN(ar1)     GIST Data, RESERVE
400  60.758849    10      MN(ar1)      CN(ar1)     GIST Data, RESERVE
401  60.760110    10      CN(ar1)     AR1(ar1)     GIST Data, NOTIFY*
402  60.760119     8      CN(ar1)     AR1(ar1)     GIST Data, NOTIFY*
403  60.760644    10      CN(ar1)      MN(ar1)     GIST Data, RESPONSE
404  60.760651     1      CN(ar1)      MN(ar1)     GIST Data, RESPONSE
405  60.761838     7    AR1(inet)    AR2(inet)     GIST Data, NOTIFY*
406  60.761848    13    AR1(inet)    AR2(inet)     GIST Data, NOTIFY*
407  60.763520    15   AR2(inet2)   AR3(inet2)     GIST Data, NOTIFY*
408  60.763532    16   AR2(inet2)   AR3(inet2)     GIST Data, NOTIFY*
409  60.765241    17     AR3(ar3)      MN(ar3)     GIST Data, NOTIFY*
```

### A.1.7 At AR1: Hop-to-Hop Refreshes for old and new path

```
410  60.954115     8     AR1(ar1)      CN(ar1)     GIST Query*
411  60.954129    10     AR1(ar1)      CN(ar1)     GIST Query*
412  60.954918    10      CN(ar1)     AR1(ar1)     GIST Response*
413  60.954927     8      CN(ar1)     AR1(ar1)     GIST Response*
414  60.955926     8     AR1(ar1)      CN(ar1)     GIST Confirm*
415  60.955931    10     AR1(ar1)      CN(ar1)     GIST Confirm*
416  61.185140    16   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE*
417  61.185153    15   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE*
418  61.861044     1      MN(ar1)      CN(ar1)     GIST Data, RESERVE
419  61.861071    10      MN(ar1)      CN(ar1)     GIST Data, RESERVE
420  62.196374    13    AR2(inet)      CN(ar1)     GIST Query*
421  62.196385     7    AR2(inet)      CN(ar1)     GIST Query*
422  62.197437     7    AR1(inet)    AR2(inet)     GIST Response*
423  62.197443    13    AR1(inet)    AR2(inet)     GIST Response*
424  62.198254    13    AR2(inet)    AR1(inet)     GIST Confirm*
425  62.198258     7    AR2(inet)    AR1(inet)     GIST Confirm*
426  62.506613     8     AR1(ar1)      CN(ar1)     GIST Data, RESERVE*
427  62.506621    10     AR1(ar1)      CN(ar1)     GIST Data, RESERVE*
428  63.509084     8     AR1(ar1)      CN(ar1)     GIST Data, RESERVE*
429  63.509101    10     AR1(ar1)      CN(ar1)     GIST Data, RESERVE*
430  64.423049    13    AR2(inet)    AR1(inet)     GIST Data, RESERVE*
431  64.423062     7    AR2(inet)    AR1(inet)     GIST Data, RESERVE*
432  65.186682    16   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE*
433  65.186693    15   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE*
434  66.507951     8     AR1(ar1)      CN(ar1)     GIST Data, RESERVE*
435  66.507961    10     AR1(ar1)      CN(ar1)     GIST Data, RESERVE*
```

```
436  67.862029     1      MN(ar1)      CN(ar1)     GIST Data, RESERVE
437  67.862037    10      MN(ar1)      CN(ar1)     GIST Data, RESERVE
438  69.508343     8     AR1(ar1)      CN(ar1)     GIST Data, RESERVE*
439  69.508351    10     AR1(ar1)      CN(ar1)     GIST Data, RESERVE*
440  70.187042    16   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE*
441  70.187049    15   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE*
442  70.425615    13    AR2(inet)    AR1(inet)     GIST Data, RESERVE*
443  70.425624     7    AR2(inet)    AR1(inet)     GIST Data, RESERVE*
444  71.470774    16   AR3(inet2)      CN(ar1)     GIST Query*
445  71.470782    15   AR3(inet2)      CN(ar1)     GIST Query*
446  71.471912    15   AR2(inet2)   AR3(inet2)     GIST Response*
447  71.471917    16   AR2(inet2)   AR3(inet2)     GIST Response*
448  71.472773    16   AR3(inet2)   AR2(inet2)     GIST Confirm*
449  71.472777    15   AR3(inet2)   AR2(inet2)     GIST Confirm*
```

### A.1.8 At AR1: Old Path Teardown

```
450  71.544132    16   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE(TEAR)*
451  71.544139    15   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE(TEAR)*
452  71.545862    13    AR2(inet)    AR1(inet)     GIST Data, RESERVE(TEAR)*
453  71.545869     7    AR2(inet)    AR1(inet)     GIST Data, RESERVE(TEAR)*
454  71.547833     8     AR1(ar1)      CN(ar1)     GIST Data, RESERVE(TEAR)*
455  71.547837    10     AR1(ar1)      CN(ar1)     GIST Data, RESERVE(TEAR)*
```

### A.1.9 At AR1: Hop-to-Hop Refreshes

```
456  72.862850     1      MN(ar1)      CN(ar1)     GIST Data, RESERVE
457  72.862856    10      MN(ar1)      CN(ar1)     GIST Data, RESERVE
458  73.072042     1      MN(ar1)      CN(ar1)     GIST Query
459  73.072055    10      MN(ar1)      CN(ar1)     GIST Query
460  73.072807    10      CN(ar1)      MN(ar1)     GIST Response
461  73.072814     1      CN(ar1)      MN(ar1)     GIST Response
462  73.073538     1      MN(ar1)      CN(ar1)     GIST Confirm
463  73.073543    10      MN(ar1)      CN(ar1)     GIST Confirm
464  78.863045     1      MN(ar1)      CN(ar1)     GIST Data, RESERVE
465  78.863057    10      MN(ar1)      CN(ar1)     GIST Data, RESERVE
466  83.864465     1      MN(ar1)      CN(ar1)     GIST Data, RESERVE
467  83.864475    10      MN(ar1)      CN(ar1)     GIST Data, RESERVE
468  88.866291     1      MN(ar1)      CN(ar1)     GIST Data, RESERVE
469  88.866302    10      MN(ar1)      CN(ar1)     GIST Data, RESERVE
470  90.498571     1      MN(ar1)      CN(ar1)     GIST Query
471  90.498581    10      MN(ar1)      CN(ar1)     GIST Query
472  90.499494    10      CN(ar1)      MN(ar1)     GIST Response
473  90.499500     1      CN(ar1)      MN(ar1)     GIST Response
474  90.500249     1      MN(ar1)      CN(ar1)     GIST Confirm
475  90.500255    10      MN(ar1)      CN(ar1)     GIST Confirm
476  93.866678     1      MN(ar1)      CN(ar1)     GIST Data, RESERVE
477  93.866684    10      MN(ar1)      CN(ar1)     GIST Data, RESERVE
478  94.865930     1      MN(ar1)      CN(ar1)     GIST Data, RESERVE
479  94.865937    10      MN(ar1)      CN(ar1)     GIST Data, RESERVE
480  96.865464     1      MN(ar1)      CN(ar1)     GIST Data, RESERVE
481  96.865472    10      MN(ar1)      CN(ar1)     GIST Data, RESERVE
482 101.866348     1      MN(ar1)      CN(ar1)     GIST Data, RESERVE
483 101.866357    10      MN(ar1)      CN(ar1)     GIST Data, RESERVE
484 103.560708     1      MN(ar1)      CN(ar1)     GIST Query
485 103.560716    10      MN(ar1)      CN(ar1)     GIST Query
486 103.561597    10      CN(ar1)      MN(ar1)     GIST Response
```

```
487 103.561603     1      CN(ar1)      MN(ar1)     GIST Response
488 103.562705     1      MN(ar1)      CN(ar1)     GIST Confirm
489 103.562710    10      MN(ar1)      CN(ar1)     GIST Confirm
490 105.867226     1      MN(ar1)      CN(ar1)     GIST Data, RESERVE
491 105.867237    10      MN(ar1)      CN(ar1)     GIST Data, RESERVE
492 108.867763     1      MN(ar1)      CN(ar1)     GIST Data, RESERVE
493 108.867771    10      MN(ar1)      CN(ar1)     GIST Data, RESERVE
494 110.867718     1      MN(ar1)      CN(ar1)     GIST Data, RESERVE
495 110.867726    10      MN(ar1)      CN(ar1)     GIST Data, RESERVE
496 113.612566     1      MN(ar1)      CN(ar1)     GIST Query
497 113.612577    10      MN(ar1)      CN(ar1)     GIST Query
498 113.613405    10      CN(ar1)      MN(ar1)     GIST Response
499 113.613410     1      CN(ar1)      MN(ar1)     GIST Response
500 113.614862     1      MN(ar1)      CN(ar1)     GIST Confirm
501 113.614867    10      MN(ar1)      CN(ar1)     GIST Confirm
502 116.868874     1      MN(ar1)      CN(ar1)     GIST Data, RESERVE
503 116.868883    10      MN(ar1)      CN(ar1)     GIST Data, RESERVE
```

### A.1.10 Handover to AR3

```
504 120.886695     1      MN(ar3)   HA(home:1)    MIPv6 Binding Update
505 120.886702    17      MN(ar3)   HA(home:1)    MIPv6 Binding Update
506 120.886921    16      MN(ar3)   HA(home:1)    MIPv6 Binding Update
507 120.886927    15      MN(ar3)   HA(home:1)    MIPv6 Binding Update
508 120.887230    13      MN(ar3)   HA(home:1)    MIPv6 Binding Update
509 120.887236     4      MN(ar3)   HA(home:1)    MIPv6 Binding Update
510 120.887307     1      MN(ar3)      CN(ar1)    MIPv6 Care-of Test Init
511 120.887310    17      MN(ar3)      CN(ar1)    MIPv6 Care-of Test Init
512 120.887588    16      MN(ar3)      CN(ar1)    MIPv6 Care-of Test Init
513 120.887592    15      MN(ar3)      CN(ar1)    MIPv6 Care-of Test Init
514 120.887826    13      MN(ar3)      CN(ar1)    MIPv6 Care-of Test Init
515 120.887831     7      MN(ar3)      CN(ar1)    MIPv6 Care-of Test Init
516 120.888103     8      MN(ar3)      CN(ar1)    MIPv6 Care-of Test Init
517 120.888108    10      MN(ar3)      CN(ar1)    MIPv6 Care-of Test Init
518 120.888779     4   HA(home:1)      MN(ar3)    MIPv6 Binding Acknowledgement
519 120.888784    13   HA(home:1)      MN(ar3)    MIPv6 Binding Acknowledgement
520 120.889995    15   HA(home:1)      MN(ar3)    MIPv6 Binding Acknowledgement
521 120.890000    16   HA(home:1)      MN(ar3)    MIPv6 Binding Acknowledgement
522 120.891269    10      CN(ar1)      MN(ar3)    MIPv6 Care-of Test
523 120.891275     8      CN(ar1)      MN(ar3)    MIPv6 Care-of Test
524 120.891525     7      CN(ar1)      MN(ar3)    MIPv6 Care-of Test
525 120.891530    13      CN(ar1)      MN(ar3)    MIPv6 Care-of Test
526 120.891824    15      CN(ar1)      MN(ar3)    MIPv6 Care-of Test
527 120.891828    16      CN(ar1)      MN(ar3)    MIPv6 Care-of Test
528 120.895566    17   HA(home:1)      MN(ar3)    MIPv6 Binding Acknowledgement
529 120.895570     1   HA(home:1)      MN(ar3)    MIPv6 Binding Acknowledgement
530 120.895600    17      CN(ar1)      MN(ar3)    MIPv6 Care-of Test
531 120.895604     1      CN(ar1)      MN(ar3)    MIPv6 Care-of Test
532 120.896717     1      MN(ar3)      CN(ar1)    MIPv6 Binding Update
533 120.896721    17      MN(ar3)      CN(ar1)    MIPv6 Binding Update
534 120.896913    16      MN(ar3)      CN(ar1)    MIPv6 Binding Update
535 120.896918    15      MN(ar3)      CN(ar1)    MIPv6 Binding Update
536 120.897295    13      MN(ar3)      CN(ar1)    MIPv6 Binding Update
537 120.897299     7      MN(ar3)      CN(ar1)    MIPv6 Binding Update
538 120.897531     8      MN(ar3)      CN(ar1)    MIPv6 Binding Update
539 120.897535    10      MN(ar3)      CN(ar1)    MIPv6 Binding Update
540 120.902291    10      CN(ar1)      MN(ar3)    MIPv6 Binding Acknowledgement
541 120.902296     8      CN(ar1)      MN(ar3)    MIPv6 Binding Acknowledgement
```

```
542 120.902569     7      CN(ar1)      MN(ar3)    MIPv6 Binding Acknowledgement
543 120.902574    13      CN(ar1)      MN(ar3)    MIPv6 Binding Acknowledgement
544 120.902790    15      CN(ar1)      MN(ar3)    MIPv6 Binding Acknowledgement
545 120.902794    16      CN(ar1)      MN(ar3)    MIPv6 Binding Acknowledgement
546 120.902994    17      CN(ar1)      MN(ar3)    MIPv6 Binding Acknowledgement
547 120.902998     1      CN(ar1)      MN(ar3)    MIPv6 Binding Acknowledgement

548 120.904962     1      MN(ar3)      CN(ar1)     GIST Query
    Message routing information object
        Object Header
        MRI type: 0
        Flags: 0x00
        IP version: 6
        Flags: 0x800 (P)
        Source address: MN(ar3)
        Destination address: CN(ar1)
        Source address prefix: 128
        Destination address prefix: 0
        Protocol: UDP (0x11)
549 120.904967    17      MN(ar3)      CN(ar1)     GIST Query
550 120.912620    17     AR3(ar3)      MN(ar3)     GIST Response
551 120.912629     1     AR3(ar3)      MN(ar3)     GIST Response
552 120.914748     1      MN(ar3)     AR3(ar3)     GIST Confirm
553 120.914752    17      MN(ar3)     AR3(ar3)     GIST Confirm
554 120.914755     1      MN(ar3)     AR3(ar3)     GIST Data, RESERVE
555 120.914759    17      MN(ar3)     AR3(ar3)     GIST Data, RESERVE
556 120.927531    16   AR3(inet2)      CN(ar1)     GIST Query
557 120.927539    15   AR3(inet2)      CN(ar1)     GIST Query
558 120.929247    15   AR2(inet2)   AR3(inet2)     GIST Response
559 120.929251    16   AR2(inet2)   AR3(inet2)     GIST Response
560 120.930160    16   AR3(inet2)   AR2(inet2)     GIST Confirm
561 120.930165    15   AR3(inet2)   AR2(inet2)     GIST Confirm
562 120.930673    16   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE
563 120.930677    15   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE
564 120.932780    13    AR2(inet)      CN(ar1)     GIST Query
565 120.932789     7    AR2(inet)      CN(ar1)     GIST Query
566 120.934351     7    AR1(inet)    AR2(inet)     GIST Response
567 120.934357    13    AR1(inet)    AR2(inet)     GIST Response
568 120.935059    13    AR2(inet)    AR1(inet)     GIST Confirm
569 120.935064     7    AR2(inet)    AR1(inet)     GIST Confirm
570 120.935567    13    AR2(inet)    AR1(inet)     GIST Data, RESERVE
571 120.935572     7    AR2(inet)    AR1(inet)     GIST Data, RESERVE
572 120.937635     8     AR1(ar1)      CN(ar1)     GIST Query
573 120.937643    10     AR1(ar1)      CN(ar1)     GIST Query
574 120.944490    10      CN(ar1)     AR1(ar1)     GIST Response
575 120.944495     8      CN(ar1)     AR1(ar1)     GIST Response
576 120.945340     8     AR1(ar1)      CN(ar1)     GIST Confirm
577 120.945345    10     AR1(ar1)      CN(ar1)     GIST Confirm
578 120.945950     8     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
579 120.945954    10     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
580 120.948065    10      CN(ar1)      MN(ar1)     GIST Data, NOTIFY*
581 120.948072     8      CN(ar1)      MN(ar1)     GIST Data, NOTIFY*
582 120.948719    10      CN(ar1)     AR1(ar1)     GIST Data, RESPONSE
583 120.948723     8      CN(ar1)     AR1(ar1)     GIST Data, RESPONSE
584 120.950655     7    AR1(inet)    AR2(inet)     GIST Data, RESPONSE
585 120.950662    13    AR1(inet)    AR2(inet)     GIST Data, RESPONSE
586 120.951963    15   AR2(inet2)   AR3(inet2)     GIST Data, RESPONSE
587 120.951970    16   AR2(inet2)   AR3(inet2)     GIST Data, RESPONSE
```

```
588 120.953182    17     AR3(ar3)      MN(ar3)     GIST Data, RESPONSE
589 120.953188     1     AR3(ar3)      MN(ar3)     GIST Data, RESPONSE
```

### A.1.11 At AR3: Hop-to-Hop Refreshes

```
590 122.193388    16   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE
591 122.193398    15   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE
592 124.514812     8     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
593 124.514829    10     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
594 124.869140     1      MN(ar3)     AR3(ar3)     GIST Data, RESERVE
595 124.869146    17      MN(ar3)     AR3(ar3)     GIST Data, RESERVE
596 126.429253    13    AR2(inet)    AR1(inet)     GIST Data, RESERVE
597 126.429264     7    AR2(inet)    AR1(inet)     GIST Data, RESERVE
598 126.515701     8     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
599 126.515710    10     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
600 128.195220    16   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE
601 128.195231    15   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE
602 129.870072     1      MN(ar3)     AR3(ar3)     GIST Data, RESERVE
603 129.870083    17      MN(ar3)     AR3(ar3)     GIST Data, RESERVE
604 130.850796    16   AR3(inet2)      CN(ar1)     GIST Query
605 130.850804    15   AR3(inet2)      CN(ar1)     GIST Query
606 130.851802    15   AR2(inet2)   AR3(inet2)     GIST Response
607 130.851807    16   AR2(inet2)   AR3(inet2)     GIST Response
608 130.852435    16   AR3(inet2)   AR2(inet2)     GIST Confirm
609 130.852440    15   AR3(inet2)   AR2(inet2)     GIST Confirm
610 132.195752    16   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE
611 132.195760    15   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE
612 132.430778    13    AR2(inet)    AR1(inet)     GIST Data, RESERVE
613 132.430787     7    AR2(inet)    AR1(inet)     GIST Data, RESERVE
614 133.516253     8     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
615 133.516260    10     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
616 133.870628     1      MN(ar3)     AR3(ar3)     GIST Data, RESERVE
617 133.870635    17      MN(ar3)     AR3(ar3)     GIST Data, RESERVE
618 134.027269     1      MN(ar3)      CN(ar1)     GIST Query
619 134.027278    17      MN(ar3)      CN(ar1)     GIST Query
620 134.027921    17     AR3(ar3)      MN(ar3)     GIST Response
621 134.027926     1     AR3(ar3)      MN(ar3)     GIST Response
622 134.029268     1      MN(ar3)     AR3(ar3)     GIST Confirm
623 134.029274    17      MN(ar3)     AR3(ar3)     GIST Confirm
624 136.195834    16   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE
625 136.195845    15   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE
626 136.430623    13    AR2(inet)    AR1(inet)     GIST Data, RESERVE
627 136.430632     7    AR2(inet)    AR1(inet)     GIST Data, RESERVE
628 136.518941     8     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
629 136.518951    10     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
630 137.195104    16   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE
631 137.195114    15   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE
632 137.872808     1      MN(ar3)     AR3(ar3)     GIST Data, RESERVE
633 137.872817    17      MN(ar3)     AR3(ar3)     GIST Data, RESERVE
634 139.451494    16   AR3(inet2)      CN(ar1)     GIST Query
635 139.451502    15   AR3(inet2)      CN(ar1)     GIST Query
636 139.452416    15   AR2(inet2)   AR3(inet2)     GIST Response
637 139.452421    16   AR2(inet2)   AR3(inet2)     GIST Response
638 139.454590    16   AR3(inet2)   AR2(inet2)     GIST Confirm
639 139.454596    15   AR3(inet2)   AR2(inet2)     GIST Confirm
640 140.196006    16   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE
641 140.196017    15   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE
642 140.517827     8     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
```

```
643 140.517835    10     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
644 140.615136     8     AR1(ar1)      CN(ar1)     GIST Query
645 140.615142    10     AR1(ar1)      CN(ar1)     GIST Query
646 140.616158    10      CN(ar1)     AR1(ar1)     GIST Response
647 140.616162     8      CN(ar1)     AR1(ar1)     GIST Response
648 140.616824     8     AR1(ar1)      CN(ar1)     GIST Confirm
649 140.616828    10     AR1(ar1)      CN(ar1)     GIST Confirm
650 140.871942     1      MN(ar3)     AR3(ar3)     GIST Data, RESERVE
651 140.871950    17      MN(ar3)     AR3(ar3)     GIST Data, RESERVE
652 141.430734    13    AR2(inet)    AR1(inet)     GIST Data, RESERVE
653 141.430743     7    AR2(inet)    AR1(inet)     GIST Data, RESERVE
654 143.253023    13    AR2(inet)      CN(ar1)     GIST Query
655 143.253034     7    AR2(inet)      CN(ar1)     GIST Query
656 143.254516     7    AR1(inet)    AR2(inet)     GIST Response
657 143.254523    13    AR1(inet)    AR2(inet)     GIST Response
658 143.255250    13    AR2(inet)    AR1(inet)     GIST Confirm
659 143.255255     7    AR2(inet)    AR1(inet)     GIST Confirm
660 143.518805     8     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
661 143.518810    10     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
662 144.196282    16   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE
663 144.196295    15   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE
664 144.872251     1      MN(ar3)     AR3(ar3)     GIST Data, RESERVE
665 144.872263    17      MN(ar3)     AR3(ar3)     GIST Data, RESERVE
666 146.517653     8     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
667 146.517663    10     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
668 147.196696    16   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE
669 147.196705    15   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE
670 147.432894    13    AR2(inet)    AR1(inet)     GIST Data, RESERVE
671 147.432905     7    AR2(inet)    AR1(inet)     GIST Data, RESERVE
672 148.872871     1      MN(ar3)     AR3(ar3)     GIST Data, RESERVE
673 148.872879     5      MN(ar3)     AR3(ar3)     GIST Data, RESERVE
```

### A.1.12 Handover Back Home

```
674 149.313293     1     MN(home)   HA(home:1)    MIPv6 Binding Update
675 149.313299     5     MN(home)   HA(home:1)    MIPv6 Binding Update
676 149.545296     5   HA(home:1)     MN(home)    MIPv6 Binding Acknowledgement
677 149.545301     1   HA(home:1)     MN(home)    MIPv6 Binding Acknowledgement
678 149.546457     1     MN(home)      CN(ar1)    MIPv6 Binding Update
679 149.546463     5     MN(home)      CN(ar1)    MIPv6 Binding Update
680 149.546663     4     MN(home)      CN(ar1)    MIPv6 Binding Update
681 149.546670     7     MN(home)      CN(ar1)    MIPv6 Binding Update
682 149.547009     8     MN(home)      CN(ar1)    MIPv6 Binding Update
683 149.547015    10     MN(home)      CN(ar1)    MIPv6 Binding Update
684 149.547683    10      CN(ar1)     MN(home)    MIPv6 Binding Acknowledgement
685 149.547688     8      CN(ar1)     MN(home)    MIPv6 Binding Acknowledgement
686 149.547857     7      CN(ar1)     MN(home)    MIPv6 Binding Acknowledgement
687 149.547861     4      CN(ar1)     MN(home)    MIPv6 Binding Acknowledgement
688 149.548143     5      CN(ar1)     MN(home)    MIPv6 Binding Acknowledgement
689 149.548148     1      CN(ar1)     MN(home)    MIPv6 Binding Acknowledgement

690 149.550009     1     MN(home)      CN(ar1)     GIST Query
    Message routing information object
        Object Header
        MRI type: 0
        Flags: 0x00
        IP version: 6
        Flags: 0x800 (P)
```

```
        Source address: MN(home)
        Destination address: CN(ar1)
        Source address prefix: 128
        Destination address prefix: 0
        Protocol: UDP (0x11)
691 149.550014     5     MN(home)      CN(ar1)     GIST Query
692 149.551529     5   HA(home:2)     MN(home)     GIST Response
693 149.551534     1   HA(home:2)     MN(home)     GIST Response
694 149.556064     1     MN(home)   HA(home:2)     GIST Confirm
695 149.556068     5     MN(home)   HA(home:2)     GIST Confirm
696 149.562307     1     MN(home)   HA(home:2)     GIST Data, RESERVE
697 149.562315     5     MN(home)   HA(home:2)     GIST Data, RESERVE
698 149.565408     4     HA(inet)      CN(ar1)     GIST Query
699 149.565417     7     HA(inet)      CN(ar1)     GIST Query
700 149.566814     7    AR1(inet)     HA(inet)     GIST Response
701 149.566819     4    AR1(inet)     HA(inet)     GIST Response
702 149.568494     4     HA(inet)    AR1(inet)     GIST Confirm
703 149.568501     7     HA(inet)    AR1(inet)     GIST Confirm
704 149.569613     4     HA(inet)    AR1(inet)     GIST Data, RESERVE
705 149.569618     7     HA(inet)    AR1(inet)     GIST Data, RESERVE
706 149.571072     7    AR1(inet)    AR2(inet)     GIST Data, NOTIFY*
707 149.571080    13    AR1(inet)    AR2(inet)     GIST Data, NOTIFY*
708 149.571881     8     AR1(ar1)      CN(ar1)     GIST Query
709 149.571887    10     AR1(ar1)      CN(ar1)     GIST Query
710 149.573437    15   AR2(inet2)   AR3(inet2)     GIST Data, NOTIFY*
711 149.573444    16   AR2(inet2)   AR3(inet2)     GIST Data, NOTIFY*
712 149.575027    17     AR3(ar3)      MN(ar3)     GIST Data, NOTIFY*
713 149.576143    10      CN(ar1)     AR1(ar1)     GIST Response
714 149.576148     8      CN(ar1)     AR1(ar1)     GIST Response
715 149.577004     8     AR1(ar1)      CN(ar1)     GIST Confirm
716 149.577009    10     AR1(ar1)      CN(ar1)     GIST Confirm
717 149.577496     8     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
718 149.577501    10     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
719 149.579884    10      CN(ar1)     AR1(ar1)     GIST Data, RESPONSE
720 149.579888     8      CN(ar1)     AR1(ar1)     GIST Data, RESPONSE
721 149.581025     7    AR1(inet)     HA(inet)     GIST Data, RESPONSE
722 149.581030     4    AR1(inet)     HA(inet)     GIST Data, RESPONSE
723 149.582750     5   HA(home:2)     MN(home)     GIST Data, RESPONSE
724 149.582755     1   HA(home:2)     MN(home)     GIST Data, RESPONSE
```

### A.1.13 At Home: Hop-to-Hop Refreshes for old and new path

```
725 150.431674    13    AR2(inet)    AR1(inet)     GIST Data, RESERVE*
726 150.431683     7    AR2(inet)    AR1(inet)     GIST Data, RESERVE*
727 150.518274     8     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
728 150.518288    10     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
729 150.926856     8     AR1(ar1)      CN(ar1)     GIST Query
730 150.926866    10     AR1(ar1)      CN(ar1)     GIST Query
731 150.927714    10      CN(ar1)     AR1(ar1)     GIST Response
732 150.927720     8      CN(ar1)     AR1(ar1)     GIST Response
733 150.928554     8     AR1(ar1)      CN(ar1)     GIST Confirm
734 150.928559    10     AR1(ar1)      CN(ar1)     GIST Confirm
735 151.397536    13    AR2(inet)      CN(ar1)     GIST Query*
736 151.397548     7    AR2(inet)      CN(ar1)     GIST Query*
737 151.398517     7    AR1(inet)    AR2(inet)     GIST Response*
738 151.398522    13    AR1(inet)    AR2(inet)     GIST Response*
739 151.399105    13    AR2(inet)    AR1(inet)     GIST Confirm*
```

```
740 151.399109     7    AR2(inet)    AR1(inet)     GIST Confirm*
741 152.935746     4     HA(inet)    AR1(inet)     GIST Data, RESERVE
742 152.935755     7     HA(inet)    AR1(inet)     GIST Data, RESERVE
743 153.197688    16   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE*
744 153.197700    15   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE*
745 153.873761     1     MN(home)   HA(home:2)     GIST Data, RESERVE
746 153.873770     5     MN(home)   HA(home:2)     GIST Data, RESERVE
747 154.520418     8     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
748 154.520426    10     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
749 156.432765    13    AR2(inet)    AR1(inet)     GIST Data, RESERVE*
750 156.432777     7    AR2(inet)    AR1(inet)     GIST Data, RESERVE*
751 156.874249     1     MN(home)   HA(home:2)     GIST Data, RESERVE
752 156.874266     5     MN(home)   HA(home:2)     GIST Data, RESERVE
753 157.609204    16   AR3(inet2)      CN(ar1)     GIST Query*
754 157.609213    15   AR3(inet2)      CN(ar1)     GIST Query*
755 157.611185    15   AR2(inet2)   AR3(inet2)     GIST Response*
756 157.611191    16   AR2(inet2)   AR3(inet2)     GIST Response*
757 157.612519    16   AR3(inet2)   AR2(inet2)     GIST Confirm*
758 157.612523    15   AR3(inet2)   AR2(inet2)     GIST Confirm*
759 157.937328     4     HA(inet)    AR1(inet)     GIST Data, RESERVE
760 157.937339     7     HA(inet)    AR1(inet)     GIST Data, RESERVE
761 158.198625    16   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE*
762 158.198633    15   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE*
763 158.480745     4     HA(inet)      CN(ar1)     GIST Query
764 158.480755     7     HA(inet)      CN(ar1)     GIST Query
765 158.481836     7    AR1(inet)     HA(inet)     GIST Response
766 158.481841     4    AR1(inet)     HA(inet)     GIST Response
767 158.483784     4     HA(inet)    AR1(inet)     GIST Confirm
768 158.483790     7     HA(inet)    AR1(inet)     GIST Confirm
769 158.993960     8     AR1(ar1)      CN(ar1)     GIST Query
770 158.993965    10     AR1(ar1)      CN(ar1)     GIST Query
771 158.994809    10      CN(ar1)     AR1(ar1)     GIST Response
772 158.994815     8      CN(ar1)     AR1(ar1)     GIST Response
773 158.995529     8     AR1(ar1)      CN(ar1)     GIST Confirm
774 158.995534    10     AR1(ar1)      CN(ar1)     GIST Confirm
775 159.937120     4     HA(inet)    AR1(inet)     GIST Data, RESERVE
776 159.937128     7     HA(inet)    AR1(inet)     GIST Data, RESERVE
777 160.433218    13    AR2(inet)    AR1(inet)     GIST Data, RESERVE
778 160.433227     7    AR2(inet)    AR1(inet)     GIST Data, RESERVE
779 160.505155     8     AR1(ar1)      CN(ar1)     GIST Query
780 160.505164    10     AR1(ar1)      CN(ar1)     GIST Query
781 160.506073    10      CN(ar1)     AR1(ar1)     GIST Response
782 160.506078     8      CN(ar1)     AR1(ar1)     GIST Response
783 160.506789     8     AR1(ar1)      CN(ar1)     GIST Confirm
784 160.506795    10     AR1(ar1)      CN(ar1)     GIST Confirm
785 160.876551     1     MN(home)   HA(home:2)     GIST Data, RESERVE
786 160.876556     5     MN(home)   HA(home:2)     GIST Data, RESERVE
787 161.519684     8     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
788 161.519690    10     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
789 163.875027     1     MN(home)   HA(home:2)     GIST Data, RESERVE
790 163.875035     5     MN(home)   HA(home:2)     GIST Data, RESERVE
```

### A.1.14 At Home: Old Path Teardown

```
791 164.036339    16   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE(TEAR)*
792 164.036347    15   AR3(inet2)   AR2(inet2)     GIST Data, RESERVE(TEAR)*
793 164.037754    13    AR2(inet)    AR1(inet)     GIST Data, RESERVE(TEAR)*
794 164.037762     7    AR2(inet)    AR1(inet)     GIST Data, RESERVE(TEAR)*
```

### A.1.15 At Home: Hop-To-Hop Refreshes

```
795 165.937370     4     HA(inet)    AR1(inet)     GIST Data, RESERVE
796 165.937379     7     HA(inet)    AR1(inet)     GIST Data, RESERVE
797 166.521201     8     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
798 166.521207    10     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
799 166.875146     1     MN(home)   HA(home:2)     GIST Data, RESERVE
800 166.875154     5     MN(home)   HA(home:2)     GIST Data, RESERVE
801 168.875961     1     MN(home)   HA(home:2)     GIST Data, RESERVE
802 168.875970     5     MN(home)   HA(home:2)     GIST Data, RESERVE
803 168.917834     1     MN(home)      CN(ar1)     GIST Query
804 168.917839     5     MN(home)      CN(ar1)     GIST Query
805 168.918854     5   HA(home:2)     MN(home)     GIST Response
806 168.918858     1   HA(home:2)     MN(home)     GIST Response
807 168.920841     1     MN(home)   HA(home:2)     GIST Confirm
808 168.920847     5     MN(home)   HA(home:2)     GIST Confirm
809 170.279245     4     HA(inet)      CN(ar1)     GIST Query
810 170.279256     7     HA(inet)      CN(ar1)     GIST Query
811 170.280285     7    AR1(inet)     HA(inet)     GIST Response
812 170.280311     4    AR1(inet)     HA(inet)     GIST Response
813 170.281300     4     HA(inet)    AR1(inet)     GIST Confirm
814 170.281305     7     HA(inet)    AR1(inet)     GIST Confirm
815 170.521784     8     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
816 170.521789    10     AR1(ar1)      CN(ar1)     GIST Data, RESERVE
817 170.903694     8     AR1(ar1)      CN(ar1)     GIST Query
818 170.903704    10     AR1(ar1)      CN(ar1)     GIST Query
819 170.904371    10      CN(ar1)     AR1(ar1)     GIST Response
820 170.904377     8      CN(ar1)     AR1(ar1)     GIST Response
821 170.905391     8     AR1(ar1)      CN(ar1)     GIST Confirm
822 170.905397    10     AR1(ar1)      CN(ar1)     GIST Confirm
823 170.939856     4     HA(inet)    AR1(inet)     GIST Data, RESERVE
824 170.939866     7     HA(inet)    AR1(inet)     GIST Data, RESERVE
825 171.875745     1     MN(home)   HA(home:2)     GIST Data, RESERVE
826 171.875756     5     MN(home)   HA(home:2)     GIST Data, RESERVE
```

## A.2 CN Sender in Sender-Initiated Mode

```
No.       Time  VLAN       Source  Destination Protocol Info
  1   0.000000     1      MN(ar1)   HA(home:1)    MIPv6 Binding Update
  2   0.000006     8      MN(ar1)   HA(home:1)    MIPv6 Binding Update
  3   0.000240     7      MN(ar1)   HA(home:1)    MIPv6 Binding Update
  4   0.000246     4      MN(ar1)   HA(home:1)    MIPv6 Binding Update
  5   0.007467     4   HA(home:1)      MN(ar1)    MIPv6 Binding Acknowledgement
  6   0.007476     7   HA(home:1)      MN(ar1)    MIPv6 Binding Acknowledgement
  7   0.008985     1      MN(ar1)      CN(ar1)    MIPv6 Care-of Test Init
  8   0.008990    10      MN(ar1)      CN(ar1)    MIPv6 Care-of Test Init
  9   0.009368    10      CN(ar1)      MN(ar1)    MIPv6 Care-of Test
 10   0.009372     1      CN(ar1)      MN(ar1)    MIPv6 Care-of Test
 11   0.009928     1      MN(ar1)      CN(ar1)    MIPv6 Binding Update
 12   0.009932    10      MN(ar1)      CN(ar1)    MIPv6 Binding Update
 13   0.011996     8   HA(home:1)      MN(ar1)    MIPv6 Binding Acknowledgement
 14   0.012001     1   HA(home:1)      MN(ar1)    MIPv6 Binding Acknowledgement
 15   0.013069    10      CN(ar1)      MN(ar1)    MIPv6 Binding Acknowledgement
 16   0.013073     1      CN(ar1)      MN(ar1)    MIPv6 Binding Acknowledgement
 17   0.014397    10      CN(ar1)      MN(ar1)     GIST Query
    Message routing information object
        Object Header
        MRI type: 0
```

```
       Flags: 0x00
       IP version: 6
       Flags: 0x800 (P)
       Source address: CN(ar1)
       Destination address: MN(ar1)
       Source address prefix: 0
       Destination address prefix: 128
       Protocol: UDP (0x11)
18   0.014402     1      CN(ar1)      MN(ar1)     GIST Query
19   0.015725     1      MN(ar1)      CN(ar1)     GIST Response
20   0.015730    10      MN(ar1)      CN(ar1)     GIST Response
21   0.016280    10      CN(ar1)      MN(ar1)     GIST Confirm
22   0.016284     1      CN(ar1)      MN(ar1)     GIST Confirm
23   0.016774    10      CN(ar1)      MN(ar1)     GIST Data, RESERVE
24   0.016778     1      CN(ar1)      MN(ar1)     GIST Data, RESERVE
25   0.020173     1      MN(ar1)      CN(ar1)     GIST Data, RESPONSE
26   0.020178    10      MN(ar1)      CN(ar1)     GIST Data, RESPONSE
27   0.022317    10      CN(ar1)     AR1(ar1)     GIST Data, RESERVE(TEAR)*
   Message routing information object
       Object Header
       MRI type: 0
       Flags: 0x00
       IP version: 6
       Flags: 0x800 (P)
       Source address: CN(ar1)
       Destination address: MN(ar3)
       Source address prefix: 0
       Destination address prefix: 128
       Protocol: UDP (0x11)
28   0.022321     8      CN(ar1)     AR1(ar1)     GIST Data, RESERVE(TEAR)*
29   0.023921     7    AR1(inet)    AR2(inet)     GIST Data, RESERVE(TEAR)*
30   0.023927    13    AR1(inet)    AR2(inet)     GIST Data, RESERVE(TEAR)*
31   0.025681    15   AR2(inet2)   AR3(inet2)     GIST Data, RESERVE(TEAR)*
32   0.025687    16   AR2(inet2)   AR3(inet2)     GIST Data, RESERVE(TEAR)*
```

# B. Setup and Tear Down Delay Benchmarks

The following are our raw benchmark numbers which are discussed in Chapter 5.3. First the raw numbers are given, including numbers we excluded as obvious outliers—marked as such. After the raw numbers we provide a histogram of the numbers including average and median, as well as the 95% confidence interval around the average.

## B.1 Benchmarks Without Addition Delay

### B.1.1 MN Sender, Sender-Initiated, No Delay

Setup Delay while at AR1:

```
0.012521, 0.009143, 0.010928, 0.013858, 0.012173, 0.012331, 0.015062, 0.010738,
0.011922, 0.017683, 0.010704, 0.013266, 0.009454, 0.012740, 0.007646, 0.008531,
0.013782, 0.012146, 0.008302, 0.014122, 0.012801, 0.013521, 0.013790, 0.010516,
0.011789, 0.013931, 0.011730, 0.014942, 0.013180, 0.007619, 0.009976, 0.011400,
0.010292, 0.015530, 0.008344, 0.013945, 0.017290, 0.011503, 0.007794, 0.009785,
0.017947, 0.009425, 0.012117, 0.011475, 0.008066, 0.008284, 0.009512, 0.008195,
0.018135, 0.008672
```

Setup Delay while at AR2:

```
0.025336, 0.026539, 0.030141, 0.031318, 0.032964, 0.021357, 0.028135, 0.022956,
0.030324, 0.022137, 0.023677, 0.027229, 0.023376, 0.024132, 0.028300, 0.029050,
0.019873, 0.025698, 0.021219, 0.024512, 0.029313, 0.039701, 0.022520, 0.039862,
0.027717, 0.028368, 0.026117, 0.021549, 0.024430, 0.030177, 0.024320, 0.021799,
0.030035, 0.020860, 0.027445, 0.041659, 0.032121, 0.028745, 0.032162, 0.025273,
0.032387, 0.024038, 0.027651, 0.028118, 0.025764, 0.020314, 0.023885, 0.032253,
0.023943, 0.029257
```

Setup Delay while at AR3:

```
0.038579, 0.036650, 0.039657, 0.034525, 0.037392, 0.030402, 0.037518, 0.039047,
0.041655, 0.031069, 0.034970, 0.035416, 0.044039, 0.038308, 0.044912, 0.032244,
0.031085, 0.039961, 0.041250, 0.031753, 0.047994, 0.045269, 0.048476, 0.032436,
0.032346, 0.032701, 0.034016, 0.045418, 0.043951, 0.037546, 0.035939, 0.028038,
0.030356, 0.029002, 0.044163, 0.041439, 0.044517, 0.037571, 0.031181, 0.028760,
0.040765, 0.035363, 0.046820, 0.032099, 0.041736, 0.044093, 0.031336, 0.049597,
0.035668
Outliers: 0.076472
```

Tear Down Delay (AR2, AR3):

```
26.742899, 23.710220, 23.548284, 20.978137, 14.682111, 14.549829, 20.704293,
19.959679, 14.345056, 22.779567, 18.524552, 14.563954, 9.959447, 28.542556,
13.331614, 16.345001, 20.918251, 28.621168, 20.613619, 20.266983, 16.554355,
15.371855, 23.479298, 18.019118, 26.489390, 21.158070, 12.501308, 22.567096,
13.852113, 10.925003, 21.516176, 25.155244, 19.973963
Outliers: 8.058581, 8.341470
```

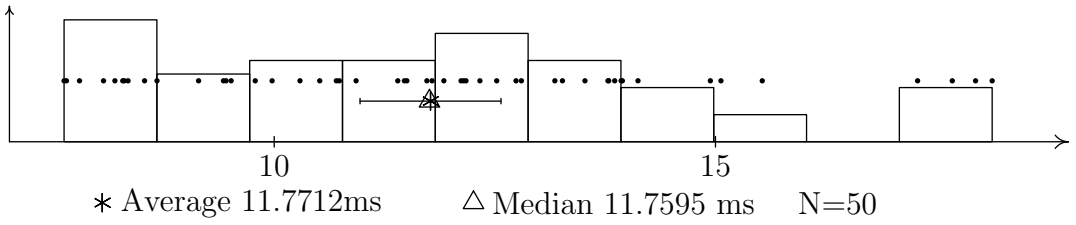


Figure B.1: Setup Delay on AR1

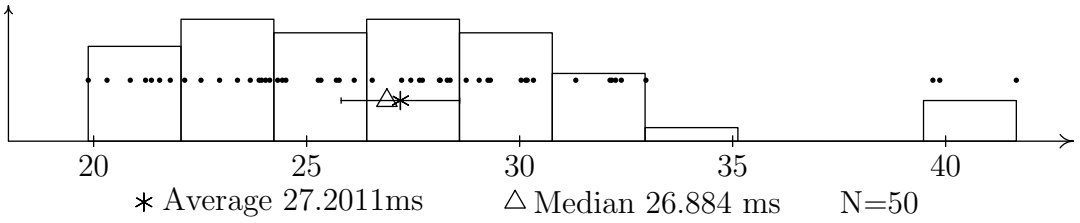


Figure B.2: Setup Delay on AR2

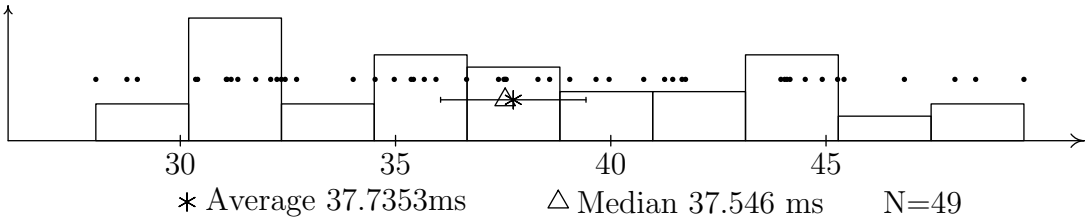


Figure B.3: Setup Delay on AR3

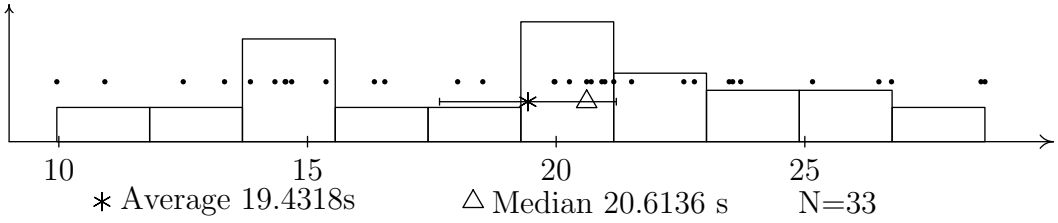


Figure B.4: Tear Down Delay

### B.1.2 CN Sender, Sender-Initiated, No Delay

Setup Delay while at AR1:

```
0.013853, 0.013677, 0.008620, 0.018207, 0.014526, 0.013480, 0.011170, 0.018370,
0.011591, 0.012736, 0.011014, 0.016108, 0.012986, 0.011952, 0.017079, 0.014406,
0.013915, 0.014064, 0.016561, 0.015510, 0.013561, 0.008973, 0.013743, 0.010331,
0.012718, 0.012584, 0.016503, 0.010726, 0.009696, 0.014255, 0.010222, 0.009381,
0.011402, 0.017715, 0.011404, 0.014891, 0.013077, 0.019733, 0.008017, 0.009934,
0.009822, 0.012948, 0.017281, 0.013277, 0.010652, 0.013364, 0.013417, 0.010499,
0.016301, 0.014927
```

Setup Delay while at AR2:

```
0.027436, 0.024706, 0.022963, 0.024295, 0.027411, 0.029320, 0.025256, 0.029543,
0.026114, 0.030737, 0.030648, 0.026370, 0.026736, 0.022031, 0.024890, 0.026803,
0.043281, 0.031497, 0.024593, 0.032592, 0.024908, 0.029950, 0.023932, 0.026164,
0.032924, 0.026957, 0.030544, 0.030144, 0.042098, 0.028132, 0.021989, 0.030204,
0.030728, 0.026715, 0.029466, 0.027129, 0.027927, 0.025110, 0.020873, 0.036279,
0.024636, 0.031318, 0.022925, 0.027564, 0.028313, 0.023412, 0.032547, 0.026641,
0.024771
Outliers: 0.050248
```

Setup Delay while at AR3:

```
0.045429, 0.033002, 0.040301, 0.043696, 0.038313, 0.054775, 0.047530, 0.040652,
0.047775, 0.037338, 0.035374, 0.035397, 0.050548, 0.045884, 0.034726, 0.036977,
0.039434, 0.037811, 0.050202, 0.041492, 0.034667, 0.034128, 0.045941, 0.037122,
0.044108, 0.037110, 0.043798, 0.050324, 0.045102, 0.039150, 0.033119, 0.038543,
0.036980, 0.036103, 0.038706, 0.060334, 0.036440, 0.045945, 0.038366, 0.031133,
0.049726, 0.044151, 0.033910, 0.037396, 0.044054, 0.044486, 0.047568, 0.035084,
0.048270
Outliers: 0.137320
```

Tear Down Delay (AR2, AR3):

```
0.025797, 0.024459, 0.022863, 0.023722, 0.024853, 0.028117, 0.025109, 0.028199,
0.023646, 0.029961, 0.028349, 0.026780, 0.026984, 0.019619, 0.023016, 0.026968,
0.039827, 0.031904, 0.023267, 0.032040, 0.024094, 0.029495, 0.023620, 0.026103,
0.032388, 0.027057, 0.028090, 0.029673, 0.039514, 0.025409, 0.021595, 0.028975,
0.027693, 0.024802, 0.029138, 0.023688, 0.027794, 0.024176, 0.019784, 0.034716,
0.022574, 0.028327, 0.022687, 0.024761, 0.027833, 0.023617, 0.029052, 0.026066,
0.023980
Outliers: 0.048519
```

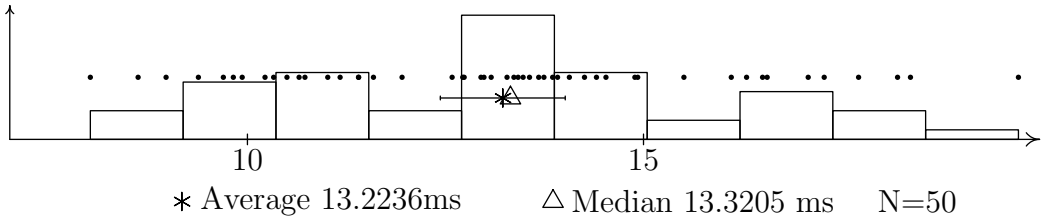


Figure B.5: Setup Delay on AR1

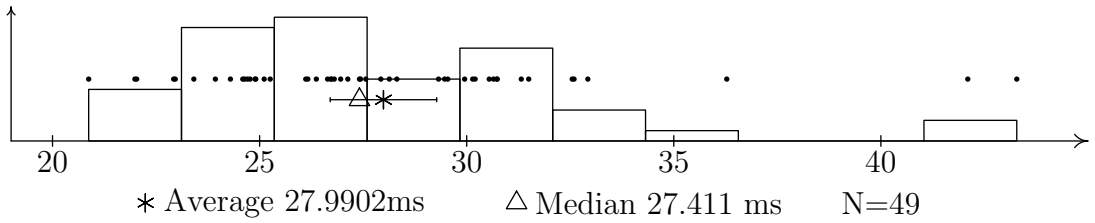


Figure B.6: Setup Delay on AR2

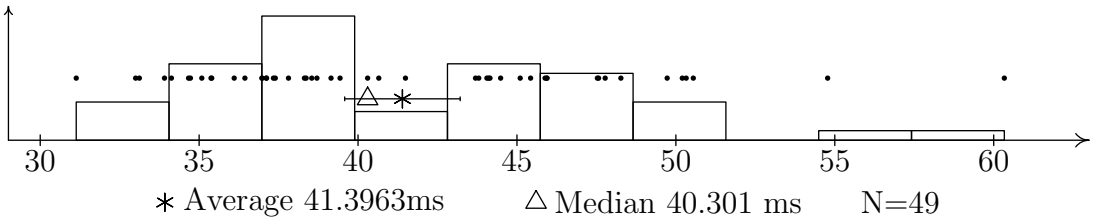


Figure B.7: Setup Delay on AR3

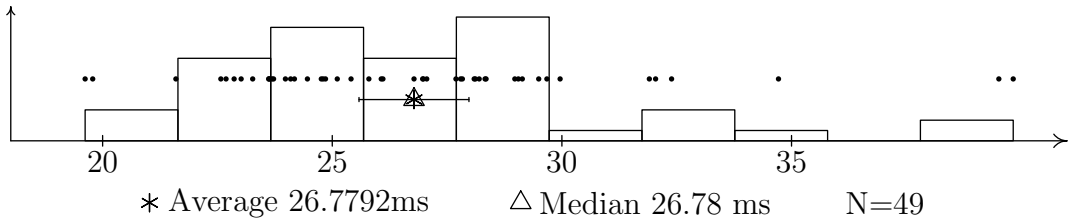


Figure B.8: Tear Down Delay

### B.1.3 MN Sender, Reciever-Initiated, No Delay

Setup Delay while at AR1:

```
0.009549, 0.013223, 0.013150, 0.016289, 0.011553, 0.011085, 0.017727, 0.009088,
0.017577, 0.016956, 0.008835, 0.007393, 0.013013, 0.013031, 0.014285, 0.009041,
0.014385, 0.009794, 0.009203, 0.010453, 0.011043, 0.008841, 0.011008, 0.014319,
0.014357, 0.016801, 0.010070, 0.009354, 0.013038, 0.012135, 0.010948, 0.009626,
0.010224, 0.013657, 0.012124, 0.009211, 0.011103, 0.012148, 0.013222, 0.009515,
0.013026, 0.012334, 0.010464, 0.014167, 0.018161, 0.008273, 0.010073, 0.010098
```

Setup Delay while at AR2:

```
0.023616, 0.026303, 0.030743, 0.028876, 0.029239, 0.033295, 0.026816, 0.027549,
0.038333, 0.027046, 0.027339, 0.029271, 0.028983, 0.026786, 0.028789, 0.032618,
0.034650, 0.028434, 0.030747, 0.025382, 0.028822, 0.028896, 0.032336, 0.032947,
0.026064, 0.026344, 0.037159, 0.026355, 0.039783, 0.030619, 0.026919, 0.028398,
0.028936, 0.036387, 0.025703, 0.042426, 0.029747, 0.028705, 0.030151, 0.034197,
0.037016, 0.039741, 0.027395, 0.037813, 0.030815, 0.036808, 0.028559, 0.028070
```

Setup Delay while at AR3:

```
0.047586, 0.042326, 0.039712, 0.040412, 0.038627, 0.049030, 0.076126, 0.052299,
0.037177, 0.044387, 0.040145, 0.047906, 0.039705, 0.036267, 0.036380, 0.046564,
0.039302, 0.041355, 0.040041, 0.052149, 0.034855, 0.050681, 0.041220, 0.046398,
0.043192, 0.044084, 0.036491, 0.043129, 0.037386, 0.045907, 0.041980, 0.046706,
0.041291, 0.055113, 0.042117, 0.043042, 0.047064, 0.045692, 0.038936, 0.045215,
0.063507, 0.035519, 0.039352, 0.044536, 0.039163, 0.051235, 0.052225
```

Tear Down Delay (AR2, AR3):

```
0.022983, 0.025580, 0.030420, 0.026824, 0.027970, 0.033298, 0.026484, 0.027166,
0.035942, 0.026116, 0.027126, 0.026535, 0.027986, 0.026269, 0.028043, 0.032122,
0.027003, 0.027674, 0.030485, 0.024962, 0.028490, 0.028078, 0.032100, 0.033016,
0.023047, 0.026350, 0.035900, 0.025987, 0.037160, 0.029716, 0.026517, 0.027396,
0.027347, 0.035347, 0.025599, 0.041065, 0.029011, 0.026391, 0.029411, 0.034230,
0.036853, 0.033686, 0.027822, 0.031134, 0.030301, 0.036525, 0.028194, 0.027517
```

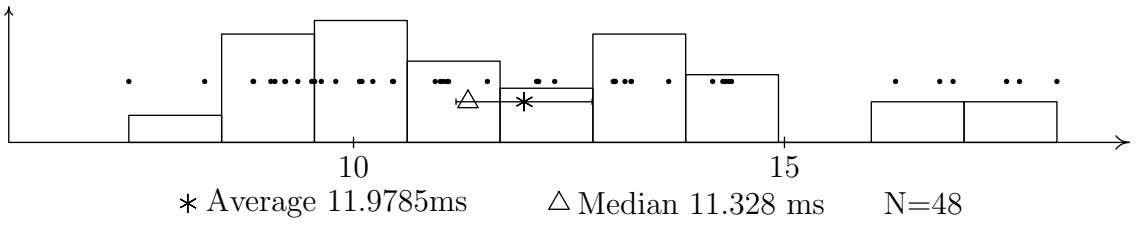


Figure B.9: Setup Delay on AR1

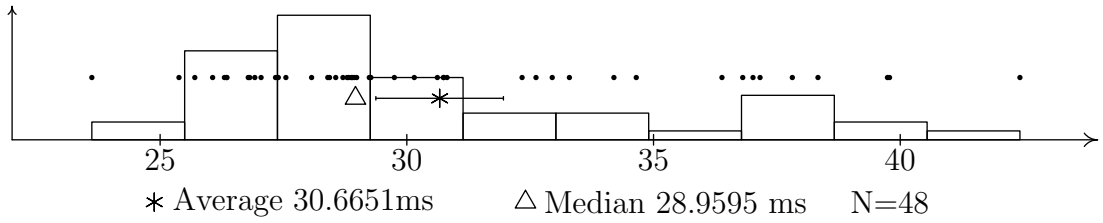


Figure B.10: Setup Delay on AR2

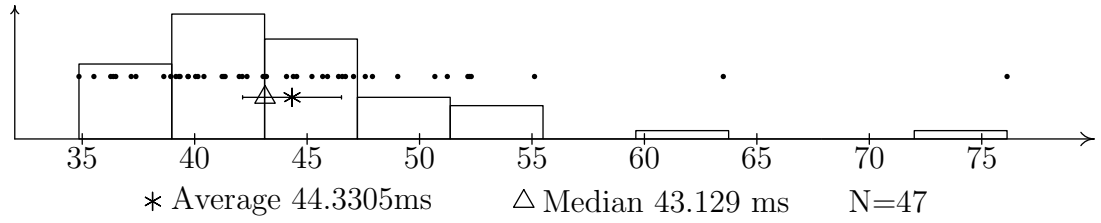


Figure B.11: Setup Delay on AR3

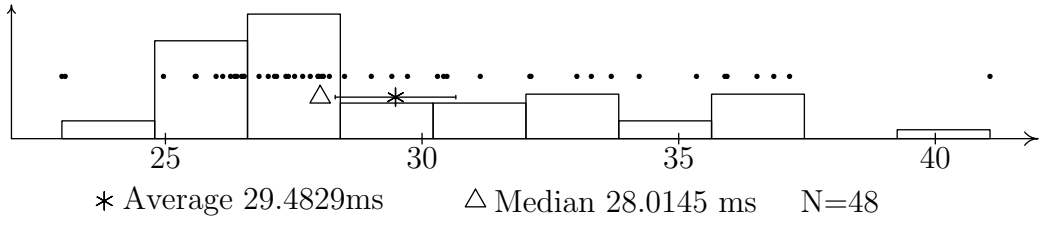


Figure B.12: Tear Down Delay

### B.1.4 CN Sender, Reciever-Initiated, No Delay

Setup Delay while at AR1:

```
0.010294, 0.017332, 0.012004, 0.012379, 0.013701, 0.014845, 0.010394, 0.013158,
0.011129, 0.015902, 0.016299, 0.010298, 0.020515, 0.009975, 0.011348, 0.010209,
0.010694, 0.014203, 0.019553, 0.015132, 0.019320, 0.021090, 0.014132, 0.009980,
0.017118, 0.013048, 0.014840, 0.009378, 0.014087, 0.011098, 0.023312, 0.011660,
0.010581, 0.010888, 0.012914, 0.009750, 0.012491, 0.011651, 0.015257, 0.010296,
0.013901, 0.009644, 0.010009, 0.011484, 0.009504, 0.009691, 0.015895, 0.008953,
0.014638
Outliers: 0.032293
```

Setup Delay while at AR2:

```
0.028894, 0.030356, 0.036248, 0.029475, 0.028105, 0.035612, 0.038178, 0.030577,
0.027729, 0.039643, 0.036938, 0.035944, 0.034448, 0.039552, 0.034640, 0.028093,
0.029647, 0.042444, 0.029197, 0.031488, 0.034024, 0.036046, 0.034787, 0.035417,
0.038291, 0.045247, 0.032693, 0.026415, 0.034683, 0.031466, 0.030707, 0.031561,
0.026901, 0.031366, 0.033734, 0.038634, 0.039605, 0.034822, 0.028109, 0.034923,
0.030162, 0.034663, 0.040779, 0.032853, 0.033013, 0.027059, 0.033433, 0.031692,
0.033372, 0.031311
```

Setup Delay while at AR3:

```
0.036377, 0.040400, 0.053457, 0.042108, 0.048257, 0.053646, 0.035054, 0.045853,
0.046708, 0.040201, 0.052751, 0.040009, 0.053269, 0.038874, 0.040076, 0.043744,
0.044630, 0.041820, 0.045384, 0.047805, 0.047128, 0.046598, 0.053503, 0.041560,
0.034691, 0.040699, 0.042717, 0.052997, 0.038704, 0.045936, 0.042190, 0.045847,
0.042866, 0.038558, 0.060388, 0.037588, 0.045020, 0.035347, 0.040896, 0.042290,
0.036778, 0.036154, 0.038887, 0.036966, 0.039646, 0.038751, 0.043647, 0.043252,
0.040570, 0.044692
```

Tear Down Delay (AR2, AR3):

```
0.028168, 0.029236, 0.035671, 0.028859, 0.027562, 0.032719, 0.037051, 0.029745,
0.027107, 0.039050, 0.036411, 0.035347, 0.033818, 0.037953, 0.034133, 0.027431,
0.027673, 0.035568, 0.028642, 0.030835, 0.031980, 0.035395, 0.034209, 0.031734,
0.037297, 0.044037, 0.030103, 0.025817, 0.034132, 0.030715, 0.030133, 0.030714,
0.026290, 0.028405, 0.033198, 0.037497, 0.036863, 0.034151, 0.027458, 0.032125,
0.029412, 0.033837, 0.038389, 0.031897, 0.032439, 0.026543, 0.029233, 0.028558,
0.032727, 0.028507
```

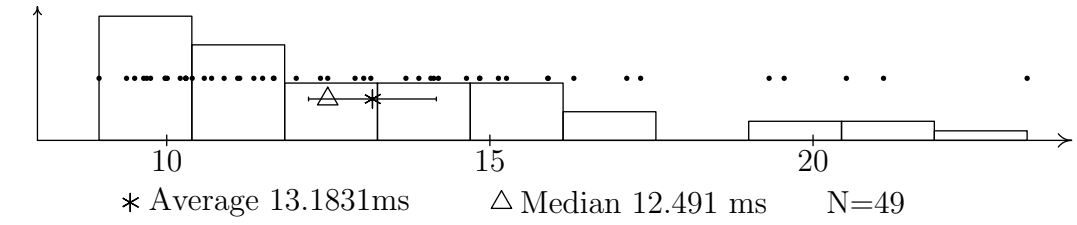


Figure B.13: Setup Delay on AR1

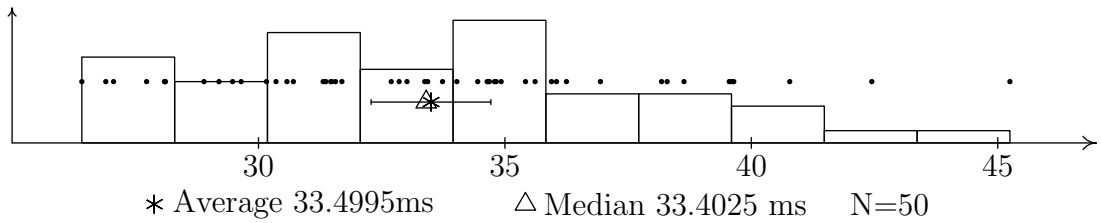


Figure B.14: Setup Delay on AR2

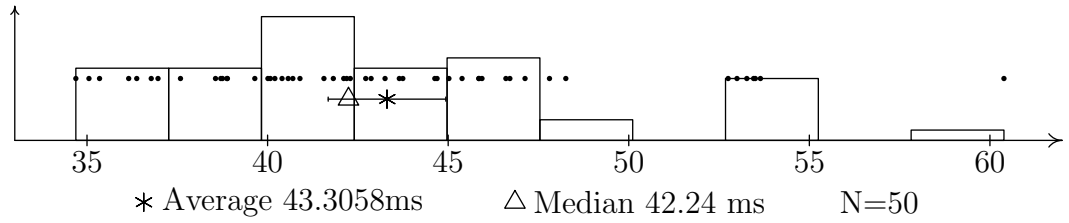


Figure B.15: Setup Delay on AR3

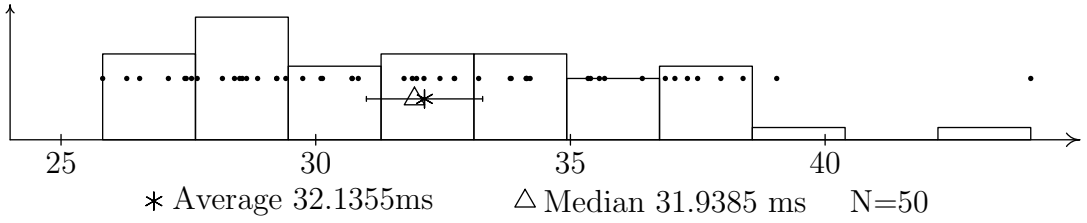


Figure B.16: Tear Down Delay

## B.2 Benchmarks With Additional Delay

### B.2.1 MN Sender, Sender-Initiated, Delay

Setup Delay while at AR2:

```
0.227109, 0.228970, 0.229176, 0.230684, 0.227169, 0.233016, 0.232014, 0.225110,
0.227048, 0.231123, 0.233058, 0.227128, 0.227032, 0.228901, 0.231209, 0.227045,
0.231339, 0.227057, 0.231372, 0.227530, 0.226342, 0.227523, 0.228799, 0.227204,
0.227137, 0.231422, 0.228859, 0.226850, 0.228803, 0.223095, 0.227326, 0.228892,
0.227033, 0.229184, 0.231073, 0.228901, 0.235086, 0.226986, 0.224854, 0.227169,
0.232974, 0.224686, 0.238022, 0.231175, 0.225025, 0.224886, 0.239260, 0.231921,
0.229069, 0.233123
```

Setup Delay while at AR3:

```
0.354903, 0.348908, 0.347158, 0.354969, 0.356937, 0.352916, 0.349000, 0.355027,
0.353245, 0.345081, 0.341408, 0.347381, 0.351049, 0.347104, 0.351357, 0.349070,
0.346979, 0.350951, 0.344929, 0.344854, 0.349045, 0.347098, 0.340886, 0.350885,
0.345224, 0.340693, 0.350991, 0.362780, 0.348834, 0.350975, 0.347104, 0.353748,
0.354985, 0.352808, 0.352992, 0.348738, 0.338586, 0.346952, 0.346665, 0.348980,
0.356901, 0.348830, 0.345316, 0.344806, 0.347159, 0.359581, 0.353376, 0.354706,
0.348879, 0.348808
```

Tear Down Delay (AR2, AR3):

```
18.077434, 10.981500, 15.229439, 18.497418, 16.017421, 24.217348, 15.661473,
23.869390, 9.113504, 14.299384, 18.099454, 20.315390, 10.089499, 20.245386,
21.061389, 16.813445, 20.819439, 16.639435, 21.425389, 20.633367, 17.319406,
17.119428, 12.413468, 20.835389, 15.327464, 15.421401, 23.591377, 19.909392,
20.721353, 14.505444, 20.051438, 15.619351, 16.223436, 19.135393, 20.463313,
11.563494, 22.599388, 21.175400, 19.157416, 21.549392
```

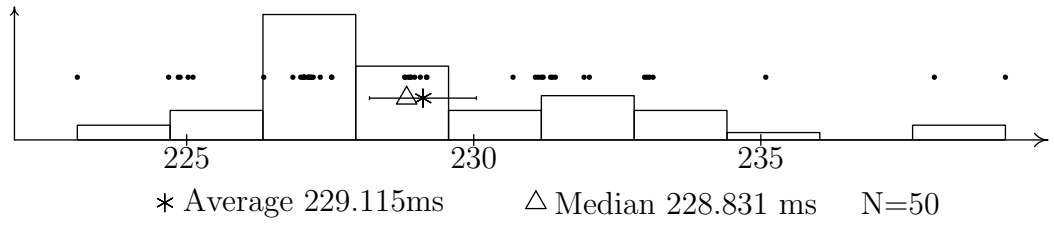


Figure B.17: Setup Delay on AR2

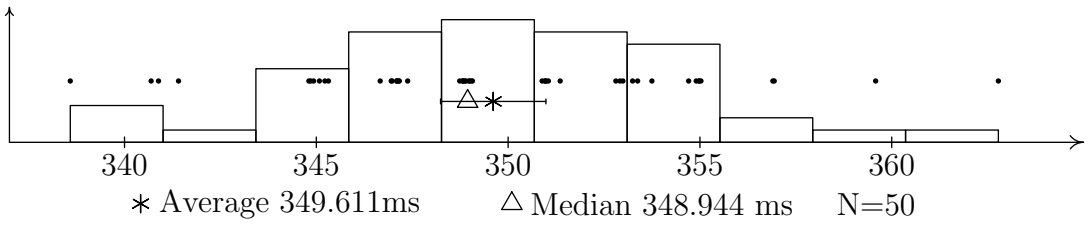


Figure B.18: Setup Delay on AR3

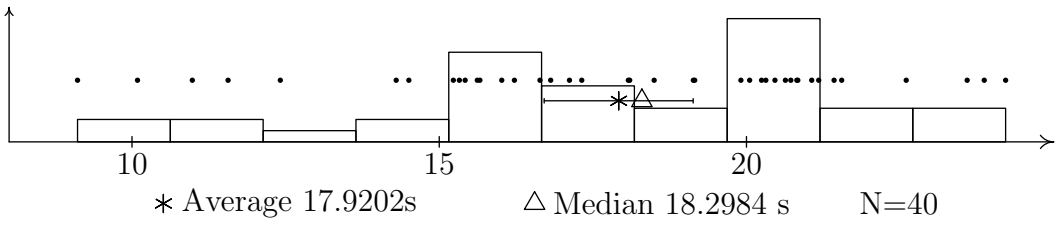


Figure B.19: Tear Down Delay

### B.2.2 CN Sender, Sender-Initiated, Delay

Setup Delay while at AR2:

```
0.228935, 0.226959, 0.240700, 0.238705, 0.226865, 0.235197, 0.229188, 0.235180,
0.226930, 0.226932, 0.233919, 0.233290, 0.232141, 0.229185, 0.227797, 0.242559,
0.230094, 0.225547, 0.243204, 0.230976, 0.238990, 0.239350, 0.240899, 0.229240,
0.228934, 0.234934, 0.233117, 0.235199, 0.232851, 0.233292, 0.226893, 0.231509,
0.230985, 0.228839, 0.228957, 0.235316, 0.235581, 0.226572, 0.235169, 0.228936,
0.228948, 0.239168, 0.228806, 0.230957, 0.235006, 0.226952, 0.225103, 0.227369,
0.235374
```

Setup Delay while at AR3:

```
0.355301, 0.352793, 0.347252, 0.360981, 0.351001, 0.353129, 0.351033, 0.349043,
0.353398, 0.346947, 0.354858, 0.353350, 0.365269, 0.345449, 0.352941, 0.355098,
0.355415, 0.360988, 0.347401, 0.348931, 0.361216, 0.353218, 0.353402, 0.355341,
0.353363, 0.354963, 0.355092, 0.346947, 0.343359, 0.349222, 0.347322, 0.357119,
0.359203, 0.351193, 0.349234, 0.354930, 0.351208, 0.348799, 0.343186, 0.346925,
0.351322, 0.354898, 0.359240, 0.350910, 0.345007, 0.350803, 0.360752, 0.351332,
0.354991
Outliers: 0.405214
```

Tear Down Delay (AR2, AR3):

```
0.207084, 0.200987, 0.213051, 0.206789, 0.200819, 0.208900, 0.205340, 0.208897,
0.201118, 0.204651, 0.209765, 0.209259, 0.203286, 0.205878, 0.201621, 0.214976,
0.205011, 0.201177, 0.217029, 0.206861, 0.213185, 0.210943, 0.214768, 0.204986,
0.204917, 0.208979, 0.208913, 0.211246, 0.209120, 0.209204, 0.203100, 0.207413,
0.207254, 0.205346, 0.203155, 0.208962, 0.209210, 0.202667, 0.209221, 0.204748,
0.204706, 0.214957, 0.203140, 0.204952, 0.209418, 0.208892, 0.200864, 0.206038,
0.210903
```

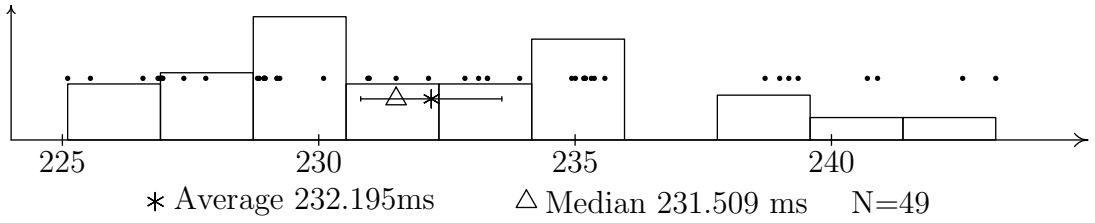


Figure B.20: Setup Delay on AR2

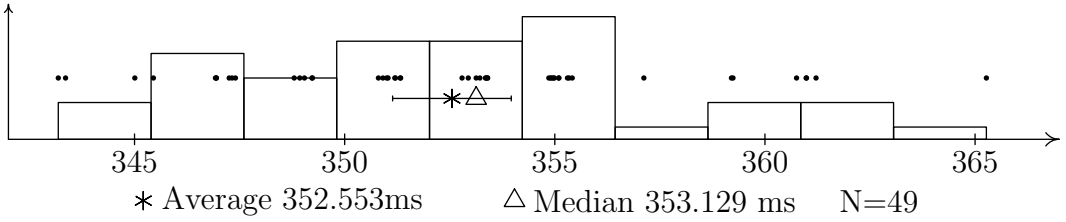


Figure B.21: Setup Delay on AR3

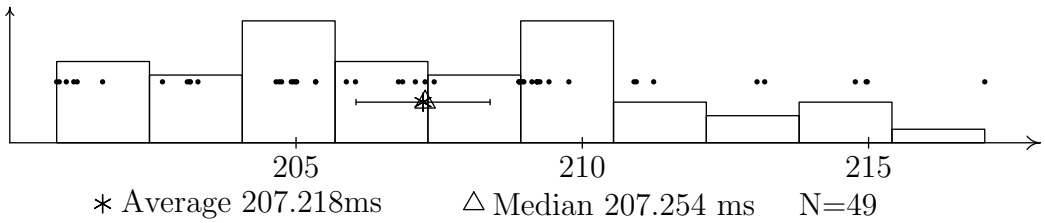


Figure B.22: Tear Down Delay

### B.2.3 MN Sender, Reciever-Initiated, Delay

Setup Delay while at AR2:

```
0.286941, 0.286878, 0.282888, 0.281112, 0.284984, 0.280975, 0.284814, 0.285123,
0.287027, 0.285001, 0.284907, 0.282913, 0.297155, 0.281210, 0.284942, 0.288806,
0.294870, 0.286899, 0.287692, 0.284833, 0.285151, 0.280844, 0.284679, 0.285775,
0.291069, 0.291001, 0.280979, 0.282919, 0.286767, 0.285051, 0.278786, 0.286917,
0.292995, 0.285129, 0.284984, 0.280280, 0.284853, 0.283090, 0.289139, 0.287142,
0.293234, 0.289040, 0.290710, 0.286998, 0.292891, 0.290077, 0.293217, 0.283160,
0.291086, 0.292996
```

Setup Delay while at AR3:

```
0.427275, 0.431290, 0.426709, 0.430330, 0.427939, 0.431060, 0.427211, 0.427381,
0.429008, 0.427156, 0.431020, 0.427112, 0.436938, 0.424787, 0.424925, 0.427124,
0.432764, 0.429037, 0.429143, 0.427048, 0.429237, 0.435469, 0.426982, 0.432958,
0.432344, 0.435277, 0.425293, 0.431107, 0.443606, 0.426963, 0.431014, 0.431217,
0.434848, 0.432877, 0.433169, 0.426883, 0.422907, 0.430817, 0.427024, 0.432816,
0.426969, 0.440866, 0.436932, 0.429317, 0.426731, 0.429014, 0.433285, 0.440856,
0.429474, 0.427163
```

Tear Down Delay (AR2, AR3):

```
0.260857, 0.262828, 0.259010, 0.256946, 0.261112, 0.256982, 0.256942, 0.258974,
0.263013, 0.262040, 0.257308, 0.259162, 0.272911, 0.257006, 0.259360, 0.264661,
0.272892, 0.260877, 0.262879, 0.260880, 0.260904, 0.254840, 0.260691, 0.258847,
0.263038, 0.266712, 0.255564, 0.256869, 0.263233, 0.261244, 0.252842, 0.263123,
0.262943, 0.258846, 0.260885, 0.256337, 0.259624, 0.258989, 0.265536, 0.260930,
0.267139, 0.263355, 0.265108, 0.263432, 0.268961, 0.266995, 0.267170, 0.259193,
0.267201, 0.267263
```

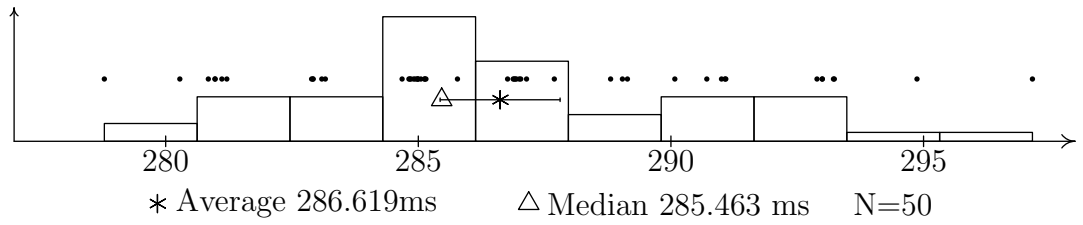


Figure B.23: Setup Delay on AR2

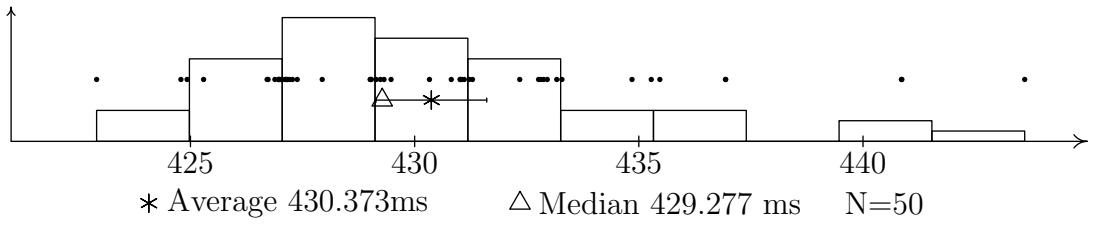


Figure B.24: Setup Delay on AR3

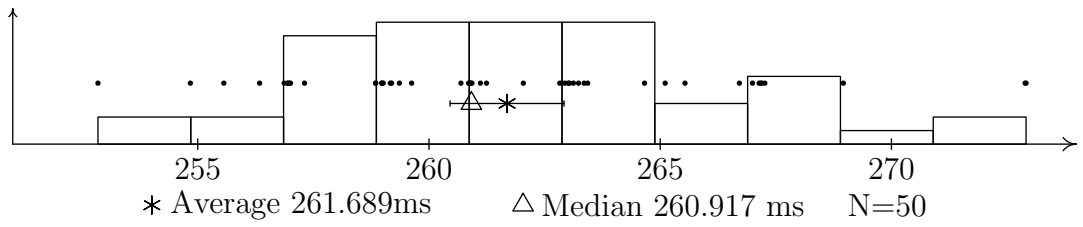


Figure B.25: Tear Down Delay

### B.2.4 CN Sender, Reciever-Initiated, Delay

Setup Delay while at AR2:

```
0.283649, 0.287659, 0.290432, 0.291489, 0.291689, 0.287839, 0.293480, 0.283402,
0.286200, 0.285530, 0.288261, 0.280015, 0.277242, 0.287607, 0.279752, 0.281774,
0.285936, 0.293802, 0.287618, 0.283729, 0.290413, 0.290840, 0.289682, 0.289911,
0.277418, 0.285048, 0.281610, 0.294068, 0.291874, 0.289710, 0.292255, 0.289778,
0.291574, 0.287443, 0.286697, 0.296223, 0.286870, 0.284980, 0.287579, 0.284810,
0.288012, 0.286100, 0.307319, 0.283885, 0.293917, 0.292004, 0.290916
Outliers: 0.234333
```

Setup Delay while at AR3:

```
0.455971, 0.431154, 0.430961, 0.432962, 0.432868, 0.425191, 0.426969, 0.431052,
0.428985, 0.427021, 0.432798, 0.424695, 0.425137, 0.426732, 0.426762, 0.438949,
0.429039, 0.433031, 0.426861, 0.428770, 0.424714, 0.428686, 0.426704, 0.426928,
0.432882, 0.430957, 0.425897, 0.430984, 0.432737, 0.427225, 0.436611, 0.431071,
0.441224, 0.425203, 0.433031, 0.443344, 0.429286, 0.441595, 0.442352, 0.428998,
0.427226, 0.429197, 0.427206, 0.428192
```

Tear Down Delay (AR2, AR3):

```
0.309614, 0.313631, 0.317595, 0.317616, 0.317619, 0.313603, 0.319620, 0.309628,
0.309585, 0.311626, 0.311541, 0.305631, 0.303348, 0.313595, 0.305617, 0.307635,
0.311651, 0.319537, 0.313605, 0.309606, 0.313650, 0.317631, 0.315620, 0.315555,
0.303596, 0.311624, 0.307612, 0.319581, 0.317592, 0.315601, 0.315597, 0.315578,
0.317656, 0.313638, 0.313091, 0.323598, 0.313609
```

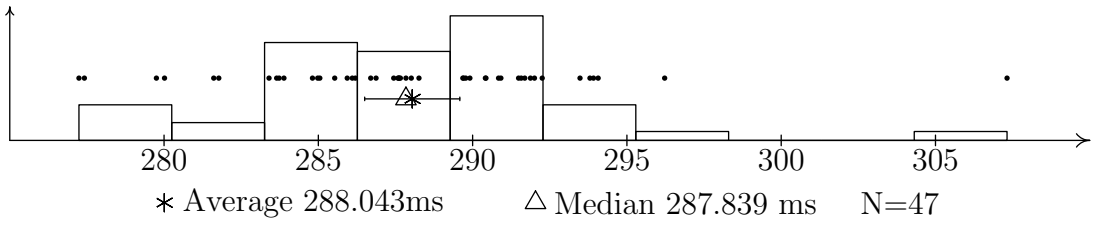


Figure B.26: Setup Delay on AR2

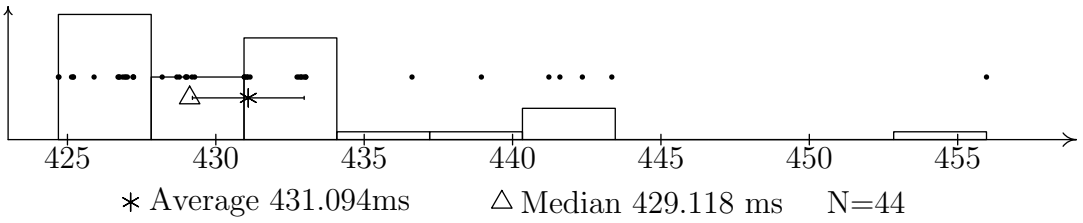


Figure B.27: Setup Delay on AR3

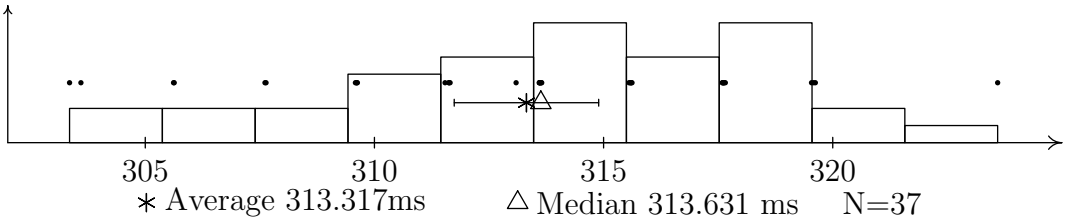


Figure B.28: Tear Down Delay

# Bibliography


[1] *USAGI Project - Linux IPv6 Development Project*. Internet: http://www.linux-ipv6.org/.

[2] Bless, Roland, Thomas Herzog, Markus Ott, Matthias Friedrich, Max Laier and Martin Röhricht: *NSIS Implementation Project: NSIS-ka*. Internet: https://projekte.tm.uka.de/trac/NSIS/, July 2005.

[3] Braden, R., L. Zhang, S. Berson, S. Herzog and S. Jamin: *Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification*. RFC 2205 (Proposed Standard), September 1997. Updated by RFCs 2750, 3936, 4495.

[4] Dell'Uomo, L. and E. Scarrone: *An all-IP solution for QoS mobility management and AAA in the 4G mobile networks*. Wireless Personal Multimedia Communications, 2002. The 5th International Symposium on, 2:591–595 vol.2, Oct. 2002.

[5] Johnson, D., C. Perkins and J. Arkko: *Mobility Support in IPv6*. RFC 3775 (Proposed Standard), June 2004.

[6] Kan, Zhigang, Dongmei Zhang, Runtong Zhang and Jian Ma: *QoS in Mobile IPv6*. In *Info-tech and Info-net, 2001. Proceedings*, volume 2, pages 492–497. Nokia China R&D Center, 2001.

[7] Keeni, G., K. Koide, K. Nagami and S. Gundavelli: *Mobile IPv6 Management Information Base*. RFC 4295 (Proposed Standard), April 2006.

[8] Manner, J., G. Karagiannis and A. McDonald: *NSLP for Quality-of-Service Signaling*. Internet: http://tools.ietf.org/id/draft-ietf-nsis-qos-nslp, February 2008. Revision 16.

[9] Marques, Victor, Rui L. Aguiar, Piotr Pacyna, Janusz Gozdecki, Christophe Beaujean, Carlos Garcia, Jose Ignacio Moreno and Hans Einsiedler: *An Architecture Supporting End-to-End QoS with User Mobility for Systems Beyond 3 rd Generation*, 2002.

[10] Pashalidis, A. and H. Tschofenig: *GIST NAT Traversal*. Internet: http://tools.ietf.org/id/draft-pashalidis-nsis-gimps-nattraversal, July 2007. Revision 05.

[11] Recio, R., B. Metzler, P. Culley, J. Hilland and D. Garcia: *A Remote Direct Memory Access Protocol Specification*. RFC 5040 (Proposed Standard), October 2007.

[12] Rizzo, Luigi: *Dummynet: a simple approach to the evaluation of network protocols*. SIGCOMM Comput. Commun. Rev., 27(1):31–41, 1997.

[13] Sanda, T., X. Fu, J. Manner S. Jeong and H. Tschofenig: *Applicability Statement of NSIS Protocols in Mobile Environments*. Internet: http://tools.ietf.org/id/draft-ietf-nsis-applicability-mobility-signaling, February 2008. Revision 09.

[14] Schulzrinne, H. and R. Hancock: *GIST: General Internet Signalling Transport*. Internet: http://tools.ietf.org/id/draft-ietf-nsis-ntlp, February 2008. Revision 15.

[15] Shen, C., H. Schulzrinne, S. Lee and J. Bang: *NSIS Operation Over IP Tunnels*. Internet: http://tools.ietf.org/id/draft-ietf-nsis-tunnel, March 2008. Revision 04.

[16] Shen, Charles Qi: *Several Framework Issues Regarding NSIS and Mobility*. Internet: http://tools.ietf.org/id/draft-shen-nsis-mobility-fw, July 2002. Revision 00.

[17] Shen, Charles Qi, Winston Seah, Anthony Lo, Haihong Zheng and Marc Greis: *Mobility Extensions to RSVP in an RSVP-Mobile IPv6 Framework*. Internet: http://tools.ietf.org/id/draft-shen-nsis-rsvp-mobileipv6, July 2002. Revision 00.

[18] Steinleitner, N., X. Fu, D. Hogrefe, H. Tschofenig and T. Schreck: *An NSIS-based Approach for Firewall Traversal in Mobile IPv6 Networks*, Oct 2007.

[19] Stiemerling, M., H. Tschofenig, C. Aoun and E. Davies: *NAT/Firewall NSIS Signaling Layer Protocol (NSLP)*. Internet: http://tools.ietf.org/id/draft-ietf-nsis-nslp-natfw, February 2008. Revision 18.