Network Working Group
Request for Comments: 3989
Category: Informational

M. Stiemerling
J. Quittek
NEC
T. Taylor
Nortel

February 2005

Middlebox Communications (MIDCOM) Protocol Semantics

Status of This Memo

Copyright Notice

Abstract

This memo specifies semantics for a Middlebox Communication (MIDCOM) protocol to be used by MIDCOM agents for interacting with middleboxes such as firewalls and Network Address Translators (NATs). The semantics discussion does not include any specification of a concrete syntax or a transport protocol. However, a concrete protocol is expected to implement the specified semantics or, more likely, a superset of it. The MIDCOM protocol semantics is derived from the MIDCOM requirements, from the MIDCOM framework, and from working group decisions.

Table of Contents

1. Introduction

The MIDCOM working group has defined a framework [MDC-FRM] and a list of requirements [MDC-REQ] for middlebox communication. The next step toward a MIDCOM protocol is the specification of protocol semantics that is constrained, but not completely implied, by the documents mentioned above.

This memo suggests a semantics for the MIDCOM protocol. It is fully compliant with the requirements listed in [MDC-REQ] and with the working group's consensus on semantic issues.

In conformance with the working group charter, the semantics description is targeted at packet filters and network address translators (NATs), and it supports applications that require dynamic configuration of these middleboxes.

The semantics is defined in terms of transactions. Two basic types of transactions are used: request-reply transactions and asynchronous transactions. For each transaction, the semantics is specified by describing (1) the parameters of the transaction, (2) the processing of request messages at the middlebox, and (3) the state transitions

at the middlebox caused by the request transactions or indicated by the asynchronous transactions, respectively, and (4) the reply and notification messages sent from the middlebox to the agent in order to inform the agent about the state change.

The semantics can be implemented by any protocol that supports these two transaction types and that is sufficiently flexible concerning transaction parameters. Different implementations for different protocols might need to extend the semantics described below by adding further transactions and/or adding further parameters to transactions and/or splitting single transactions into a set of transactions. Regardless of such extensions, the semantics below provides a minimum necessary subset of what must be implemented.

The remainder of this document is structured as follows. Section 2 describes the protocol semantics. It is structured in four subsections:

- General Protocol Issues (section 2.1)
- Session Control (section 2.2)
- Policy Rules (section 2.3)
- Policy Rule Groups (section 2.4)

Section 3 contains conformance statements for MIDCOM protocol definitions and MIDCOM protocol implementations with respect to the semantics defined in section 2. Section 4 gives two elaborated usage examples. Finally, section 5 explains how the semantics meets the MIDCOM requirements.

1.1. Terminology

The terminology in this memo follows the definitions given in the framework [MDC-FRM] and requirements [MDC-REQ] document.

In addition, the following terms are used:

request transaction
: A request transaction consists of a request message transfer from the agent to the middlebox, processing of the message at the middlebox, a reply message transfer from the middlebox to the agent, and the optional transfer of notification messages from the middlebox to agents other than the one requesting the transaction. A request transaction might cause a state transition at the middlebox.

configuration transaction	A configuration transaction is a request transaction containing a request for state change in the middlebox. If accepted, it causes a state change at the middlebox.
monitoring transaction	A monitoring transaction is a request transaction containing a request for state information from the middlebox. It does not cause a state transition at the middlebox.
asynchronous transaction	An asynchronous transaction is not triggered by an agent. It may occur without any agent participating in a session with the middlebox. Potentially, an asynchronous transaction includes the transfer of notification messages from the middlebox to agents that participate in an open session. A notification message is sent to each agent that needs to be notified about the asynchronous event. The message indicates the state transition at the middlebox.
agent-unique	An agent-unique value is unique in the context of the agent. This context includes all MIDCOM sessions the agent participates in. An agent-unique value is assigned by the agent.
middlebox-unique	A middlebox-unique value is unique in the context of the middlebox. This context includes all MIDCOM sessions the middlebox participates in. A middlebox-unique value is assigned by the middlebox.
policy rule	In general, a policy rule is "a basic building block of a policy-based system. It is the binding of a set of actions to a set of conditions -- where the conditions are evaluated to determine whether the actions are performed." [RFC3198]. In the MIDCOM context the condition is a specification of a set of packets to which rules are applied. The set of actions always contains just a single element per rule, either action "reserve" or action "enable".

policy reserve rule	A policy rule containing a reserve action. The policy condition of this rule is always true. The action is the reservation of just an IP address or a combination of an IP address and a range of port numbers on neither side, one side, or both sides of the middlebox, depending on the middlebox configuration.
policy enable rule	A policy rule containing an enable action. The policy condition consists of a descriptor of one or more unidirectional or bidirectional packet flows, and the policy action enables packets belonging to this flow to traverse the middlebox. The descriptor identifies the protocol, the flow direction, and the source and destination addresses, optionally with a range of port numbers.
NAT binding	The term NAT binding as used in this document does not necessarily refer to a NAT bind as defined in [NAT-TERM]. A NAT binding in the MIDCOM semantics refers to an abstraction that enables communication between two end points through the NAT-type middlebox. An enable action may result in a NAT bind or a NAT session, depending on the request and its parameters.

1.2. Transaction Definition Template

In the following sections, the semantics of the MIDCOM protocol is specified per transaction. A transaction specification contains the following entries. Parameter entries, failure reason, and notification message type are only specified if applicable.

transaction-name
: A description name for this type of transaction.

transaction-type
: The transaction type is either 'configuration', 'monitoring', or 'asynchronous'. See section 1.1 for a description of transaction types.

transaction-compliance
: This entry contains either ’mandatory’ or ’optional’. For details see section 2.1.8.

request-parameters
: This entry lists all parameters necessary for this request. A description for each parameter is given.

reply-parameters (success)
: This entry lists all parameters sent back from the middlebox to the agent as positive response to the prior request. A description for each parameter is given.

failure reason
: All negative replies have two parameters: a request identifier identifying the request on which the reply is sent and a parameter indicating the failure reason. As these parameters are compulsory, they are not listed in the template. But the template contains a list of potential failure reasons that may be indicated by the second parameter. The list is not exhaustive. A concrete protocol specification may extend the list.

notification message type
: The type of the notification message type that may be used by this transaction.

semantics
: This entry describes the actual semantics of the transaction. Particularly, it describes the processing of the request message by the middlebox, and middlebox state transitions caused by or causing the transaction, respectively.

2. Semantics Specification

2.1. General Protocol Design

The semantics specification aims at a balance between proper support of applications that require dynamic configuration of middleboxes and simplicity of specification and implementation of the protocol.

Protocol interactions are structured into transactions. The state of middleboxes is described by state machines. The state machines are defined by states and state transitions. A single transaction may cause or be caused by state transitions in more than one state machine, but per state machine there is no more than one transition per transaction.

2.1.1. Protocol Transactions

State transitions are initiated either by a request message from the agent to the middlebox or by some other event at the middlebox. In the first case, the middlebox informs the agent by sending a reply message on the actual state transition; in the second, the middlebox sends an unsolicited asynchronous notification message to each agent affected by the transaction (if it participates in an open session with the middlebox).

Request and reply messages contain an agent-unique request identifier that allows the agent to determine to which sent request a received reply corresponds.

An analysis of the requirements showed that four kinds of transactions are required:

- Configuration transactions allowing the agent to request state transitions at the middlebox.

- Asynchronous transactions allowing the middlebox to change state without a request by an agent.

- Monitoring transactions allowing the agent to request state information from the middlebox.

- Convenience transactions combining a set of configuration transactions.

Configuration transactions and asynchronous transactions provide the basic MIDCOM protocol functionality. They are related to middlebox state transitions, and they concern establishment and termination of MIDCOM sessions and of policy rules.

Monitoring transactions are not related to middlebox state transitions. They are used by agents to explore the number, status, and properties of policy rules established at the middlebox.

Convenience transactions simplify MIDCOM sessions by combining a set of configuration transactions into a single one. They are not necessary for MIDCOM protocol operation.

As specified in detail in section 3, configuration transactions and asynchronous transactions are mandatory. They must be implemented by a compliant middlebox. All convenience transactions are optional, and some of the monitoring transactions are optional.

2.1.2. Message Types

The MIDCOM protocol supports three kinds of messages: request messages, reply messages, and notification messages. For each kind, different message types exist. In this semantics document, message types are only defined by the list of parameters. The order of the parameters and their encoding is left to a concrete protocol definition. A protocol definition may also add further parameters to a message type or combine several parameters into one, as long as the information contained in the parameters defined in the semantics is still present.

For request messages and positive reply messages there exists one message type per request transaction. Each reply transaction defines the parameter list of the request message and of the positive (successful) reply message by using the transaction definition template defined in section 1.2.

In case of a failed request transaction, a negative reply message is sent from the middlebox to the agent. This message is the same for all request transactions; it contains the request identifier identifying the request to which the reply is sent and a parameter indicating the failure reason.

There are three notification message types: the Session Termination Notification (STN), the Policy Rule Event Notification (REN), and the Group Event Notification (GEN). All of these contain a middlebox-unique notification identifier.

STN The Session Termination Notification message additionally contains a single parameter indicating the reason for session termination by the middlebox.

REN The Policy Rule Event Notification message contains the notification identifier, a policy rule identifier, and the remaining policy lifetime.

GEN The Group Event Notification message contains the notification identifier, a policy rule group identifier, and the remaining policy rule group lifetime.

2.1.3. Session, Policy Rule, and Policy Rule Group

All transactions can be further grouped into transactions concerning sessions, transactions concerning policy rules, and transactions concerning policy rule groups. Policy rule groups can be used to

indicate relationships between policy rules and to simplify transactions on a set of policy rules by using a single transaction per group instead of one per policy rule.

Sessions and policy rules at the middlebox are stateful. Their states are independent of each other, and their state machines (one per session and one per policy rule) can be separated. Policy rule groups are also stateful, but the middlebox does not need to maintain state for policy rule groups, because the semantics were chosen so that the policy rule group state is implicitly defined by the state of all policy rules belonging to the group (see section 2.4).

The separation of session state and policy rule state simplifies the specification of the semantics as well as a protocol implementation. Therefore, the semantics specification is structured accordingly and we use two separated state machines to illustrate the semantics. Please note that state machines of concrete protocol designs and implementations will probably be more complex than the state machines presented here. However, the protocol state machines are expected to be a superset of the semantics state machines in this document.

2.1.4. Atomicity

All request transactions are atomic with respect to each other. This means that processing of a request at the middlebox is never interrupted by another request arriving or already queued. This particularly applies when the middlebox concurrently receives requests originating in different sessions. However, asynchronous transactions may interrupt and/or terminate processing of a request at any time.

All request transactions are atomic from the point of view of the agent. The processing of a request does not start before the complete request arrives at the middlebox. No intermediate state is stable at the middlebox, and no intermediate state is reported to any agent.

The number of transactions specified in this document is rather small. Again, for simplicity, we reduced it to a minimal set that still meets the requirements. A real implementation of the protocol might require splitting some of the transactions specified below into two or more transactions of the respective protocol. Reasons for this might include constraints of the particular protocol or the desire for more flexibility. In general this should not be a problem. However, it should be considered that this might change atomicity of the affected transactions.

2.1.5. Access Control

Ownership determines access to policy rules and policy rule groups. When a policy rule is created, a middlebox-unique identifier is generated to identify it in further transactions. Beyond the identifier, each policy rule has an owner. The owner is the authenticated agent that established the policy rule. The middlebox uses the owner attribute of a policy rule to control access to it; each time an authenticated agent requests to modify an existing policy rule, the middlebox determines the owner of the policy rule and checks whether the requesting agent is authorized to perform transactions on the owning agent's policy rules.

All policy rules belonging to the same policy rule group must have the same owner. Therefore, authenticated agents have access either to all members of a policy rule group, or to none of them.

The middlebox may be configured to allow specific authenticated agents to access and modify policy rules with certain specific owners. Certainly, a reasonable default configuration would let each agent access its own policy rules. Also, it might be good to configure an agent identity to act as administrator, allowing modification of all policy rules owned by any agent. However, the configuration of authorization at the middlebox is out of scope of the MIDCOM semantics and protocol.

2.1.6. Middlebox Capabilities

For several reasons it is useful that at session establishment the agent learns about particular capabilities of the middlebox. Therefore, the session establishment procedure described in section 2.2.1 includes a transfer of capability information from the middlebox to the agent. The list of covered middlebox capabilities includes the following:

- Support of firewall function
- List of supported NAT functions, perhaps including
    - address translation
    - port translation
    - protocol translation
    - twice-NAT
- Internal IP address wildcard support
- External IP address wildcard support
- Port wildcard support
- Supported IP version(s) for internal network: IPv4, IPv6, or both

- Supported IP version(s) for external network: IPv4, IPv6, or both
- List of supported optional MIDCOM protocol transactions
- Optional interface-specific policy rule support: not supported or supported
- Policy rule persistence: persistent or non-persistent (a rule is persistent when the middlebox can save the rule to a non-volatile memory, e.g., a hard disk or flash memory)
- Maximum remaining lifetime of a policy rule or policy rule group
- Idle-timeout of policy rules in the middlebox (reserved and enabled policy rules not used by any data traffic for the time of this idle-timeout are deleted automatically by the middlebox; for the deletion of policy rules by middleboxes, see section 2.3.13 about Asynchronous Policy Rule Event).
- Maximum number of simultaneous MIDCOM sessions

The list of middlebox capabilities may be extended by a concrete protocol specification with further information useful for the agent.

2.1.7. Agent and Middlebox Identifiers

To allow both agents and middleboxes to maintain multiple sessions, each request message contains a parameter identifying the requesting agent, and each reply message and each notification message contains a parameter identifying the middlebox. These parameters are not explicitly listed in the description of the individual transactions, because they are common to all of them. They are not further referenced in the individual semantics descriptions. Although, they are not necessarily passed explicitly as parameters of the MIDCOM protocol, they might be provided by the underlying (secure) transport protocol being used. Agent identifiers at the middlebox are middlebox-unique, and middlebox identifiers at the agent are agent-unique, respectively.

2.1.8. Conformance

The MIDCOM requirements in [MDC-REQ] demand capabilities of the MIDCOM protocol that are met by the set of transactions specified below. However, an actual implementation of a middlebox may support only a subset of these transactions. The set of announced supported transactions may be different for different authenticated agents. The middlebox informs the authenticated agent with the capability exchange at session establishment about the transactions that the agent is authorized to perform. Some transactions need to be offered to every authenticated agent.

Each transaction definition below has a conformance entry that contains either 'mandatory' or 'optional'. A mandatory transaction needs to be implemented by every middlebox offering MIDCOM service and must be must be offered to each of the authenticated agents. An optional transaction does not necessarily need to be implemented by a middlebox; it may offer these optional transactions only to certain authenticated agents. The middlebox may offer one, several, all, or no optional transactions to the agents. Whether an agent is allowed to use an optional request transaction is determined by the middlebox's authorization procedure, which is not further specified by this document.

2.2. Session Control Transactions

Before any transaction on policy rules or policy rule groups is possible, a valid MIDCOM session must be established. A MIDCOM session is an authenticated and authorized association between agent and middlebox. Sessions are initiated by agents and can be terminated by either the agent or the middlebox. Both agent and middlebox may participate in several sessions (with different entities) at the same time. To distinguish different sessions, each party uses local session identifiers.

All transactions are transmitted within this MIDCOM session.

Session control is supported by three transactions:

- Session Establishment (SE)
- Session Termination (ST)
- Asynchronous Session Termination (AST)

The first two are configuration transactions initiated by the agent, and the last one is an asynchronous transaction initiated by the middlebox.

2.2.1. Session Establishment (SE)

transaction-name: session establishment

transaction-type: configuration

transaction-compliance: mandatory

request-parameters:

- request identifier: An agent-unique identifier for matching corresponding request and reply at the agent.

- version: The version of the MIDCOM protocol.

- middlebox authentication challenge (mc): An authentication challenge token for authentication of the middlebox. As seen below, this is present only in the first iteration of the request.

- agent authentication (aa): An authentication token authenticating the agent to the middlebox. As seen below, this is updated in the second iteration of the request with material responding to the middlebox challenge.

reply-parameters (success):

- request identifier: An identifier matching the identifier request.

- middlebox authentication (ma): An authentication token authenticating the middlebox to the agent.

- agent challenge token (ac): An authentication challenge token for the agent authentication.

- middlebox capabilities: A list describing the middlebox's capabilities. See section 2.1.6 for the list of middlebox capabilities.

failure reason:

- authentication failed
- no authorization
- protocol version of agent and middlebox do not match
- lack of resources

semantics:

This session establishment transaction is used to establish a MIDCOM session. For mutual authentication of both parties two subsequent session establishment transactions are required as shown in Figure 1.

Figure 1: Mutual authentication of agent and middlebox

Session establishment may be simplified by using only a single transaction. In this case, server challenge and agent challenge are omitted by the sender or ignored by the receiver, and authentication must be provided by other means, for example by TLS [RFC2246] or IPsec [RFC2402][RFC2406].

The middlebox checks with its policy decision point whether the requesting agent is authorized to open a MIDCOM session. If it is not, the middlebox generates a negative reply with 'no authorization' as failure reason. If authentication and authorization are successful, the session is established, and the agent may start with requesting transactions on policy rules and policy rule groups.

Part of the successful reply is an indication of the middlebox's capabilities.

2.2.2. Session Termination (ST)

transaction-name: session termination

transaction-type: configuration

transaction-compliance: mandatory

request-parameters:

- request identifier: An agent-unique identifier for matching corresponding request and reply at the agent.

reply-parameters (success only):

- request identifier: An identifier matching the identifier of the request.

semantics:

This transaction is used to close the MIDCOM session on behalf of the agent. After session termination, the middlebox keeps all established policy rules until their lifetime expires or until an event occurs that causes the middlebox to terminate them.

The middlebox always generates a successful reply. After sending the reply, the middlebox will not send any further messages to the agent within the current session. It also will not process any further request within this session that it received while processing the session termination request, or that it receives later.

2.2.3. Asynchronous Session Termination (AST)

transaction-name: asynchronous session termination

transaction-type: asynchronous

transaction-compliance: mandatory

notification message type: Session Termination Notification (STN)

reply-parameters (success only):

- termination reason: The reason why the session is terminated.

semantics:

The middlebox may decide to terminate a MIDCOM session at any time. Before terminating the actual session the middlebox generates a STN message and sends it to the agent. After sending the notification, the middlebox will not process any further request by the agent, even if it is already queued at the middlebox.

After session termination, the middlebox keeps all established policy rules until their lifetime expires or until an event occurs for which the middlebox terminates them.

Unlike in other asynchronous transactions, no more than one notification is sent, because there is only one agent affected by the transaction.

2.2.4. Session Termination by Interruption of Connection

If a MIDCOM session is based on an underlying network connection, the session can also be terminated by an interruption of this connection. If the middlebox detects this, it immediately terminates the session. The effect on established policy rules is the same as for the Asynchronous Session Termination.

2.2.5. Session State Machine

A state machine illustrating the semantics of the session transactions is shown in Figure 2. The transaction abbreviations used can be found in the headings of the particular transaction section.

All sessions start in state CLOSED. If mutual authentication is already provided by other means, a successful SE transaction can cause a state transition to state OPEN. Otherwise, it causes a transition to state NOAUTH. From this state a failed second SE transaction returns to state CLOSED. A successful SE transaction causes a transition to state OPEN. At any time, an AST transaction or a connection failure may occur, causing a transition to state CLOSED. A successful ST transaction from either NOAUTH or OPEN also causes a return to CLOSED. The parameters of the transactions are explained in Figure 2; the value mc=0 represents an empty middlebox challenge.

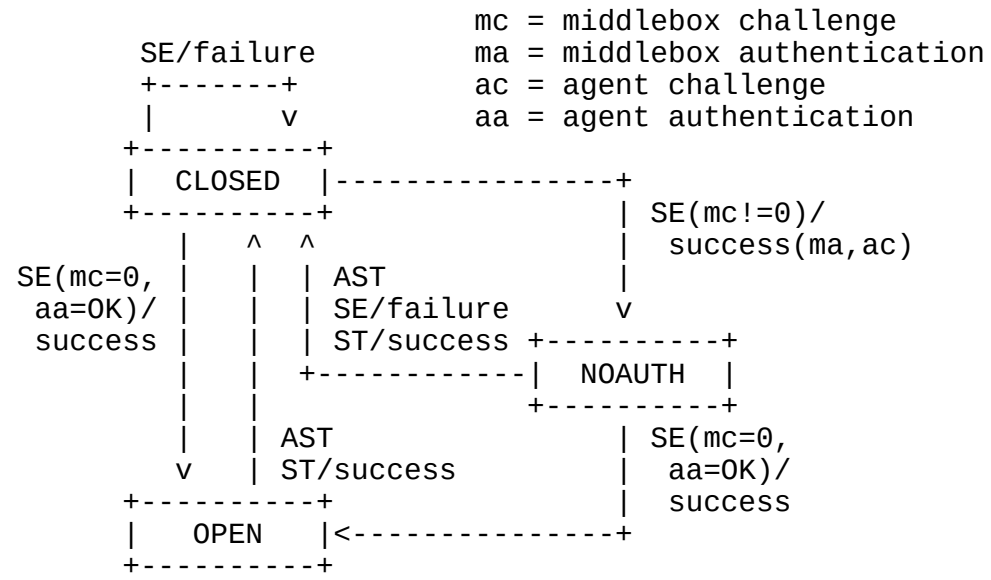

Figure 2: Session State Machine

2.3. Policy Rule Transactions

This section describes the semantics for transactions on policy rules. The following transactions are specified:

- Policy Reserve Rule (PRR)
- Policy Enable Rule (PER)
- Policy Rule Lifetime Change (RLC)
- Policy Rule List (PRL)
- Policy Rule Status (PRS)
- Asynchronous Policy Rule Event (ARE)

The first three transactions (PRR, PER, RLC) are configuration transactions initiated by the agent. The fourth and fifth (PRL, PRS) are monitoring transactions. The last one (ARE) is an asynchronous transaction. The PRL and PRS and transactions do not have any effect on the policy rule state machine.

Before any transaction can start, a valid MIDCOM session must be established.

2.3.1. Configuration Transactions

Policy Rule transactions PER and RLC constitute the core of the MIDCOM protocol. Both are mandatory, and they serve for

- configuring NAT bindings (PER)
- configuring firewall pinholes (PER)
- extending the lifetime of established policy rules (RLC)
- deleting policy rules (RLC)

Some cases require knowing in advance which IP address (and port number) would be chosen by NAT in a PER transaction. This information is required before sufficient information for performing a complete PER transaction is available (see example in section 4.2). For supporting such cases, the core transactions are extended by the Policy Reserve Rule (PRR) transaction serving for

- reserving addresses and port numbers at NATs (PRR)

2.3.2. Establishing Policy Rules

Both PRR and PER establish a policy rule. The action within the rule is ’reserve’ if set by PRR and ’enable’ if set by PER.

The Policy Reserve Rule (PRR) transaction is used to establish an address reservation on neither side, one side, or both sides of the middlebox, depending on the middlebox configuration. The transaction returns the reserved IP addresses and the optional ranges of port numbers to the agent. No address binding or pinhole configuration is performed at the middlebox. Packet processing at the middlebox remains unchanged.

On pure firewalls, the PRR transaction is successfully processed without any reservation, but the state transition of the MIDCOM protocol engine is exactly the same as on NATs.

On a traditional NAT (see [NAT-TRAD]), only an external address is reserved; on a twice-NAT, an internal and an external address are reserved. The reservation at a NAT is for required resources, such as IP addresses and port numbers, for future use. How the reservation is exactly done depends on the implementation of the NAT. In both cases the reservation concerns either an IP address only or a combination of an IP address with a range of port numbers.

The Policy Enable Rule (PER) transaction is used to establish a policy rule that affects packet processing at the middlebox. Depending on its input parameters, it may make use of the reservation established by a PRR transaction or create a new rule from scratch.

On a NAT, the enable action is interpreted as a bind action establishing bindings between internal and external addresses. At a firewall, the enable action is interpreted as one or more allow actions configuring pinholes. The number of allow actions depends on the parameters of the request and the implementation of the firewall.

On a combined NAT/firewall, the enable action is interpreted as a combination of bind and allow actions.

The PRR transaction and the PER transaction are described in more detail in sections 2.3.8 and 2.3.9 below.

2.3.3. Maintaining Policy Rules and Policy Rule Groups

Each policy rule has a middlebox-unique identifier.

Each policy rule has an owner. Access control to the policy rule is based on ownership (see section 2.1.5). Ownership of a policy rule does not change during lifetime of the policy rule.

Each policy rule has an individual lifetime. If the policy rule lifetime expires, the policy rule will be terminated at the middlebox. Typically, the middlebox indicates termination of a policy rule by an ARE transaction. A policy rule lifetime change (RLC) transaction may extend the lifetime of the policy rule up to the limit specified by the middlebox at session setup. Also an RLC transaction may be used for shortening a policy rule’s lifetime or deleting a policy rule by requesting a lifetime of zero. (Please note that policy rule lifetimes may also be modified by the group lifetime change (GLC) transaction.)

Each policy rule is a member of exactly one policy rule group. Group membership does not change during the lifetime of a policy rule. Selecting the group is part of the transaction establishing the policy rule. This transaction implicitly creates a new group if the agent does not specify one. The new group identifier is chosen by the middlebox. New members are added to an existing group if the agent’s request designates one. A group only exists as long as it has member policy rules. As soon as all policies belonging to the group have reached the ends of their lifetimes, the group does not exist anymore.

Agents can explore the properties and status of all policy rules they are allowed to access by using the Policy Rule Status (PRS) transaction.

2.3.4. Policy Events and Asynchronous Notifications

If a policy rule changes its state or if its remaining lifetime is changed in ways other than being decreased by time, then all agents that can access this policy rule and that participate in an open session with the middlebox are notified by the middlebox. If the state or lifetime change was requested explicitly by a request message, then the middlebox notifies the requesting agent by returning the corresponding reply. All other agents that can access the policy are notified by a Policy Rule Event Notification (REN) message.

Note that a middlebox can serve multiple agents at the same time in different parallel sessions. Between these agents, the sets of policy rules that can be accessed by them may overlap. For example, there might be an agent that authenticates as administrator and that can access all policies of all agents. Or there could be a backup agent running a session in parallel to a main agent and authenticating itself as the same entity as the main agent.

In case of a PER, PRR, or RLC transaction, the requesting agent receives a PER, PRR, or RLC reply, respectively. To all other agents that can access the created, modified, or terminated policy rule (and that participate in an open session with the middlebox) the middlebox sends an REN message carrying the policy rule identifier (PID) and the remaining lifetime of the policy rule.

In case of a rule termination by lifetime truncation or other events not triggered by an agent, then the middlebox sends an REN message to each agent that can access the particular policy rule and that participates in an open session with the middlebox. This ensures that an agent always knows the most recent state of all policy rules it can access.

2.3.5. Address Tuples

Request and reply messages of the PRR, PER, and PRS transactions contain address specifications for IP and transport addresses. These parameters include

- IP version
- IP address
- IP address prefix length
- transport protocol

- port number
- port parity
- port range

Additionally, the request message of PER and the reply message of PRS contain a direction of flow parameter. This direction of flow parameter indicates for UDP and IP the direction of packets traversing the middlebox. For 'inbound', the UDP packets are traversing from outside to inside; for 'outbound', from inside to the outside. In both cases, the packets can traverse the middelbox only uni-directionally. A bi-directional flow is enabled through 'bi-directional' as direction of flow parameter. For TCP, the packet flow is always bi-directional, but the direction of the flow parameter is defined as

- inbound: bi-directional TCP packet flow. First packet, with TCP SYN flag set and ACK flag not set, must arrive at the middlebox at the outside interface.

- outbound: bi-directional TCP packet flow. First packet, with TCP SYN flag set and ACK flag not set, must arrive at the middlebox at the inside interface.

- bi-directional: bi-directional TCP packet flow. First packet, with TCP SYN flag set and ACK flag not set, may arrive at inside or outside interface.

We refer to the set of these parameters as an address tuple. An address tuple specifies either a communication endpoint at an internal or external device or allocated addresses at the middlebox. In this document, we distinguish four kinds of address tuples, as shown in Figure 3.

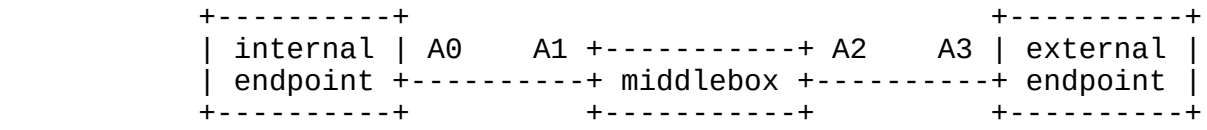

Figure 3: Address tuples A0 - A3

- A0 -- internal endpoint: Address tuple A0 specifies a communication endpoint of a device within -- with respect to the middlebox -- the internal network.

- A1 -- middlebox inside address: Address tuple A1 specifies a virtual communication endpoint at the middlebox within the internal network. A1 is the destination address for packets

passing from the internal endpoint to the middlebox and is the source for packets passing from the middlebox to the internal endpoint.

- A2 -- middlebox outside address: Address tuple A2 specifies a virtual communication endpoint at the middlebox within the external network. A2 is the destination address for packets passing from the external endpoint to the middlebox and is the source for packets passing from the middlebox to the external endpoint.

- A3 -- external endpoint: Address tuple A3 specifies a communication endpoint of a device within -- with respect to the middlebox -- the external network.

For a firewall, the inside and outside endpoints are identical to the corresponding external or internal endpoints, respectively. In this case the installed policy rule sets the same value in A2 as in A0 (A0=A2) and sets the same value in A1 as in A3 (A1=A3).

For a traditional NAT, A2 is given a value different from that of A0, but the NAT binds them. As for the firewall, it is also as it is at a traditional NAT: A1 has the same value as A3.

For a twice-NAT, there are two bindings of address tuples: A1 and A2 are both assigned values by the NAT. The middlebox outside address A2 is bound to the internal endpoint A0, and the middlebox inside address A1 is bound to the external endpoint A3.

2.3.6. Address Parameter Constraints

For transaction parameters belonging to an address tuple, some constraints exist that are common for all messages using them. Therefore, these constraints are summarized in the following and are not repeated again when describing the parameters in the transaction descriptions are presented.

The MIDCOM semantics defined in this document specifies the handling of IPv4 and IPv6 as network protocols, and of TCP and UDP (over IPv4 and IPv6) as transport protocols. The handling of any other transport protocol, e.g., SCTP, is not defined within the semantics but may be supported by concrete protocol specifications.

The IP version parameter has either the value 'IPv4' or 'IPv6'. In a policy rule, the value of the IP version parameter must be the same for address tuples A0 and A1, and for A2 and A3.

The value of the IP address parameter must conform with the specified IP version.

The IP address of an address tuple may be wildcarded. Whether IP address wildcarding is allowed or in which range it is allowed depends on the local policy of the middlebox; see also section 6, "Security Considerations". Wildcarding is specified by the IP address prefix length parameter of an address tuple. In line with the common use of a prefix length, this parameter indicates the number of high significant bits of the IP address that are fixed, while the remaining low significant bits of the IP address are wildcarded.

The value of the transport protocol parameter can be either 'TCP', 'UDP', or 'ANY'. If the transport protocol parameter has the value 'ANY', only IP headers are considered for packet handling in the middlebox -- i.e., the transport header is not considered. The values of the parameters port number, port range, and port parity are irrelevant if the protocol parameter is 'ANY'. In a policy rule, the value of the transport protocol parameter must be the same for all address tuples A0, A1, A2, and A3.

The value of the port number parameter is either zero or a positive integer. A positive integer specifies a concrete UDP or TCP port number. The value zero specifies port wildcarding for the protocol specified by the transport protocol parameter. If the port number parameter has the value zero, then the value of the port range parameter is irrelevant. Depending on the value of the transport protocol parameter, this parameter may truly refer to ports or may refer to an equivalent concept.

The port parity parameter is differently used in the context of policy reserve rules (PRR) and policy enable rules (PER). In the context of a PRR, the value of the parameter may be 'odd', 'even', or 'any'. It specifies the parity of the first (lowest) reserved port number.

In the context of a PER, the port parity parameter indicates to the middlebox whether port numbers allocated at the middlebox should have the same parity as the corresponding internal or external port numbers, respectively. In this context, the parameter has the value 'same' or 'any'. If the value is 'same', then the parity of the port number of A0 must be the same as the parity of the port number of A2, and the parity of the port number of A1 must be the same as the parity of the port number of A3. If the port parity parameter has the value 'any', then there are no constraints on the parity of any port number.

The port range parameter specifies a number of consecutive port numbers. Its value is a positive integer. Like the port number parameter, this parameter defines a set of consecutive port numbers starting with the port number specified by the port number parameter as the lowest port number and having as many elements as specified by the port range parameter. A value of 1 specifies a single port number. The port range parameter must have the same value for each address tuple A0, A1, A2, and A3.

A single policy rule P containing a port range value greater than one is equivalent to a set of policy rules containing a number n of policies P_1, P_2, ..., P_n where n equals the value of the port range parameter. Each policy rule P_1, P_2, ..., P_n has a port range parameter value of 1. Policy rule P_1 contains a set of address tuples A0_1, A1_1, A2_1, and A3_1, each of which contains the first port number of the respective address tuples in P; policy rule P_2 contains a set of address tuples A0_2, A1_2, A2_2, and A3_2, each of which contains the second port number of the respective address tuples in P; and so on.

2.3.7. Interface-specific Policy Rules

Usually agents request policy rules with the knowledge of A0 and A3 only, i.e., the address tuples (see section 2.3.5). But in very special cases, agents may need to select the interfaces to which the requested policy rule is bound. Generally, the middlebox is careful about choosing the right interfaces when reserving or enabling a policy rule, as it has the overall knowledge about its configuration. For agents that want to select the interfaces, optional parameters are included in the Policy Reserve Rule (PRR) and Policy Enable Rule (PER) transactions. These parameters are called

- inside interface: The selected interface at the inside of the middlebox -- i.e., in the private or protected address realm.

- outside interface: The selected interface at the outside of the middlebox -- i.e., in the public address realm.

The Policy Rule Status (PRS) transactions include these optional parameters in its replies when they are supported.

Agents can learn at session startup whether interface-specific policy rules are supported by the middlebox, by checking the middlebox capabilities (see section 2.1.6).

2.3.8. Policy Reserve Rule (PRR)

transaction-name: policy reserve rule

transaction-type: configuration

transaction-compliance: mandatory

request-parameters:

- request identifier: An agent-unique identifier for matching corresponding request and reply at the agent.

- group identifier: A reference to the group of which the policy reserve rule should be a member. As indicated in section 2.3.3, if this value is not supplied, the middlebox assigns a new group for this policy reserve rule.

- service: The requested NAT service of the middlebox. Allowed values are 'traditional' or 'twice'.

- internal IP version: Requested IP version at the inside of the middlebox; see section 2.3.5.

- internal IP address: The IP address of the internal communication endpoint (A0 in Figure 3); see section 2.3.5.

- internal port number: The port number of the internal communication endpoint (A0 in Figure 3); see section 2.3.5.

- inside interface (optional): Interface at the inside of the middlebox; see section 2.3.7.

- external IP version: Requested IP version at the outside of the middlebox; see section 2.3.5.

- outside interface (optional): Interface at the outside of the middlebox; see Section 2.3.7.

- transport protocol: See section 2.3.5.

- port range: The number of consecutive port numbers to be reserved; see section 2.3.5.

- port parity: The requested parity of the first (lowest) port number to be reserved; allowed values for this parameter are 'odd', 'even', and 'any'. See also section 2.3.5.

- policy rule lifetime: A lifetime proposal to the middlebox for the requested policy rule.

reply-parameters (success):

- request identifier: An identifier matching the identifier of the request.

- policy rule identifier: A middlebox-unique policy rule identifier. It is assigned by the middlebox and used as policy rule handle in further policy rule transactions, particularly to refer to the policy reserve rule in a subsequent PER transaction.

- group identifier: A reference to the group of which the policy reserve rule is a member.

- reserved inside IP address: The reserved IPv4 or IPv6 address on the internal side of the middlebox. For an outbound flow, this will be the destination to which the internal endpoint sends its packets (A1 in Figure 3). For an inbound flow, it will be the apparent source address of the packets as forwarded to the internal endpoint (A0 in Figure 3). The middlebox reserves and reports an internal address only in the case where twice-NAT is in effect. Otherwise, an empty value for the addresses indicates that no internal reservation was made. See also Section 2.3.5.

- reserved inside port number: See section 2.3.5.

- reserved outside IP address: The reserved IPv4 or IPv6 address on the external side of the middlebox. For an inbound flow, this will be the destination to which the external endpoint sends its packets (A2 in Figure 4). For an outbound flow, it will be the apparent source address of the packets as forwarded to the external endpoint (A3 in Figure 3). If the middlebox is configured as a pure firewall, an empty value for the addresses indicates that no external reservation was made. See also section 2.3.5.

- reserved outside port number: See section 2.3.5.

- policy rule lifetime: The policy rule lifetime granted by the middlebox, after which the reservation will be revoked if it has not been replaced already by a policy enable rule in a PER transaction.

failure reason:

- agent not authorized for this transaction
- agent not authorized to add members to this group
- lack of IP addresses
- lack of port numbers
- lack of resources
- specified inside/outside interface does not exist
- specified inside/outside interface not available for specified service

notification message type: Policy Rule Event Notification (REN)

semantics:

The agent can use this transaction type to reserve an IP address or a combination of IP address, transport type, port number, and port range at neither side, one side, or both sides of the middlebox as required to support the enabling of a flow. Typically the PRR will be used in scenarios where it is required to perform such a reservation before sufficient parameters for a complete policy enable rule transaction are available. See section 4.2 for an example.

When receiving the request, the middlebox determines how many address (and port) reservations are required based on its configuration. If it provides only packet filter services, it does not perform any reservation and returns empty values for the reserved inside and outside IP addresses and port numbers. If it is configured for twice-NAT, it reserves both inside and outside IP addresses (and an optional range of port numbers) and returns them. Otherwise, it reserves and returns an outside IP address (and an optional range of port numbers) and returns empty values for the reserved inside address and port range.

The A0 parameter (inside IP address version, inside IP address, and inside port number) can be used by the middlebox to determine the correct NAT mapping and thus A2 if necessary. Once a PRR transaction has reserved an outside address (A2) for an internal end point (A0) at the middlebox, the middlebox must ensure that this reserved A2 is available in any subsequent PER and PRR transaction.

For middleboxes supporting interface-specific policy rules, as defined in section 2.3.7, the optional inside and outside interface parameters must both be included in the request, or neither of them should be included. In the presence of these parameters, the middlebox uses the outside interface parameter to

select the interface at which the outside address tuple (outside IP address and port number) is reserved, and the inside interface parameter to select the interface at which the inside address tuple (inside IP address and port number) is reserved. Without the presence of these parameters, the middlebox selects the particular interfaces based on its internal configuration.

If there is a lack of resources, such as available IP addresses, port numbers, or storage for further policy rules, then the reservation fails, and an appropriate failure reply is generated.

If a non-existing policy rule group was specified, or if an existing policy rule group was specified that is not owned by the requesting agent, then no new policy rule is established, and an appropriate failure reply is generated.

In case of success, this transaction creates a new policy reserve rule. If an already existing policy rule group is specified, then the new policy rule becomes a member of it. If no policy group is specified, a new group is created with the new policy rule as its only member. The middlebox generates a middlebox-unique identifier for the new policy rule. The owner of the new policy rule is the authenticated agent that sent the request. The middlebox chooses a lifetime value that is greater than zero and less than or equal to the minimum of the requested value and the maximum lifetime specified by the middlebox at session startup, i.e.,

```
0 <= lt_granted <= MINIMUM(lt_requested, lt_maximum)
```

where
- lt_granted is the lifetime actually granted by the middlebox
- lt_requested is the lifetime the agent requested
- lt_maximum is the maximum lifetime specified at session setup

A middlebox with NAT capability always reserves a middlebox external address tuple (A2) in response to a PRR request. In the special case of a combined twice-NAT/NAT middlebox, the agent can request only NAT service or twice-NAT service by choosing the service parameter 'traditional' or 'twice', respectively. An agent that does not have any preference chooses 'twice'. The 'traditional' value should only be used in order to select traditional NAT service at middleboxes offering both traditional NAT and twice NAT. In the 'twice' case, the combined twice-NAT/NAT middlebox reserves A2 and A1; the 'traditional' case results in a reservation of A2 only. An agent

must always use the PRR transaction for choosing NAT only or twice-NAT service in the special case of a combined twice-NAT/NAT middlebox. A firewall middlebox ignores this parameter.

If the protocol identifier is 'ANY', then the middlebox reserves available inside and/or outside IP address(es) only. The reserved address(es) are returned to the agent. In this case, the request-parameters "port range" and "port parity" as well as reply-parameters "inside port number" and "outside port number", are irrelevant.

If the protocol identifier is 'UDP' or 'TCP', then a combination of an IP address and a consecutive sequence of port numbers, starting with the specified parity, is reserved, on neither side, one side, or both sides of the middlebox, as appropriate. The IP address(es) and the first (lowest) reserved port number(s) of the consecutive sequence are returned to the agent. (This also applies to other protocols supporting ports or the equivalent.)

After a new policy reserve rule is successfully established and the reply message has been sent to the requesting agent, the middlebox checks whether there are other authenticated agents participating in open sessions, which can access the new policy rule. If the middlebox finds one or more of these agents, then it sends a REN message reporting the new policy rule to each of them.

MIDCOM agents use the policy enable rule (PER) transaction to enable policy reserve rules that have been established beforehand by a policy reserve rule (PRR) transaction. See also section 2.3.2.

2.3.9. Policy Enable Rule (PER)

transaction-name: policy enable rule

transaction-type: configuration

transaction-compliance: mandatory

request-parameters:

- request identifier: An agent-unique identifier for matching corresponding request and reply at the agent.

- policy reserve rule identifier: A reference to an already existing policy reserve rule created by a PRR transaction. The reference may be empty, in which case the middlebox must assign any necessary addresses and port numbers within this PER transaction. If it is not empty, then the following request

parameters are irrelevant: group identifier, transport protocol, port range, port parity, internal IP version, external IP version.

- group identifier: A reference to the group of which the policy enable rule should be a member. As indicated in section 2.3.3, if this value is not supplied, the middlebox assigns a new group for this policy reserve rule.

- transport protocol: See section 2.3.5.

- port range: The number of consecutive port numbers to be reserved; see section 2.3.5.

- port parity: The requested parity of the port number(s) to be mapped. Allowed values of this parameter are 'same' and 'any'. See also section 2.3.5.

- direction of flow: This parameter specifies the direction of enabled communication, either 'inbound', 'outbound', or 'bi-directional'.

- internal IP version: Requested IP version at the inside of the middlebox; see section 2.3.5.

- internal IP address: The IP address of the internal communication endpoint (A0 in Figure 3); see section 2.3.5.

- internal port number: The port number of the internal communication endpoint (A0 in Figure 3); see section 2.3.5.

- inside interface (optional): Interface at the inside of the middlebox; see section 2.3.7.

- external IP version: Requested IP version at the outside of the middlebox; see section 2.3.5.

- external IP address: The IP address of the external communication endpoint (A3 in Figure 3); see section 2.3.5.

- external port number: The port number of the external communication endpoint (A3 in Figure 4), see section 2.3.5.

- outside interface (optional): Interface at the outside of the middlebox; see section 2.3.7.

- policy rule lifetime: A lifetime proposal to the middlebox for the requested policy rule.

reply-parameters (success):

- request identifier: An identifier matching the identifier of the request.

- policy rule identifier: A middlebox-unique policy rule identifier. It is assigned by the middlebox and used as policy rule handle in further policy rule transactions. If a policy reserve rule identifier was provided in the request, then the returned policy rule identifier has the same value.

- group identifier: A reference to the group of which the policy enable rule is a member. If a policy reserve rule identifier was provided in the request, then this parameter identifies the group of which the policy reserve rule was a member.

- inside IP address: The IP address provided at the inside of the middlebox (A1 in Figure 3). In case of a twice-NAT, this parameter will be an internal IP address reserved at the inside of the middlebox. In all other cases, this reply-parameter will be identical with the external IP address passed with the request. If the policy reserve rule identifier parameter was supplied in the request and the respective PRR transaction reserved an inside IP address, then the inside IP address provided in the PER response will be the identical value to that returned by the response to the PRR request. See also section 2.3.5.

- inside port number: The internal port number provided at the inside of the middlebox (A1 in Figure 3); see also section 2.3.5.

- outside IP address: The external IP address provided at the outside of the middlebox (A2 in Figure 4). In case of a pure firewall, this parameter will be identical with the internal IP address passed with the request. In all other cases, this reply-parameter will be an external IP address reserved at the outside of the middlebox. See also section 2.3.5.

- outside port number: The external port number provided at the outside of the NAT (A2 in Figure 3); see section 2.3.5..

- policy rule lifetime: The policy rule lifetime granted by the middlebox.

failure reason:

- agent not authorized for this transaction

- agent not authorized to add members to this group
- no such policy reserve rule
- agent not authorized to replace this policy reserve rule
- conflict with already existing policy rule (e.g., the same internal address-port is being mapped to different outside address-port pairs)
- lack of IP addresses
- lack of port numbers
- lack of resources
- no internal IP wildcarding allowed
- no external IP wildcarding allowed
- specified inside/outside interface does not exist
- specified inside/outside interface not available for specified service
- reserved A0 to requested A0 mismatch

notification message type: Policy Rule Event Notification (REN)

semantics:

This transaction can be used by an agent to enable communication between an internal endpoint and an external endpoint independently of the type of middlebox (NAT, NAPT, firewall, NAT-PT, combined devices), for unidirectional or bi-directional traffic.

The agent sends an enable request specifying the endpoints (optionally including wildcards) and the direction of communication (inbound, outbound, bi-directional). The communication endpoints are displayed in Figure 3. The basic operation of the PER transaction can be described by

1. the agent sending A0 and A3 to the middlebox,

2. the middlebox reserving A1 and A2 or using A1 and A2 from a previous PRR transaction,

3. the middlebox enabling packet transfer between A0 and A3 by binding A0-A2 and A1-A3 and/or by opening the corresponding pinholes, both according to the specified direction, and

4. the middlebox returning A1 and A2 to the agent.

In case of a pure packet filtering firewall, the returned address tuples are the same as those in the request: A2=A0 and A1=A3. Each partner uses the other’s real address. In case of a traditional NAT, the internal endpoint may use the real address of the external endpoint (A1=A3), but the external endpoint uses an

address tuple provided by the NAT (A2!=A0). In case of a twice-NAT device, both endpoints use address tuples provided by the NAT for addressing their communication partner (A3!=A1 and A2!=A0).

If a firewall is combined with a NAT or a twice-NAT, the replied address tuples will be the same as for pure traditional NAT or twice-NAT, respectively, but the middlebox will configure its packet filter in addition to the performed NAT bindings. In case of a firewall combined with a traditional NAT, the policy rule may imply more than one enable action for the firewall configuration, as incoming and outgoing packets may use different source-destination pairs.

For middleboxes supporting interface specific policy rules, as defined in Section 2.3.7, the optional inside and outside interface parameters must both be included in the request, or neither of them should be included. In the presence of these parameters, the middlebox uses the outside interface parameter to select the interface at which the outside address tuple (outside IP address and port number) is bound, and the inside interface parameter to select the interface at which the inside address tuple (inside IP address and port number) is bound. Without the presence of these parameters, the middlebox selects the particular interfaces based on its internal configuration.

Checking the Policy Reservation Rule Identifier

If the parameter specifying the policy reservation rule identifier is not empty, then the middlebox checks whether the referenced policy rule exists, whether the agent is authorized to replace this policy rule, and whether this policy rule is a policy reserve rule.

In case of success, this transaction creates a new policy enable rule. If a policy reserve rule was referenced, then the policy reserve rule is terminated without an explicit notification sent to the agent (other than the successful PER reply).

The PRR transaction sets the internal endpoint A0 during the reservation process. In the process of creating a new policy enable rule, the middlebox may check whether the requested A0 is equal to the reserved A0. The middlebox may reject a PER request with a requested A0 not equal to the reserved A0 and must then send an appropriate failure message. Alternatively, the middlebox may change A0 due to the PER request.

The middlebox generates a middlebox-unique identifier for the new policy rule. If a policy reserve rule was referenced, then the identifier of the policy reserve rule is reused.

The owner of the new policy rule is the authenticated agent that sent the request.

Checking the Policy Rule Group Identifier

If no policy reserve rule was specified, then the policy rule group parameter is checked. If a non-existing policy rule group is specified, or if an existing policy rule group is specified that is not owned by the requesting agent, then no new policy rule is established, and an appropriate failure reply is generated.

If an already existing policy rule group is specified, then the new policy rule becomes a member. If no policy group is specified, then a new group is created with the new policy rule as its only member.

If the transport protocol parameter value is 'ANY', then the middlebox enables communication between the specified external IP address and the specified internal IP address. The addresses to be used by the communication partners to address each other are returned to the agent as inside IP address and outside IP address. If the reservation identifier is not empty and if the reservation used the same transport protocol type, then the reserved IP addresses are used.

For the transport protocol parameter values 'UDP' and 'TCP', the middlebox acts analogously as for 'ANY' but also maps ranges of port numbers, keeping the port parity, if requested.

The configuration of the middlebox may fail because of lack of resources, such as available IP addresses, port numbers, or storage for further policy rules. It may also fail because of a conflict with an established policy rule. In case of a conflict, the first-come first-served mechanism is applied. Existing policy rules remain unchanged and arriving new ones are rejected. However, in case of a non-conflicting overlap of policy rules (including identical policy rules), all policy rules are accepted.

The middlebox chooses a lifetime value that is greater than zero and less than or equal to the minimum of the requested value and the maximum lifetime specified by the middlebox at session startup, i.e.,

0 <= lt_granted <= MINIMUM(lt_requested, lt_maximum)

where
- lt_granted is the lifetime actually granted by the middlebox
- lt_requested is the lifetime the agent requested
- lt_maximum is the maximum lifetime specified at session setup

In each case of failure, an appropriate failure reply is generated. The policy reserve rule that is referenced in the PER transaction is not affected in case of a failure within the PER transaction -- i.e., the policy reserve rule remains.

After a new policy enable rule is successfully established and the reply message has been sent to the requesting agent, the middlebox checks whether there are other authenticated agents participating in open sessions that can access the new policy rule. If the middlebox finds one or more of these agents, then it sends a REN message reporting the new policy rule to each of them.

2.3.10. Policy Rule Lifetime Change (RLC)

transaction-name: policy rule lifetime change

transaction-type: configuration

transaction-compliance: mandatory

request-parameters:

- request identifier: An agent-unique identifier for matching corresponding request and reply at the agent.
- policy rule identifier: Identifying the policy rule for which the lifetime is requested to be changed. This may identify either a policy reserve rule or a policy enable rule.
- policy rule lifetime: The new lifetime proposal for the policy rule.

reply-parameters (success):

- request identifier: An identifier matching the identifier of the request.
- policy rule lifetime: The remaining policy rule lifetime granted by the middlebox.

failure reason:

- agent not authorized for this transaction
- agent not authorized to change lifetime of this policy rule
- no such policy rule
- lifetime cannot be extended

notification message type: Policy Rule Event Notification (REN)

semantics:

The agent can use this transaction type to request the extension of an established policy rule's lifetime, the shortening of the lifetime, or policy rule termination. Policy rule termination is requested by suggesting a new policy rule lifetime of zero.

The middlebox first checks whether the specified policy rule exists and whether the agent is authorized to access this policy rule. If one of the checks fails, an appropriate failure reply is generated. If the requested lifetime is longer than the current one, the middlebox also checks whether the lifetime of the policy rule may be extended and generates an appropriate failure message if it may not.

A failure reply implies that the new lifetime was not accepted, and the policy rule remains unchanged. A success reply is generated by the middlebox if the lifetime of the policy rule was changed in any way.

The success reply contains the new lifetime of the policy rule. The middlebox chooses a lifetime value that is greater than zero and less than or equal to the minimum of the requested value and the maximum lifetime specified by the middlebox at session startup, i.e.,

$$0 <= lt\_granted <= MINIMUM(lt\_requested, lt\_maximum)$$

whereas

- lt_granted is the lifetime actually granted by the middlebox
- lt_requested is the lifetime the agent requested
- lt_maximum is the maximum lifetime specified at session setup

After sending a success reply with a lifetime of zero, the middlebox will consider the policy rule non-existent. Any further transaction on this policy rule results in a negative reply, indicating that this policy rule does not exist anymore.

Note that policy rule lifetime may also be changed by the Group Lifetime Change (GLC) transaction, if applied to the group of which the policy rule is a member.

After the remaining policy rule lifetime was successfully changed and the reply message has been sent to the requesting agent, the middlebox checks whether there are other authenticated agents participating in open sessions that can access the policy rule. If the middlebox finds one or more of these agents, then it sends a REN message reporting the new remaining policy rule lifetime to each of them.

2.3.11. Policy Rule List (PRL)

transaction-name: policy rule list

transaction-type: monitoring

transaction-compliance: mandatory

request-parameters:

- request identifier: An agent-unique identifier for matching corresponding request and reply at the agent.

reply-parameters (success):

- request identifier: An identifier matching the identifier of the request.

- policy list: List of policy rule identifiers of all policy rules that the agent can access.

failure reason:

- transaction not supported
- agent not authorized for this transaction

semantics:

The agent can use this transaction type to list all policies that it can access. Usually, the agent has this information already, but in special cases (for example, after an agent fail-over) or for special agents (for example, an administrating agent that can access all policies) this transaction can be helpful.

The middlebox first checks whether the agent is authorized to request this transaction. If the check fails, an appropriate failure reply is generated. Otherwise a list of all policies the agent can access is returned indicating the identifier and the owner of each policy.

This transaction does not have any effect on the policy rule state.

2.3.12. Policy Rule Status (PRS)

transaction-name: policy rule status

transaction-type: monitoring

transaction-compliance: mandatory

request-parameters:

- request identifier: An agent-unique identifier for matching corresponding request and reply at the agent.
- policy rule identifier: The middlebox-unique policy rule identifier.

reply-parameters (success):

- request identifier: An identifier matching the identifier of the request.
- policy rule owner: An identifier of the agent owning this policy rule.
- group identifier: A reference to the group of which the policy rule is a member.
- policy rule action: This parameter has either the value ’reserve’ or the value ’enable’.

- transport protocol: Identifies the protocol for which a reservation is requested; see section 2.3.5.

- port range: The number of consecutive port numbers; see section 2.3.5.

- direction: The direction of the communication enabled by the middlebox. Applicable only to policy enable rules.

- internal IP address version: The version of the internal IP address (IP version of A0 in Figure 3).

- external IP address version: The version of the external IP address (IP version of A3 in Figure 3).

- internal IP address: The IP address of the internal communication endpoint (A0 in Figure 3); see section 2.3.5.

- internal port number: The port number of the internal communication endpoint (A0 in Figure 3); see section 2.3.5.

- external IP address: The IP address of the external communication endpoint (A3 in Figure 3); see section 2.3.5.

- external port number: The port number of the external communication endpoint (A3 in Figure 3); see section 2.3.5.

- inside interface (optional): The inside interface at the middlebox; see section 2.3.7.

- inside IP address: The internal IP address provided at the inside of the NAT (A1 in Figure 3); see section 2.3.5.

- inside port number: The internal port number provided at the inside of the NAT (A1 in Figure 3); see section 2.3.5.

- outside interface (optional): The outside interface at the middlebox; see section 2.3.7.

- outside IP address: The external IP address provided at the outside of the NAT (A2 in Figure 3); see section 2.3.5.

- outside port number: The external port number provided at the outside of the NAT (A2 in Figure 3); see section 2.3.5.

- port parity: The parity of the allocated ports.

- service: The selected service in the case of mixed traditional and twice-NAT middlebox (see section 2.3.8).

- policy rule lifetime: The remaining lifetime of the policy rule.

failure reason:

- transaction not supported
- agent not authorized for this transaction
- no such policy rule
- agent not authorized to access this policy rule

semantics:

The agent can use this transaction type to list all properties of a policy rule. Usually, the agent has this information already, but in special cases (for example, after an agent fail-over) or for special agents (for example, an administrating agent that can access all policy rules) this transaction can be helpful.

The middlebox first checks whether the specified policy rule exists and whether the agent is authorized to access this group. If one of the checks fails, an appropriate failure reply is generated. Otherwise all properties of the policy rule are returned to the agent. Some of the returned parameters may be irrelevant, depending on the policy rule action ('reserve' or 'enable') and depending on other parameters -- for example, the protocol identifier.

This transaction does not have any effect on the policy rule state.

2.3.13. Asynchronous Policy Rule Event (ARE)

transaction-name: asynchronous policy rule event

transaction-type: notification

transaction-compliance: mandatory

notification message type: Policy Rule Event Notification (REN)

semantics:

The middlebox may decide at any point in time to terminate a policy rule. This transaction is triggered most frequently by lifetime expiration of the policy rule. Among other events that

may cause this transaction are changes in the policy rule decision point.

The middlebox sends an REN message to all agents that participate in an open session with the middlebox and that are authorized to access the policy rule. The notification is sent to the agents before the middlebox changes the policy rule's lifetime. The change of lifetime may be triggered by any other authorized agent and results in shortening (lt_new < lt_existing), extending (lt_new > lt_existing), or terminating the policy rule (lt_new = 0).

The ARE transaction corresponds to the REN message handling described in section 2.3.4 for multiple agents.

2.3.14. Policy Rule State Machine

The state machine for the policy rule transactions is shown in Figure 4 with all possible state transitions. The used transaction abbreviations may be found in the headings of the particular transaction section.

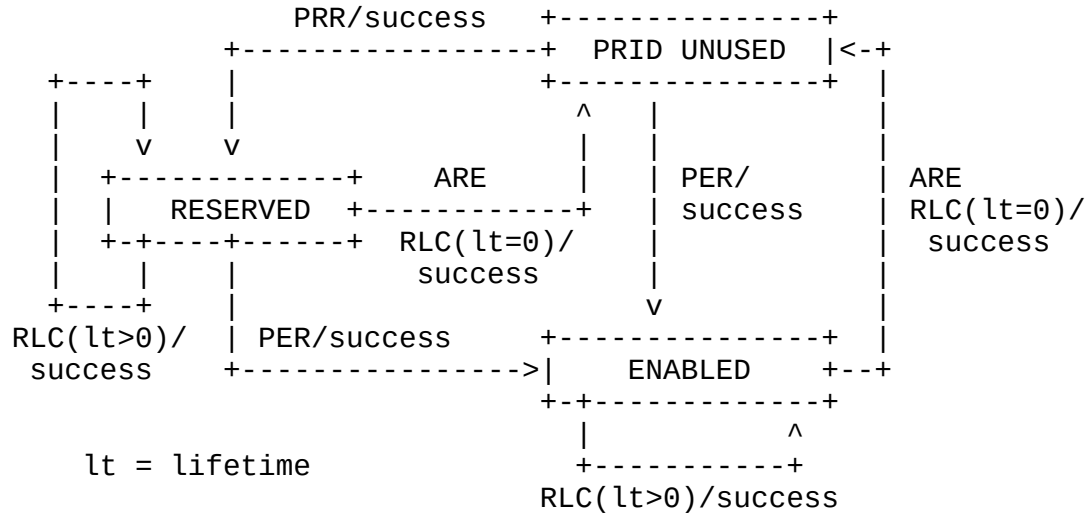

Figure 4: Policy Rule State Machine

This state machine exists per policy rule identifier (PRID). Initially all policy rules are in state PRID UNUSED, which means that the policy rule does not exist or is not active. After returning to state PRID UNUSED, the policy rule identifier is no longer bound to an existing policy rule and may be reused by the middlebox.

A successful PRR transaction causes a transition from the initial state PRID UNUSED to the state RESERVED, where an address reservation is established. From there, state ENABLED can be entered by a PER transaction. This transaction can also be used for entering state ENABLED directly from state PRID UNUSED without a reservation. In state ENABLED the requested communication between the internal and the external endpoint is enabled.

The states RESERVED and ENABLED can be maintained by successful RLC transactions with a requested lifetime greater than 0. Transitions from both of these states back to state PRID UNUSED can be caused by an ARE transaction or by a successful RLC transaction with a lifetime parameter of 0.

A failed request transactions does not change state at the middlebox.

Note that transitions initiated by RLC transactions may also be initiated by GLC transactions.

2.4. Policy Rule Group Transactions

This section describes the semantics for transactions on groups of policy rules. These transactions are specified as follows:

- Group Lifetime Change (GLC)
- Group List (GL)
- Group Status (GS)

All are request transactions initiated by the agent. GLC is a convenience transaction. GL and GS are monitoring transactions that do not have any effect on the group state machine.

2.4.1. Overview

A policy rule group has only one attribute: the list of its members. All member policies of a single group must be owned by the same authenticated agent. Therefore, an implicit property of a group is its owner, which is the owner of the member policy rules.

A group is implicitly created when its first member policy rule is established. A group is implicitly terminated when the last remaining member policy rule is terminated. Consequently, the lifetime of a group is the maximum of the lifetimes of all member policy rules.

A group has a middlebox-unique identifier.

Group transactions are declared as 'optional' by their respective compliance entry in section 3. However, they provide some functionalities, such as convenience for the agent in sending only one request instead of several, that is not available if only mandatory transactions are available.

The Group Lifetime Change (GLC) transaction is equivalent to simultaneously performed Policy Rule Lifetime Change (RLC) transactions on all members of the group. The result of a successful GLC transaction is that all member policy rules have the same lifetime. As with the RLC transaction, the GLC transaction can be used to delete all member policy rules by requesting a lifetime of zero.

The monitoring transactions Group List (GL) and Group Status (GS) can be used by the agent to explore the state of the middlebox and to explore its access rights. The GL transaction lists all groups that the agent may access, including groups owned by other agents. The GS transaction reports the status on an individual group and lists all policy rules of this group by their policy rule identifiers. The agent can explore the state of the individual policy rules by using the policy rule identifiers in a policy rule status (PRS) transaction (see section 2.3.12).

The GL and GS transactions are particularly helpful in case of an agent fail-over. The agent taking over the role of a failed one can use these transactions retrieve whichever policies have been established by the failed agent.

Notifications on group events are generated analogously to policy rule events. To notify agents about group events, the Policy Rule Group Event Notification (GEN) message type is used. GEN messages contain an agent-unique notification identifier, the policy rule group identifier, and the remaining lifetime of the group.

2.4.2. Group Lifetime Change (GLC)

transaction-name: group lifetime change

transaction-type: convenience

transaction-compliance: optional

request-parameters:

- request identifier: An agent-unique identifier for matching corresponding request and reply at the agent.

- group identifier: A reference to the group for which the lifetime is requested to be changed.

- group lifetime: The new lifetime proposal for the group.

reply-parameters (success):

- request identifier: An identifier matching the identifier of the request.

- group lifetime: The group lifetime granted by the middlebox.

failure reason:

- transaction not supported
- agent not authorized for this transaction
- agent not authorized to change lifetime of this group
- no such group
- lifetime cannot be extended

notification message type: Policy Rule Group Event Notification (GEN)

semantics:

The agent can use this transaction type to request an extension of the lifetime of all members of a policy rule group, to request shortening the lifetime of all members, or to request termination of all member policies (which implies termination of the group). Termination is requested by suggesting a new group lifetime of zero.

The middlebox first checks whether the specified group exists and whether the agent is authorized to access this group. If one of the checks fails, an appropriate failure reply is generated. If the requested lifetime is longer than the current one, the middlebox also checks whether the lifetime of the group may be extended and generates an appropriate failure message if it may not.

A failure reply implies that the lifetime of the group remains unchanged. A success reply is generated by the middlebox if the lifetime of the group was changed in any way.

The success reply contains the new common lifetime of all member policy rules of the group. The middlebox chooses the new lifetime less than or equal to the minimum of the requested lifetime and the maximum lifetime that the middlebox specified at session setup along with its other capabilities, i.e.,

0 <= lt_granted <= MINIMUM(lt_requested, lt_maximum)

where
- lt_granted is the lifetime actually granted by the middlebox
- lt_requested is the lifetime the agent requested
- lt_maximum is the maximum lifetime specified at session setup

After sending a success reply with a lifetime of zero, the middlebox will terminate the member policy rules without any further notification to the agent, and will consider the group and all of its members non-existent. Any further transaction on this policy rule group or on any of its members results in a negative reply, indicating that this group or policy rule, respectively, does not exist anymore.

After the remaining policy rule group lifetime is successfully changed and the reply message has been sent to the requesting agent, the middlebox checks whether there are other authenticated agents participating in open sessions that can access the policy rule group. If the middlebox finds one or more of these agents, it sends a GEN message reporting the new remaining policy rule group lifetime to each of them.

2.4.3. Group List (GL)

transaction-name: group list

transaction-type: monitoring

transaction-compliance: optional

request-parameters:

- request identifier: An agent-unique identifier for matching corresponding request and reply at the agent.

reply-parameters (success):

- request identifier: An identifier matching the identifier of the request.

- group list: List of all groups that the agent can access. For each listed group, the identifier and the owner are indicated.

failure reason:

- transaction not supported
- agent not authorized for this transaction

semantics:

The agent can use this transaction type to list all groups that it can access. Usually, the agent has this information already, but in special cases (for example, after an agent fail-over) or for special agents (for example, an administrating agent that can access all groups) this transaction can be helpful.

The middlebox first checks whether the agent is authorized to request this transaction. If the check fails, an appropriate failure reply is generated. Otherwise a list of all groups the agent can access is returned indicating the identifier and the owner of each group.

This transaction does not have any effect on the group state.

2.4.4. Group Status (GS)

transaction-name: group status

transaction-type: monitoring

transaction-compliance: optional

request-parameters:

- request identifier: An agent-unique identifier for matching corresponding request and reply at the agent.
- group identifier: A reference to the group for which status information is requested.

reply-parameters (success):

- request identifier: An identifier matching the identifier of the request.
- group owner: An identifier of the agent owning this policy rule group.
- group lifetime: The remaining lifetime of the group. This is the maximum of the remaining lifetime of all members, policy rules.

- member list: List of all policy rules that are members of the group. The policy rules are specified by their middlebox-unique policy rule identifier.

failure reason:

- transaction not supported
- agent not authorized for this transaction
- no such group
- agent not authorized to list members of this group

semantics:

The agent can use this transaction type to list all member policy rules of a group. Usually, the agent has this information already, but in special cases (for example, after an agent fail-over) or for special agents (for example, an administrating agent that can access all groups) this transaction can be helpful.

The middlebox first checks whether the specified group exists and whether the agent is authorized to access this group. If one of the checks fails, an appropriate failure reply is generated. Otherwise a list of all group members is returned indicating the identifier of each group.

This transaction does not have any effect on the group state.

3. Conformance Statements

A protocol definition complies with the semantics defined in section 2 if the protocol specification includes all specified transactions with all their mandatory parameters. However, concrete implementations of the protocol may support only some of the optional transactions, not all of them. Which transactions are required for compliance is different for agent and middlebox.

This section contains conformance statements for MIDCOM protocol implementations related to the semantics. Conformance is specified differently for agents and middleboxes. These conformance statements will probably be extended by a concrete protocol specification. However, such an extension is expected to extend the statements below in such a way that all of them still hold.

The following list shows the transaction-compliance property of all transactions as specified in the previous section:

- Session Control Transactions
  - Session Establishment (SE) mandatory
  - Session Termination (ST) mandatory
  - Asynchronous Session Termination (AST) mandatory

- Policy Rule Transactions
  - Policy Reserve Rule (PRR) mandatory
  - Policy Enable Rule (PER) mandatory
  - Policy Rule Lifetime Change (RLC) mandatory
  - Policy Rule List (PRL) mandatory
  - Policy Rule Status (PRS) mandatory
  - Asynchronous Policy Rule Event (ARE) mandatory

- Policy Rule Group Transactions
  - Group Lifetime Change (GLC) optional
  - Group List (GL) optional
  - Group Status (GS) optional

3.1. General Implementation Conformance

A compliant implementation of a MIDCOM protocol must support all mandatory transactions.

A compliant implementation of a MIDCOM protocol may support none, one, or more of the following transactions: GLC, GL, GS.

A compliant implementation may extend the protocol semantics by further transactions.

A compliant implementation of a MIDCOM protocol must support all mandatory parameters of each transaction concerning the information contained. The set of parameters can be redefined per transaction as long as the contained information is maintained.

A compliant implementation of a MIDCOM protocol may support the use of interface-specific policy rules. Either both or neither of the optional inside and outside interface parameters in PRR, PER, and PRS must be included when interface-specific policy rules are supported.

A compliant implementation may extend the list of parameters of transactions.

A compliant implementation may replace a single transaction by a set of more fine-grained transactions. In such a case, it must be ensured that requirement 2.1.4 (deterministic behavior) and requirement 2.1.5 (known and stable state) of [MDC-REQ] are still met. When a single transaction is replaced by a set of multiple fine-grained transactions, this set must be equivalent to a single

transaction. Furthermore, this set of transactions must further meet the atomicity requirement stated in section 2.1.3.

3.2. Middlebox Conformance

A middlebox implementation of a MIDCOM protocol supports a request transaction if it is able to receive and process all possible correct message instances of the particular request transaction and if it generates a correct reply for any correct request it receives.

A middlebox implementation of a MIDCOM protocol supports an asynchronous transaction if it is able to generate the corresponding notification message properly.

A compliant middlebox implementation of a MIDCOM protocol must inform the agent about the list of supported transactions within the SE transaction.

3.3. Agent Conformance

An agent implementation of a MIDCOM protocol supports a request transaction if it can generate the corresponding request message properly and if it can receive and process all possible correct replies to the particular request.

An agent implementation of a MIDCOM protocol supports an asynchronous transaction if it can receive and process all possible correct message instances of the particular transaction.

A compliant agent implementation of a MIDCOM protocol must not use any optional transaction that is not supported by the middlebox. The middlebox informs the agent about the list of supported transactions within the SE transaction.

4. Transaction Usage Examples

This section gives two usage examples of the transactions specified in Section 2. The first shows how an agent can explore all policy rules and policy rule groups that it may access at a middlebox. The second example shows the configuration of a middlebox in combination with the setup of a voice over IP session with the Session Initiation Protocol (SIP) [RFC3261].

4.1. Exploring Policy Rules and Policy Rule Groups

This example assumes an already established session. It shows how an agent can find out

- which groups it may access and who owns these groups,
- the status and member list of all accessible groups, and
- the status and properties of all accessible policy rules.

If there is just a single session, these actions are not needed, because the middlebox informs the agent about each state transition of any policy rule or policy rule group. However, after the disruption of a session or after an intentional session termination, the agent might want to re-establish the session and explore which of the groups and policy rules it established are still in place.

Also, an agent system may fail and another one may take over. Then the new agent system needs to find out what has already been configured by the failing system and what still needs to be done.

A third situation where exploring policy rules and groups is useful is the case of an agent with 'administrator' authorization. This agent may access and modify any policy rule or group created by any other agent.

All agents will probably start their exploration with the Group List (GL) transaction, as shown in Figure 5. On this request, the middlebox returns a list of pairs, each containing an agent identifier and a group identifier (GID). The agent is informed which of its own groups and which other agents' groups it may access.

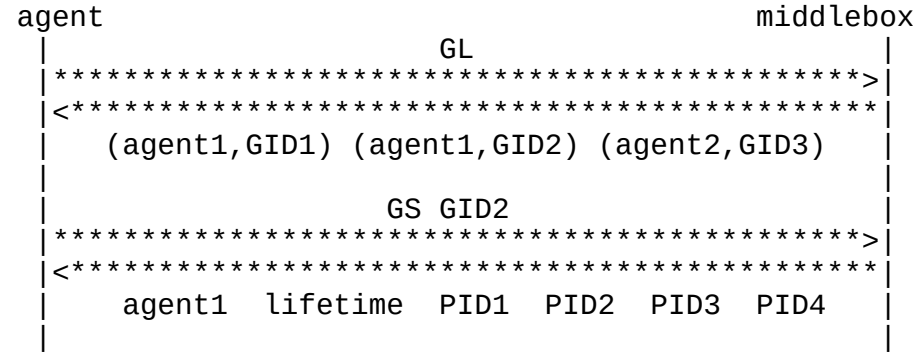

Figure 5: Using the GL and the GS transaction

In Figure 5, three groups are accessible to the agent, and the agent retrieves information about the second group by using the Group Status (GS) transaction. It receives the owner of the group, the remaining lifetime, and the list of member policy rules, in this case containing four policy rule identifiers (PIDs).

In the following, the agent explores these four policy rules. The example assumes that the middlebox is a traditional NAPT. Figure 6 shows the exploration of the first policy rule. In reply to a Policy Rule Status (PRS) transaction, the middlebox always returns the following list of parameters:

- policy rule owner
- group identifier
- policy rule action (reserve or enable)
- protocol type
- port range
- direction
- internal IP address
- internal port number
- external address
- external port number
- middlebox inside IP address
- middlebox inside port number
- middlebox outside IP address
- middlebox outside port number
- IP address versions (not printed)
- middlebox service (not printed)
- inside and outside interface (optional, not printed)

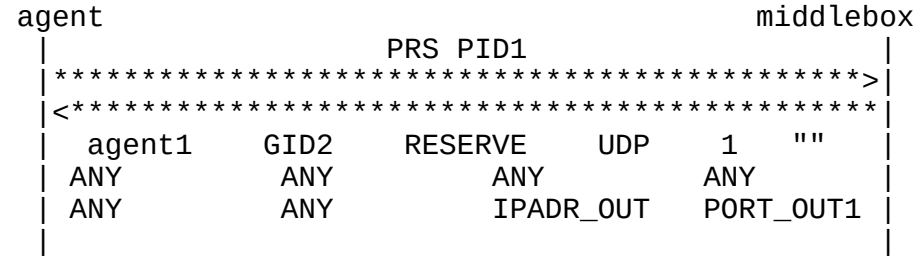

Figure 6: Status report for an outside reservation

The 'ANY' parameter printed in Figure 6 is used as a placeholder in policy rules status replies for policy reserve rules. The policy rule with PID1 is a policy reserve rule for UDP traffic at the outside of the middlebox. Since this is a reserve rule, direction is empty. As there is no internal or external address involved yet, these four fields are wildcarded in the reply. The same holds for the inside middlebox address and port number. The only address information given by the reply is the reserved outside IP address of the middlebox (IPADDR_OUT) and the corresponding port number (PORT_OUT1). Note that IPADR_OUT and PORT_OUT1 may not be wildcarded, as the reserve action does not support this.

Applying PRS to PID2 (Figure 7) shows that the second policy rule is a policy enable rule for inbound UDP packets. The internal destination is fixed concerning IP address, protocol, and port number, but for the external source, the port number is wildcarded. The outside IP address and port number of the middlebox are what the external sender needs to use as destination in the original packet it sends. At the middlebox, the destination address is replaced with the internal address of the final receiver. During address translation, the source IP address and the source port numbers of the packets remain unchanged. This is indicated by the inside address, which is identical to the external address.

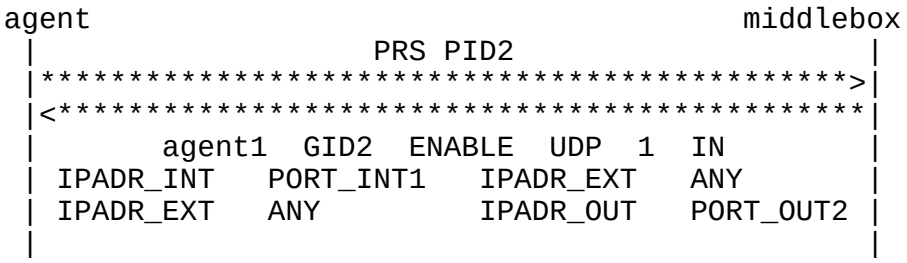

Figure 7: Status report for enabled inbound packets

For traditional NATs, the identity of the inside IP address and port number with the external IP address and port number always holds (A1=A3 in Figure 3). For a pure firewall, the outside IP address and port number are always identical with the internal IP address and port number (A0=A2 in Figure 3).

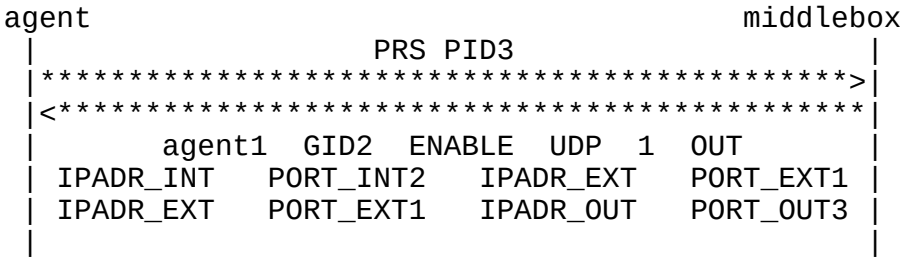

Figure 8: Status report for enabled outbound packets

Figure 8 shows enabled outbound UDP communication between the same host. Here all port numbers are known. Since again A1=A3, the internal sender uses the external IP address and port number as destination in the original packets. At the firewall, the internal source IP address and port number are replaced by the shown outside IP address and port number of the middlebox.

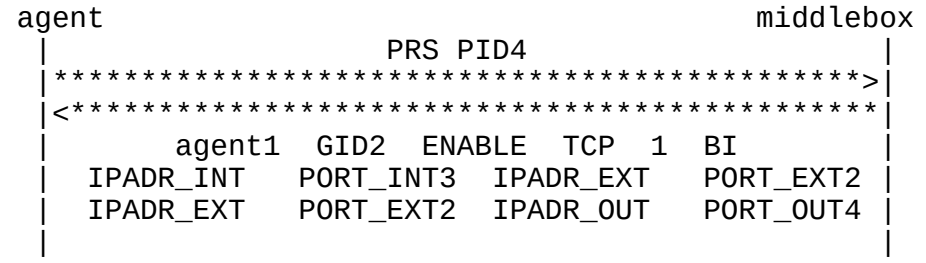

Figure 9: Status report for bi-directional TCP traffic

Finally, Figure 9 shows the status report for enabled bi-directional TCP traffic. Note that, still, A1=A3. For outbound packets, only the source IP address and port number are replaced at the middlebox, and for inbound packets, only the destination IP address and port number are replaced.

4.2. Enabling a SIP-Signaled Call

This elaborated transaction usage example shows the interaction between a SIP proxy and a middlebox. The middlebox itself is a traditional Network Address and Port Translator (NAPT), and two SIP user agents communicate with each other via the SIP proxy and NAPT, as shown in Figure 10. The MIDCOM agent is co-located with the SIP proxy, and the MIDCOM server is at the middlebox. Thus, the MIDCOM protocol runs between the SIP proxy and middlebox.

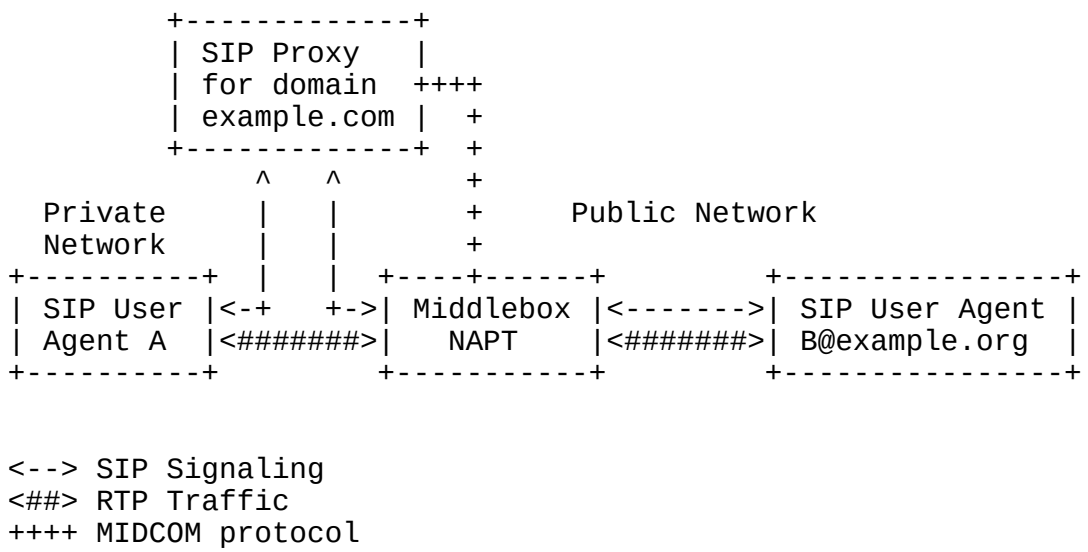

Figure 10: Example of a SIP Scenario

For the sequence charts below, we make these assumptions:

- The NAPT is statically configured to forward SIP signaling from the outside to the SIP proxy server -- i.e., traffic to the NAPT's external IP address and port 5060 is forwarded to the internal SIP proxy.

- The SIP user agent A, located inside the private network, is registered at the SIP proxy with its private IP address.

- User A knows the general SIP URL of user B. The URL is B@example.org. However, the concrete URL of the SIP User Agent B, which user B currently uses, is not known.

- The RTP paths are configured, but not the RTCP paths.

- The middlebox and the SIP server share an established MIDCOM session.

- Some parameters are omitted, such as the request identifier (RID).

Furthermore, the following abbreviations are used:

- IP_AI: Internal IP address of user agent A
- P_AI: Internal port number of user agent A to receive RTP data
- P_AE: External mapped port number of user agent A
- IP_AE: External IP address of the middlebox
- IP_B: IP address of user agent B
- P_B: Port number of user agent B to receive RTP data
- GID: Group identifier
- PID: Policy rule identifier

The abbreviations of the MIDCOM transactions can be found in the particular section headings.

In our example, user A tries to call user B. The user agent A sends an INVITE SIP message to the SIP proxy server (see Figure 10). The SDP part of the particular SIP message relevant for the middlebox configuration is shown in the sequence chart as follows:

```
SDP: m=..P_AI..
     c=IP_AI
```

where the m tag is the media tag that contains the receiving UDP port number, and the c tag contains the IP address of the terminal receiving the media stream.

The INVITE message forwarded to user agent B must contain a public IP address and a port number to which user agent B can send its RTP media stream. The SIP proxy requests a policy enable rule at the middlebox with a PER request with the wildcarded IP address and port number of user agent B. As neither the IP address nor port numbers of user agent B are known at this point, the address of user agent B must be wildcarded. The wildcarded IP address and port number enables the 'early media' capability but results in some insecurity, as any outside host can reach user agent A on the enabled port number through the middlebox.

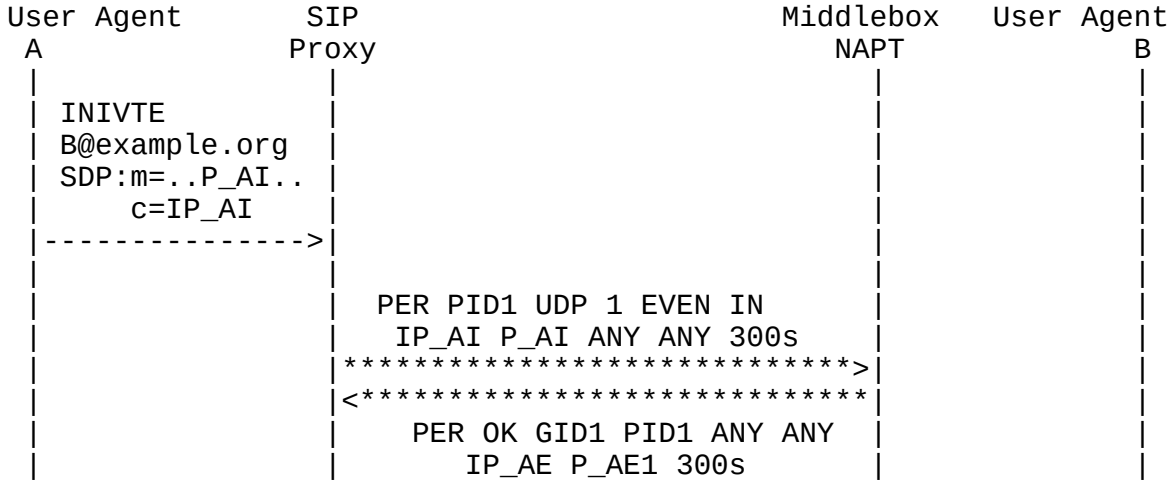

Figure 11: PER with wildcard address and port number

A successful PER reply, as shown in Figure 11, results in an NAT binding at the middlebox. This binding enables UDP traffic from any host outside user agent A's private network to reach user agent A. So user agent B could start sending traffic immediately after receiving the INVITE message, as could any other host -- even hosts that are not intended to participate, such as any malicious host.

If the middlebox does not support or does not permit IP address wildcarding for security reasons, the PER request will be rejected with an appropriate failure reason, like 'IP wildcarding not supported'. Nevertheless, the SIP proxy server needs an outside IP address and port number at the middlebox (the NAPT) in order to forward the SIP INVITE message.

If the IP address of user agent B is still not known (it will be sent by user agent B in the SIP reply message) and IP address wildcarding is not permitted, the SIP proxy server uses the PRR transaction.

By using the PRR request, the SIP proxy requests an outside IP address and port number (see Figure 12) without already establishing a NAT binding or pin hole. The PRR request contains the service parameter 'tw' -- i.e., the MIDCOM agent chooses the default value. In this configuration, with NAPT and without a twice NAT, only an outside address is reserved. In the SDP payload of the INVITE message, the SIP proxy server replaces the IP address and port number of user agent A with the reserved IP address and port from PRR reply (see Figure 12). The SIP INVITE message is forwarded to user agent B with a modified SDP body containing the outside address and port number, to which user agent B will send its RTP media stream.

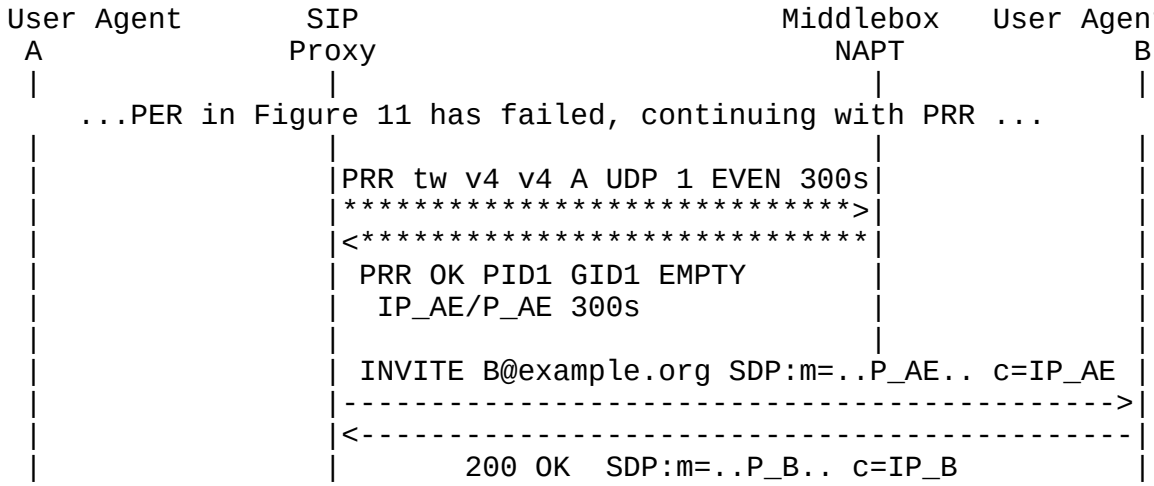

Figure 12: Address reservation with PRR transaction

This SIP '200 OK' reply contains the IP address and port number at which user agent B will receive a media stream. The IP address is assumed to be equal to the IP address from which user agent B will send its media stream.

Now, the SIP proxy server has sufficient information for establishing the complete NAT binding with a policy enable rule (PER) transaction, i.e., the UDP/RTP data of the call can flow from user agent B to user agent A. The PER transaction references the reservation by passing the PID of the PRR (PID1).

For the opposite direction, UDP/RTP data from user agent A to B has to be enabled also. This is done by a second PER transaction with all the necessary parameters (see Figure 13). The request message contains the group identifier (GID1) the middlebox has assigned in the first PER transaction. Therefore, both policy rules have become

members of the same group. After having enabled both UDP/RTP streams, the SIP proxy can forward the '200 OK' SIP message to user agent A to indicate that the telephone call can start.

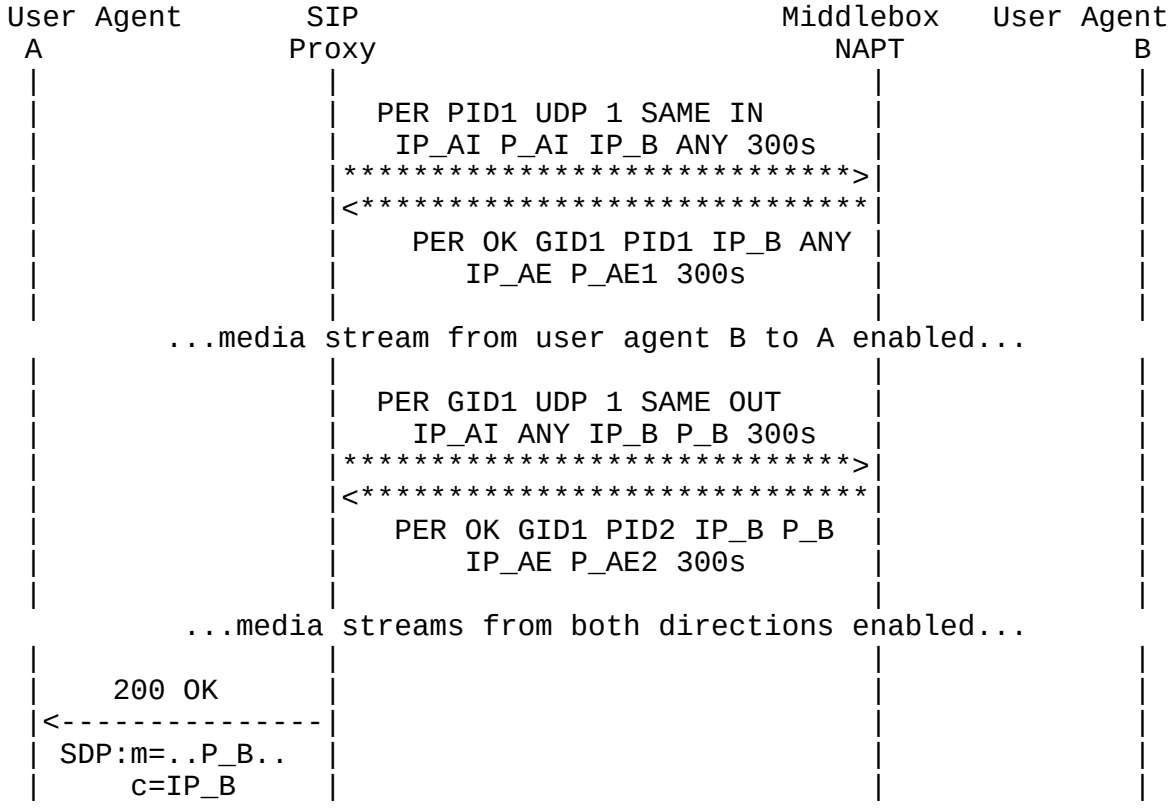

Figure 13: Policy rule establishment for UDP flows

User agent B decides to terminate the call and sends its 'BYE' SIP message to user agent A. The SIP proxy forwards all SIP messages and terminates the group afterwards, using a group lifetime change (GLC) transaction with a requested remaining lifetime of 0 seconds (see Figure 14). Termination of the group includes terminating all member policy rules.

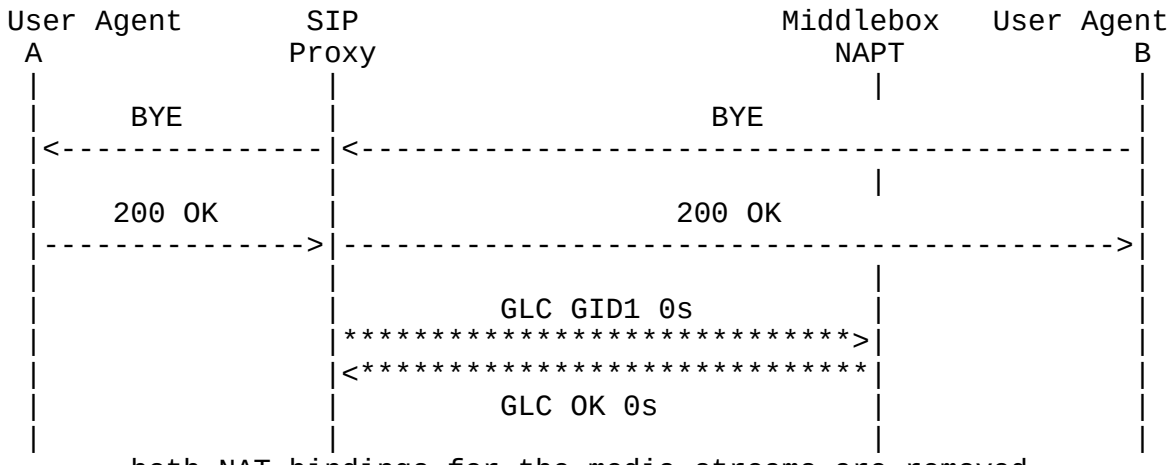

Figure 14: Termination of policy rule groups

5. Compliance with MIDCOM Requirements

This section explains the compliance of the specified semantics with the MIDCOM requirements. It is structured according to [MDC-REQ]:

- Compliance with Protocol Machinery Requirements (section 5.1)
- Compliance with Protocol Semantics Requirements (section 5.2)
- Compliance with Security Requirements (section 5.3)

The requirements are referred to with the number of the section in which they are defined: "requirement x.y.z" refers to the requirement specified in section x.y.z of [MDC-REQ].

5.1. Protocol Machinery Requirements

5.1.1. Authorized Association

The specified semantics enables a MIDCOM agent to establish an authorized association between itself and the middlebox. The agent identifies itself by the authentication mechanism of the Session Establishment transaction described in section 2.2.1. Based on this authentication, the middlebox can determine whether or not the agent will be permitted to request a service. Thus, requirement 2.1.1 is met.

5.1.2. Agent Connects to Multiple Middleboxes

As specified in section 2.2, the MIDCOM protocol allows the agent to communicate with more than one middlebox simultaneously. The selection of a mechanism for separating different sessions is left to the concrete protocol definition. It must provide a clear mapping of protocol messages to open sessions. Then requirement 2.1.2 is met.

5.1.3. Multiple Agents Connect to same Middlebox

As specified in section 2.2, the MIDCOM protocol allows the middlebox to communicate with more than one agent simultaneously. The selection of a mechanism for separating different sessions is left to the concrete protocol definition. It must provide a clear mapping of protocol messages to open sessions. Then requirement 2.1.3 is met.

5.1.4. Deterministic Behavior

Section 2.1.2 states that the processing of a request of an agent may not be interrupted by any request of the same or another agent. This provides atomicity among request transactions and avoids race conditions resulting in unpredictable behavior by the middlebox.

The behavior of the middlebox can only be predictable in the view of its administrators. In the view of an agent, the middlebox behavior is unpredictable, as the administrator can, for example, modify the authorization of the agent at any time without the agent being able to observe this change. Consequently, the behavior of the middlebox is not necessarily deterministic from the point of view of any agent.

As predictability of the middlebox behavior is given for its administrator, requirement 2.1.4 is met.

5.1.5. Known and Stable State

Section 2.1 states that request transactions are atomic with respect to each other and from the point of view of an agent. All transactions are clearly defined as state transitions that either leave the current stable, well-defined state and enter a new stable, well-defined one or that remain in the current stable, well-defined state. Section 2.1 clearly demands that intermediate states are not stable and are not reported to any agent.

Furthermore, for each state transition a message is sent to the corresponding agent, either a reply or a notification. The agent can uniquely map each reply to one of the requests that it sent to the

middlebox, because agent-unique request identifiers are used for this purpose. Notifications are self-explanatory by their definition.

Furthermore, the Group List transaction (section 2.4.3), the Group Status transaction (section 2.4.4), the Policy Rule List transaction (section 2.3.11), and the Policy Rule Status transaction (section 2.3.12) allow the agent at any time during a session to retrieve information about

- all policy rule groups it may access,
- the status and member policy rules of all accessible groups,
- all policy rules it may access, and
- the status of all accessible policy rules.

Therefore, the agent is precisely informed about the state of the middlebox (as far as the services requested by the agent are affected), and requirement 2.1.5 is met.

5.1.6. Status Report

As argued in the previous section, the middlebox unambiguously informs the agent about every state transition related to any of the services requested by the agent. Also, at any time the agent can retrieve full status information about all accessible policy rules and policy rule groups. Thus, requirement 2.1.6 is met.

5.1.7. Unsolicited Messages (Asynchronous Notifications)

The semantics includes asynchronous notifications messages from the middlebox to the agent, including the Session Termination Notification message, the Policy Rule Event Notification (REN) message, and the Group Event Notification (GEN) message (see section 2.1.2). These notifications report every change of state of policy rules or policy rule groups that was not explicitly requested by the agent. Thus, requirement 2.1.7 is met by the semantics specified above.

5.1.8. Mutual Authentication

As specified in section 2.2.1, the semantics requires mutual authentication of agent and middlebox, by using either two subsequent Session Establishment transactions or mutual authentication provided on a lower protocol layer. Thus, requirement 2.1.8 is met.

5.1.9. Session Termination by Any Party

The semantics specification states in section 2.2.2 that the agent may request session termination by generating the Session Termination request and that the middlebox may not reject this request. In turn, section 2.2.3 states that the middlebox may send the Asynchronous Session Termination notification at any time and then terminate the session. Thus, requirement 2.1.9 is met.

5.1.10. Request Result

Section 2.1 states that each request of an agent is followed by a reply of the middlebox indicating either success or failure. Thus, requirement 2.2.10 is met.

5.1.11. Version Interworking

Section 2.2.1 states that the agent needs to specify the protocol version number that it will use during the session. The middlebox may accept this and act according to this protocol version or may reject the session if it does not support this version. If the session setup is rejected, the agent may try again with another version. Thus, requirement 2.2.11 is met.

5.1.12. Deterministic Handling of Overlapping Rules

The only policy rule actions specified are ’reserve’ and ’enable’. For firewalls, overlapping enable actions or reserve actions do not create any conflict, so a firewall will always accept overlapping rules as specified in section 2.3.2 (assuming the required authorization is given).

For NATs, reserve and enable may conflict. If a conflicting request arrives, it is rejected, as stated in section 2.3.2. If an overlapping request arrives that does not conflict with those it overlaps, it is accepted (assuming the required authorization is given).

Therefore, the behavior of the middlebox in the presence of overlapping rules can be predicted deterministically, and requirement 2.1.12 is met.

5.2. Protocol Semantics Requirements

5.2.1. Extensible Syntax and Semantics

Requirement 2.2.1 explicitly requests extensibility of protocol syntax. This needs to be addressed by the concrete protocol definition. The semantics specification is extensible anyway, because new transactions may be added.

5.2.2. Policy Rules for Different Types of Middleboxes

Section 2.3 explains that the semantics uses identical transactions for all middlebox types and that the same policy rule can be applied to all of them. Thus, requirement 2.2.2 is met.

5.2.3. Ruleset Groups

The semantics explicitly supports grouping of policy rules and transactions on policy rule groups, as described in section 2.4. The group transactions can be used for lifetime extension and termination of all policy rules that are members of the particular group. Thus, requirement 2.2.3 is met.

5.2.4. Policy Rule Lifetime Extension

The semantics includes a transaction for explicit lifetime extension of policy rules, as described in section 2.3.3. Thus, requirement 2.2.4 is met.

5.2.5. Robust Failure Modes

The state transitions at the middlebox are clearly specified and communicated to the agent. There is no intermediate state reached by a partial processing of a request. All requests are always processed completely, either successfully or unsuccessfully. All request transactions include a list of failure reasons. These failure reasons cover indication of invalid parameters where applicable. In case of failure, one of the specified reasons is returned from the middlebox to the agent. Thus, requirement 2.2.5 is met.

5.2.6. Failure Reasons

The semantics includes a failure reason parameter in each failure reply. Thus, requirement 2.2.6 is met.

5.2.7. Multiple Agents Manipulating Same Policy Rule

As specified in sections 2.3 and 2.4, each installed policy rule and policy rule group has an owner, which is the authenticated agent that created the policy rule or group, respectively. The authenticated identity is input to authorize access to policy rules and groups.

If the middlebox is sufficiently configurable, its administrator can configure it so that one authenticated agent is authorized to access and modify policy rules and groups owned by another agent. Because specified semantics does not preclude this, it meets requirement 2.2.7.

5.2.8. Carrying Filtering Rules

The Policy Enable Rule transaction specified in section 2.3.8 can carry 5-tuple filtering rules. This meets requirement 2.2.8.

5.2.9. Parity of Port Numbers

As specified in section 2.3.6, the agent is able to request keeping the port parity when reserving port numbers with the PRR transaction (see section 2.3.8) and when establishing address bindings with the PER transaction (see section 2.3.9). Thus requirement 2.2.9 is met.

5.2.10. Consecutive Range of Port Numbers

As specified in section 2.3.6, the agent is able to request a consecutive range of port numbers when reserving port numbers with the PRR transaction (see section 2.3.8) and when establishing address bindings or pinholes with the PER transaction (see section 2.3.9). Thus requirement 2.2.10 is met.

5.2.11. Contradicting Overlapping Policy Rules

Requirement 2.2.11 is based on the assumption that contradictory policy rule actions, such as ’enable’/’allow’ and ’disable’/’disallows’ are supported. In conformance with decisions made by the working group after finalizing the requirements document, this requirement is not met by the semantics because no ’disable’/’disallow’ action is supported.

5.3. Security Requirements

5.3.1. Authentication, Confidentiality, Integrity

The semantics definition supports mutual authentication of agent and middlebox in the Session Establishment transaction (section 2.2.1). The use of an underlying protocol such as TLS or IPsec is mandatory. Thus, requirement 2.3.1 is met.

5.3.2. Optional Confidentiality of Control Messages

The use of IPsec or TLS allows agent and middlebox to use an encryption method (including no encryption). Thus, requirement 2.3.2 is met.

5.3.3. Operation across Untrusted Domains

Operation across untrusted domains is supported by mutual authentication and by the use of TLS or IPsec protection. Thus, requirement 2.3.3 is met.

5.3.4. Mitigate Replay Attacks

The specified semantics mitigates replay attacks and meets requirement 2.3.4 by requiring mutual authentication of agent and middlebox, and by mandating the use of TLS or IPsec protection.

Further mitigation can be provided as part of a concrete MIDCOM protocol definition -- for example, by requiring consecutively increasing numbers for request identifiers.

6. Security Considerations

The interaction between a middlebox and an agent (see [MDC-FRM]) is a very sensitive point with respect to security. The configuration of policy rules from a middlebox-external entity appears to contradict the nature of a middlebox. Therefore, effective means have to be used to ensure

- mutual authentication between agent and middlebox,
- authorization,
- message integrity, and
- message confidentiality.

The semantics defines a mechanism to ensure mutual authentication between agent and middlebox (see section 2.2.1). In combination with the authentication, the middlebox is able to decide whether an agent is authorized to request an action at the middlebox. The semantics

relies on underlying protocols, such as TLS or IPsec, to maintain message integrity and confidentiality of the transferred data between both entities.

For the TLS and IPsec use, both sides must use securely configured credentials for authentication and authorization.

The configuration of policy rules with wildcarded IP addresses and port numbers results in certain risks, such as opening overly wildcarded policy rules. An excessively wildcarded policy rule would be A0 and A3 with IP address set to 'any' IP address, for instance. This type of pinhole would render the middlebox, in the sense of security, useless, as any packet could traverse the middlebox without further checking. The local policy of the middlebox should reject such policy rule enable requests.

A reasonable default configuration for wildcarding would be that only one port number may be wildcarded and all IP addresses must be set without wildcarding. However, there are some cases where security needs to be balanced with functionality.

The example described in section 4.2 shows how SIP-signaled calls can be served in a secure way without wildcarding IP addresses. But some SIP-signaled applications make use of early media (see section 5.5 of [RFC3398]). To receive early media, the middleboxes need to be configured before the second participant in a session is known. As it is not known, the IP address of the second participant needs to be wildcarded.

In such cases and in several similar ones, there is a security policy decision to be made by the middlebox operator. The operator can configure the middlebox so that it supports more functionality, for example, by allowing wildcarded IP addresses, or so that network operation is more secure, for example, by disallowing wildcarded IP addresses.

7. IAB Considerations on UNSAF

UNilateral Self-Address Fixing (UNSAF) is described in [RFC3424] as a process at originating endpoints that attempt to determine or fix the address (and port) by which they are known to another endpoint. UNSAF proposals, such as STUN [RFC3489] are considered as a general class of workarounds for NAT traversal and as solutions for scenarios with no middlebox communication (MIDCOM).

This document describes the protocol semantics for such a middlebox communication (MIDCOM) solution. MIDCOM is not intended as a short-term workaround, but more as a long-term solution for middlebox

communication. In MIDCOM, endpoints are not involved in allocating, maintaining, and deleting addresses and ports at the middlebox. The full control of addresses and ports at the middlebox is located at the MIDCOM server.

Therefore, this document addresses the UNSAF considerations in [RFC3424] by proposing a long-term alternative solution.

8. Acknowledgements

We would like to thank all the people contributing to the semantics discussion on the mailing list for a lot of valuable comments.

9. References

9.1. Normative References

[MDC-FRM] Srisuresh, P., Kuthan, J., Rosenberg, J., Molitor, A., and A. Rayhan, "Middlebox communication architecture and framework", RFC 3303, August 2002.

[MDC-REQ] Swale, R., Mart, P., Sijben, P., Brim, S., and M. Shore, "Middlebox Communications (midcom) Protocol Requirements", RFC 3304, August 2002.

[NAT-TERM] Srisuresh, P. and M. Holdrege, "IP Network Address Translator (NAT) Terminology and Considerations", RFC 2663, August 1999.

[NAT-TRAD] Srisuresh, P. and K. Egevang, "Traditional IP Network Address Translator (Traditional NAT)", RFC 3022, January 2001.

9.2. Informative References

[RFC2246] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

[RFC2402] Kent, S. and R. Atkinson, "IP Authentication Header", RFC 2402, November 1998.

[RFC2406] Kent, S. and R. Atkinson, "IP Encapsulating Security Payload (ESP)", RFC 2406, November 1998.

[RFC3198] Westerinen, A., Schnizlein, J., Strassner, J., Scherling, M., Quinn, B., Herzog, S., Huynh, A., Carlson, M., Perry, J., and S. Waldbusser, "Terminology for Policy-Based Management", RFC 3198, November 2001.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC3398] Camarillo, G., Roach, A., Peterson, J., and L. Ong, "Integrated Services Digital Network (ISDN) User Part (ISUP) to Session Initiation Protocol (SIP) Mapping", RFC 3398, December 2002.

[RFC3424] Daigle, L. and IAB, "IAB Considerations for UNilateral Self-Address Fixing (UNSAF) Across Network Address Translation", RFC 3424, November 2002.

[RFC3489] Rosenberg, J., Weinberger, J., Huitema, C., and R. Mahy, "STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)", RFC 3489, March 2003.

Authors' Addresses

Martin Stiemerling
NEC Europe Ltd.
Network Laboratories
Kurfuersten-Anlage 36
69115 Heidelberg
Germany

Phone: +49 6221 90511-13
EMail: stiemerling@netlab.nec.de

Juergen Quittek
NEC Europe Ltd.
Network Laboratories
Kurfuersten-Anlage 36
69115 Heidelberg
Germany

Phone: +49 6221 90511-15
EMail: quittek@netlab.nec.de

Tom Taylor
Nortel
1852 Lorraine Ave.
Ottawa, Ontario
Canada K1H 6Z8

Phone: +1 613 763 1496
EMail: taylor@nortel.com

Full Copyright Statement

Intellectual Property

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.