ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

ΜΕΤΑΠΤΥΧΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΤΜΗΜΑΤΟΣ ΔΙΔΑΚΤΙΚΗΣ ΤΗΣ ΤΕΧΝΟΛΟΓΙΑΣ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΚΑΤΕΥΘΥΝΣΗ ΔΙΚΤΥΟΚΕΝΤΡΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

WiMAX Network Architecture and ASN Gateway Design

Ευγενία Βασιλίεβα
ΜΕ/0665

ΕΠΙΒΛΕΠΩΝ: Δρ. Π. ΔΕΜΕΣΤΙΧΑΣ

Πειραιάς, Ιούνιος 2009

ΕΥΧΑΡΙΣΤΙΕΣ
Θα ήθελα να ευχαριστήσω θερμά τον καθηγητή μου κ. Παναγιώτη Δεμέστιχα για την ανάθεση της παρούσας διπλωματικής εργασίας καθώς και για τη υποστήριξή του κατά την διάρκεια εκπόνησής της.
Ακόμη, θα ήθελα να ευχαριστήσω τον Κωνσταντίνο Τσαγκάρη και την Λιάνα Κουτσογιάννη για την άψογη συνεργασία και την πολύτιμη βοήθειά τους, καθώς χωρίς τη δική τους συνδρομή η ολοκλήρωση αυτής της εργασίας δεν θα μπορούσε να είχε επιτευχθεί.

Table of Contents

Table of Figures

Table of Tables

Εκτενής Περίληψη

Το WiMAX που είναι γνωστό ως Worldwide Interoperability for Micro Wave Access είναι ένα πρότυπο σχεδιασμένο από την IEEE (Institute of Electrical and Electronic Engineers) το οποίο καθορίζεται από το πρωτόκολλο IEEE 802.16-2004 (σταθερές ασύρματες εφαρμογές) και το IEEE 802.16e-2005 (κινητές ασύρματες εφαρμογές). Η ομάδα επιμέλειας του WiMAX (WiMAX Forum's Network Working Group (NWG)) έχει ορίσει το WiMAX ως το μέσο παροχής ασύρματης ευρηζωνικής πρόσβασης υψηλών ταχυτήτων σε μεγάλες αποστάσεις.

Αυτή η εργασία θα προσπαθήσει να δώσει μια γενική τεχνική επισκόπηση της WiMax αρχιτεκτονικής σύμφωνα με τις προδιαγραφές, και στην συνέχεια θα παρουσιάσει το προτεινόμενο σχεδιασμό του WiMax ASN GW δείχνοντας τις βασικές λειτουργίες των Data Plane και Control Plane, ενώ στην συνέχεια θα επικεντρωθεί στην παρουσίαση του προτεινομένου σχεδιασμού του WiMax ASN GW Management Plane.
Η εργασία έχει την παρακάτω δομή:

Κεφάλαιο 1: Εισαγωγή και η περιγραφή της δομής της εργασίας.

Κεφάλαιο 2: WiMax Network Architecture, το κεφάλαιο παρουσιάζει την αρχιτεκτονική του συστήματος δικτύου που έχει αναπτυχθεί από την WiMAX NWG. Στο πρώτο μέρος, παρουσιάζεται το μοντέλο αναφοράς του WiMAX δικτύου (Wimax Network Reference Model) και προσδιορίζει τις βασικές λειτουργικές οντότητες όπως και την αλληλοσύνδεση τους.

Το Μοντέλο Αναφοράς του WiMAX δικτύου έχει ως στόχο την δημιουργία μίας ενοποιημένης δικτυακής αρχιτεκτονικής, η οποία να υποστηρίζει σταθερές αλλά και κινητές υλοποιήσεις και επίσης θα βασίζεται σε IP Service Model.

Οι βασικές λειτουργικές οντότητες ενός WiMAX δικτύου είναι ο Κινητός Σταθμός(Mobile Station-MS), ο Σταθμός Βάσης (Base Station-BS), το Δίκτυο Εξυπηρέτησης Πρόσβασης (Access Service Network–ASN), ο πυρήνας του Δικτύου Εξυπηρέτησης Πρόσβασης (ASN Gateway-ASN GW), το Δίκτυο Εξυπηρέτησης Συνδεσιμότητας (Connectivity Service Network-CSN). Η Αλληλοσύνδεση των

παραπάνω οντοτήτων γίνεται διαμέσου των Συνδέσμων Αναφοράς (Reference Points - RP) που έχουν προσδιοριστεί από την WiMAX NWG.

Στο δεύτερο μέρος, παρουσιάζονται οι βασικές λειτουργίες που υποστηρίζονται από το WiMAX δίκτυο, όπως η Διαχείριση κινητικότητας (Mobility Managementt), Radio resource management, Ανακάλυψη και επιλογή Δικτύου (Network Discovery and Selection), Ανάθεση IP Διεύθυνσης (IP address assignment), Αρχιτεκτονική Πιστοποίησης και Ασφάλειας (Authentication and security architecture), Paging and Idle-Mode Λειτουργία.

Κεφάλαιο 3: WiMAX ASN Gateway Design, στο πρώτο μέρος γίνεται αναφορά στις λειτουργικές περιγραφές του ASN GW Data Plane, όπως προδιαγράφονται από το WiMAX πρότυπο. Οι Βασικές Λειτουργίες του Data Plane Υποσυστήματος είναι : GRE Encapsulation-Decapsulation, IP over IP Encapsulation-Decapsulation, Packet Classification, Σήμανση του QoS, Διαχείριση της Κίνησης, Υποστήριξη των χρεώσεων, Υποστήριξη του Handover και τέλος η Ασφάλεια.

Στο δεύτερο μέρος γίνεται αναφορά στην λειτουργικότητα του ASN GW Control Plane, παρουσιάζεται ο προτεινόμενος διαχωρισμός των βασικών λειτουργιών του σε διαφορετικές μονάδες λογισμικού. Το Μοντέλο Αναφοράς του WiMAX δικτύου προσδιορίζει διαφορετικά προφίλ για το ASN ώστε να εξυπηρετούνται οι μεταβλητές απαιτήσεις των διαφορετικών δικτυακών αρχιτεκτονικών. Το κάθε ASN Προφίλ ορίζει διαφορετικό διαχωρισμό των λειτουργιών μέσα στο ASN. Για παράδειγμα, το ASN Προφίλ Β αφορά μία ενιαία οντότητα που συνδυάζει τις λειτουργίες του Σταθμό Βάσης (BS) και του ASN GW. Ενώ, τα ASN Προφίλ Α και C διαχωρίζουν τις λειτουργίες μεταξύ του Σταθμού Βάσης (BS) και του ASN GW. Στο κεφάλαιο αυτό παρουσιάζεται μία πρόταση του πιθανού διαχωρισμού των βασικών ASN GW Control Plane λειτουργιών σε ξεχωριστές υπομονάδες λογισμικού (Software Modules). Οι παρακάτω υπομονάδες θα απαρτίζουν το ASN GW Control Plane και θα εμπλέκονται σε μία ή περισσότερες WiMAX λειτουργίες ελέγχου: Data Path Module, Session Management Module, Security Module, Accounting Module, Routing, Proxy/Relay ARP, Update Forwarding Entries, QoS Module, RRM Module, IP Addressing Module, Mobility Module, Location Update/Paging Module.

Και τέλος, στο τρίτο μέρος γίνεται λεπτομερή παρουσίαση του προτεινόμενου σχεδιασμού του ASN GW Management Plane, το οποίο θα είναι υπεύθυνο για την διαχείριση του ASN GW. Πρώτα, παρουσιάζεται ο διαχωρισμός της

λειτουργικότητας του ASN GW Management Plane σύμφωνα με το πρότυπο ITU-T X.700 σε βασικές λειτουργικές περιοχές διαχείρισης, γνωστός ως FCAPS (Faut (Σφάλμα), Configuration (Διαμόρφωση), Performance (Απόδοση), Security(Ασφάλεια)). Και στην συνέχεια παρουσιάζεται ο προτεινόμενος σχεδιασμός του ASN GW Management Plane.

Στην συνέχεια σύμφωνα με την προσέγγιση που ακολουθείται στην εργασία παρουσιάζονται,

1) ο προτεινόμενος σχεδιασμός του Configuration Agent, ο οποίος βρίσκεται στο ASN GW και διαχειρίζεται στην μνήμη του το Configuration Tree.

2) ο μηχανισμός υπεύθυνος για την αξιόπιστη επικοινωνία μεταξύ των Configuration Clients και του Configuration Agent, και

3) η Βάση Δεδομένων για την αποθήκευση του Configuration Tree.

4) ο προτεινόμενος σχεδιασμός της υποστήριξης του Πλεονασμού του Συστήματος και της ανοχής του σε σφάλματα (Redundancy and Fault Tolerance).

5) η προτεινόμενη προσέγγιση της Ασφάλειας και της Εξουσιοδότησης (Security and Authorization), ο προτεινόμενος τρόπος ελέγχου της πρόσβασης που θα έχουν οι χρήστες και άλλες οντότητες στο Configuration Tree. σύμφωνα με τα δικαιώματα τους. Γίνεται πρόταση τριών διαφορετικών μεθόδων: Agent Based, Radius Based, TACACS Based.

6) οι Tree Manipulation Primitives, η εξασφάλιση συνέπειας και εγκυρότητας της πρόσβασης και της διαχείρισης του Configuration Tree (εσωτερικά μέσω του Configuration Agent και εξωτερικά μέσω των Configuration Clients) θα γίνεται μέσω εκτέλεσης των Primitives. Στο σημείο αυτό περιγράφονται οι προτεινόμενες Primitives, η δομή και η λειτουργία τους.

7) το προτεινόμενο Πρωτόκολλο επικοινωνίας και η δομή των πακέτων για τα μηνύματα που θα ανταλλάσσουν οι Configuration Clients με τον Configuration Agent.

1 Introduction

WiMAX is a short name for Worldwide Interoperability of Microwave Access. The WiMAX Forum's Network Working Group (NWG) has developed and standardized these end-to-end networking aspects that are beyond the scope of the IEEE 802.16-2004 and IEEE 802.16e-2005 standards. The forum describes WiMAX as "a standards-based technology enabling the delivery of last mile wireless broadband access as an alternative to cable and DSL".

The goals of this work are, Firstly, to provide a general review of the end-to-end network systems architecture developed by the WiMAX NWG. Secondary, to present WiMAX Data Plane and Control Plane functionalities, and finally to introduce the design specification of the WiMAX ASN GW Management Plane.

The document is organized as follows:

Chapter 2: The First Part presents the WiMAX Network Architecture and the WiMAX network reference model, defines the various functional entities and their interconnections. The Second Part presents WiMAX Network Functionalities, like Mobility Management, Radio resource management, Network discovery and selection, IP address assignment, Authentication and security architecture, Paging and Idle-Mode Functionality.

Chapter 3: First Part presents a functional description of the ASN GW Data Plane, based on the WiMAX standards. Second Part describes ASN GW Control Plane Functionalities and a possible decomposition of the main ASN GW control plane functionality into various software modules. And the Last but not Least Part, presents the design specification for the development of Configuration Management of WiMAX Management Plane.

2 WiMax Network Architecture

An interoperable network architecture framework that deals with the end-to-end service aspects such as IP connectivity and session management, security, QoS, and mobility management is needed. The WiMAX Forum's Network Working Group (NWG) has developed and standardized these end-to-end networking aspects that are beyond the scope of the IEEE 802.16-2004 and IEEE 802.16e-2005 standards.
This chapter looks at the end-to-end network systems architecture developed by the WiMAX NWG and introduces the WiMAX network reference model and define the various functional entities and their interconnections.

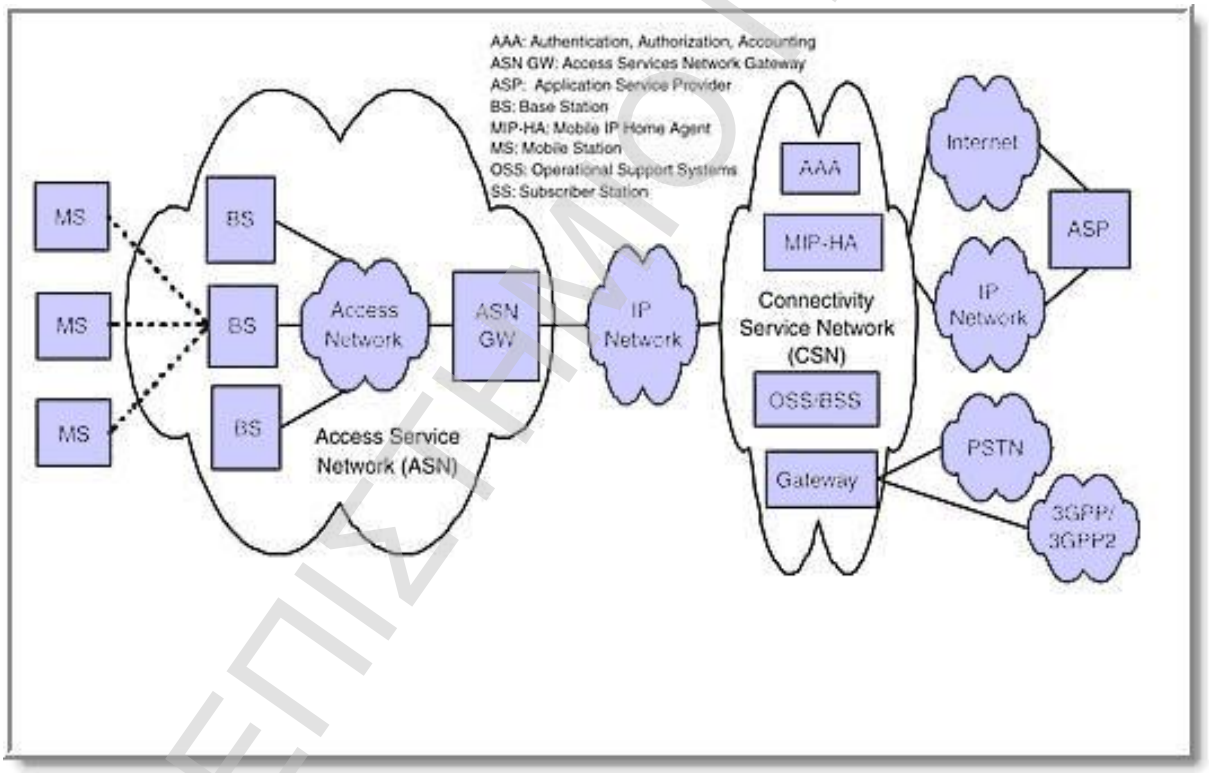


Figure 1: *WiMax Network Architecture*

2.1 Network Reference Model

2.1.1 Overview and Definitions

A network reference model has been developed by the WiMax Network Working Group (WiMAX NWG) in order to provide an architecture framework for WiMAX deployments. This model aims at ensuring interoperability among different WiMAX equipment and operators. The network reference model envisions unified network

architecture for supporting fixed, nomadic, and mobile deployments and is based on an IP service model.

The main parts of the network include the mobile stations (MS) used by the users to connect to the network, the Access Service Network (ASN) and the Connectivity Service Network (CSN) that provides the IP connectivity. The Network Reference Model (NRM) is shown in Figure 2.

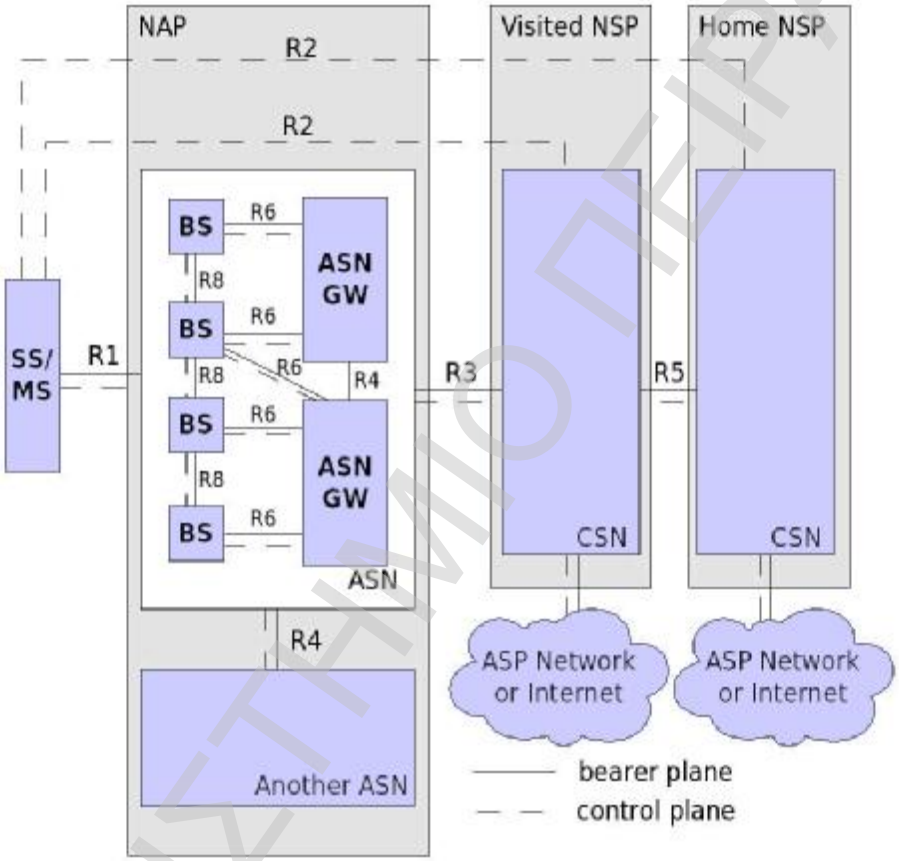


Figure 2: *Network reference model (NRM)*

Three different Profiles (A, B and C) are defined in the specifications that divide the functionality of the ASN between the BS and the ASN GW. In Profile B, ASN GW and BS functions are developed within the same system. In Profiles A and C, ASN functions are mapped into ASN GW and BS. Their main difference is in the mapping of the Handover control and the Radio Resource Controller which in Profile A are mapped to the ASN GW while in Profile C they are mapped to the BS.

A *Reference Point* is defined within WiMAX NWG as a conceptual link that connects two groups of functions that reside in different functional entities of the ASN, CSN,

or MS. Reference points are not necessarily a physical interface, except when the functional entities on either side of it are implemented on different physical devices. The reference points are explained in the following subsection.

2.1.2 Reference Points

Figure 2 introduces several interoperability reference points. A reference point (RP) is a conceptual point between two groups of functions that resides in different functional entities on either side of it. These functions expose various protocols associated with an RP. All protocols associated with a RP may not always terminate in the same functional entity. The normative reference points between the major functional entities are in the following subsections.

Reference point R1

Reference Point R1 represents the interface between the wireless device and the base station, and consists of the protocols and procedures between MS and ASN as per the air interface (PHY and MAC) specifications (IEEE P802.16e-2005, IEEE P802.16-2004 and IEEE 802.16g). Reference point R1 may include additional protocols related to the management plane.

Reference point R2

Reference Point R2 represents the link between the MS and the CSN, and consists of protocols and procedures between the MS and CSN associated with Authentication, Services Authorization and IP Host Configuration management. This reference point is *logical* in that it does not reflect a direct protocol interface between MS and CSN. The authentication part of reference point R2 runs between the MS and the CSN operated by the home NSP, however the ASN and CSN operated by the visited NSP may partially process the aforementioned procedures and mechanisms. Reference Point R2 might support IP Host Configuration Management running between the MS and the CSN (operated by either the home NSP or the visited NSP).

Reference point R3

Reference Point R3 represents the link between the ASN and the CSN. R3 consists of the set of Control Plane protocols between the ASN and the CSN to support AAA, policy enforcement and mobility management capabilities. It also encompasses the Bearer Plane methods (e.g. tunneling) to transfer user data between the ASN and the CSN.

Reference point R4

Reference Point R4 represents the link between an ASN and another ASN. R4 consists of the set of Control and Bearer Plane protocols originating/terminating in various functional entities of an ASN that coordinate MS mobility between ASNs and ASN GWs. R4 is the only interoperable reference point between similar or heterogeneous ASNs.

Reference point R5

Reference Point R5 represents the link between a CSN and another CSN. R5 consists of the set of Control Plane and Bearer Plane protocols for internetworking between the CSN operated by the home NSP and that operated by a visited NSP.

Reference point R6

Reference point R6 located within an ASN and represents a link between the BS and the ASN GW. R6 consists of the set of control and Bearer Plane protocols for communication between the BS and the ASN GW. The Bearer Plane consists of intra-ASN datapath between the BS and ASN gateway. The Control Plane includes protocols for datapath establishment, modification, and release control in accordance with the MS mobility events.

Reference point R7

Reference Point R7 located within the ASN GW and represents internal communication within the gateway. R7 consists of the optional set of Control Plane protocols e.g., for AAA and Policy coordination in the ASN gateway as well as other protocols for co-ordination between the two groups of functions identified in R6. The decomposition of the ASN functions using the R7 protocols *is optional*.

Reference point R8

Reference Point R8 located within an ASN and represents a link between two base stations. R8 consists of the set of Control Plane message flows and optionally Bearer Plane data flows between the base stations to ensure fast and seamless handover. The Bearer Plane consists of protocols that allow the data transfer between Base Stations involved in handover of a certain MS.

2.1.3 ASN Reference Model

2.1.3.1 ASN Definition

The Access Service Network (ASN) defines a logical boundary and represents a convenient way to describe aggregation of functional entities and corresponding message flows associated with the access services. The ASN represents a boundary for functional interoperability with WiMAX clients, WiMAX connectivity service functions and aggregation of functions embodied by different vendors. Mapping of functional entities to logical entities within ASNs as depicted in the Network Reference Model (NRM) is informational.

2.1.3.2 ASN Decomposition

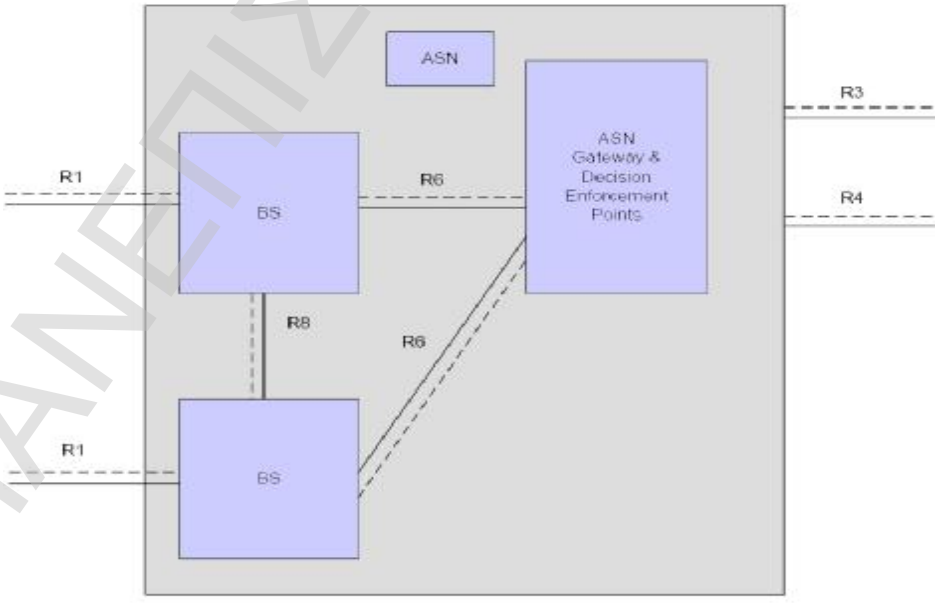


Figure 3: *ASN Reference Model containing a single ASN GW*

An ASN shares R1 reference point (RP) with an MS, R3 RP with a Connectivity Service Network (CSN) and R4 RP with another ASN. The ASN consists of at least one instance of a Base Stations (BS) and at least one instance of an ASN Gateway (ASN GW). A BS is logically connected to one or more ASN Gateways (Figure 4) The R4 reference point is the only RP for Control and Bearer Planes for interoperability between similar or heterogeneous ASNs. Interoperability between any types of ASNs is feasible with the specified protocols and primitives exposed across R1, R3 and R4 Reference Points.

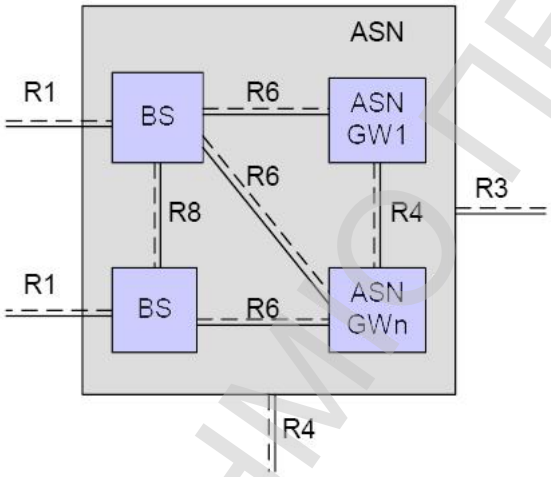


Figure 4*: ASN Reference Model containing multiple ASN GW*

When ASN is composed of *n* ASN GWs (where *n* > 1), Intra ASN mobility MAY involve R4 control messages and Bearer Plane establishment. For all applicable protocols and procedures, the Intra-ASN reference point R4 shall be fully compatible with the Inter-ASN equivalent.

2.1.3.2 BS

The WiMAX *Base station* (BS) is defined by the standard as a logical entity that embodies a full instance of the WiMAX MAC and PHY in compliance with the IEEE 802.16 suite of applicable standards and may host one or more access functions. Thus, it is responsible for providing the air interface to the MS. Additional functions that may be part of the BS include micro mobility management functions, such as handoff triggering and tunnel establishment, radio resource management, QoS policy enforcement, traffic classification, DHCP (Dynamic Host Control Protocol) proxy, key management, session management, and multicast group management. A BS may

be connected to more than one ASN GW if required, as a load balancing or redundancy option.

2.1.3.4 ASN GW

The *ASN Gateway* (ASN GW), as defined by the standard, is a logical entity that represents an aggregation of Control Plane functional entities that are either paired with a corresponding function in the ASN (e.g. BS instance), a resident function in the CSN or a function in another ASN. The ASN GW may also perform Bearer Plane routing or bridging function. Practically the ASN GW is a *control and data traffic aggregation point* within the ASN.

Functions within the ASN GW include service flow authorization, service flow authentication, key distribution, session and context maintenance, handover coordination, mobility management, paging control, RRM control/relay, data path management, QoS and policy enforcement, traffic classification, mobile IP foreign agent, accounting client and routing to the selected CSN.

The ASN GW may be decomposed as shown in Figure 5 into the decision point (DP) which includes non-bearer plane function and the enforcement point (EP) which includes bearer-plane function. DP and EP are connected through the reference point R7.

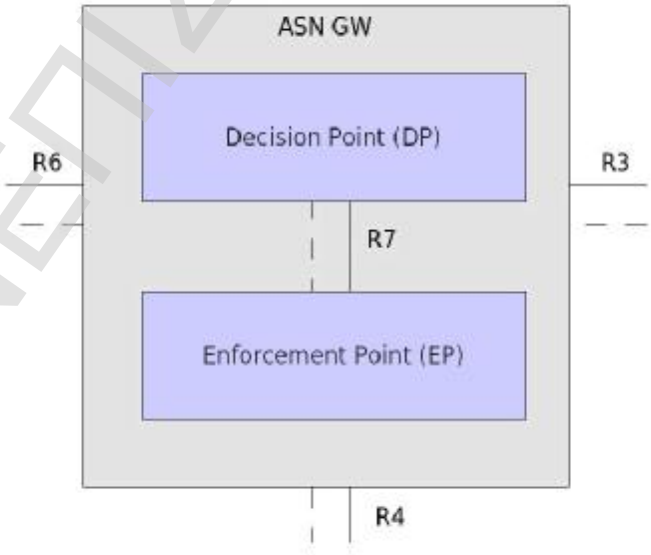


Figure 5: *ASN GW Decomposition*

2.1.4 ASN Profile Introduction

A profile maps ASN functions into BS and ASN GW so that protocols and messages over the exposed reference point are identified. (The following text describes the three profiles of an ASN based on the R1 v1.2 Stage 2 specifications.) These three profiles show three possible implementations of the ASN and do not necessarily mandate a vendor to support all three. If a vendor chooses to implement any given profile, then that vendor's implementation shall conform to the chosen profile. The depiction of a function on either the ASN GW or the BS in the figures below does not imply that the function exists in all manifestations of this profile. Instead, it indicates that if the function existed in a manifestation it would reside on the entity shown.

2.1.4.1 Profile A

ASN functions are mapped into ASN GW and BS as shown in Figure 6. Some of the key attributes of Profile A are:

- *Handover* (HO) Control is in the ASN GW
- *Radio Resource Control* (RRC) is in ASN GW that allows *Radio Resource Management* (RRM) among multiple BSs
- ASN Anchored mobility among BSs shall be achieved by utilizing R6 and R4 physical connections.

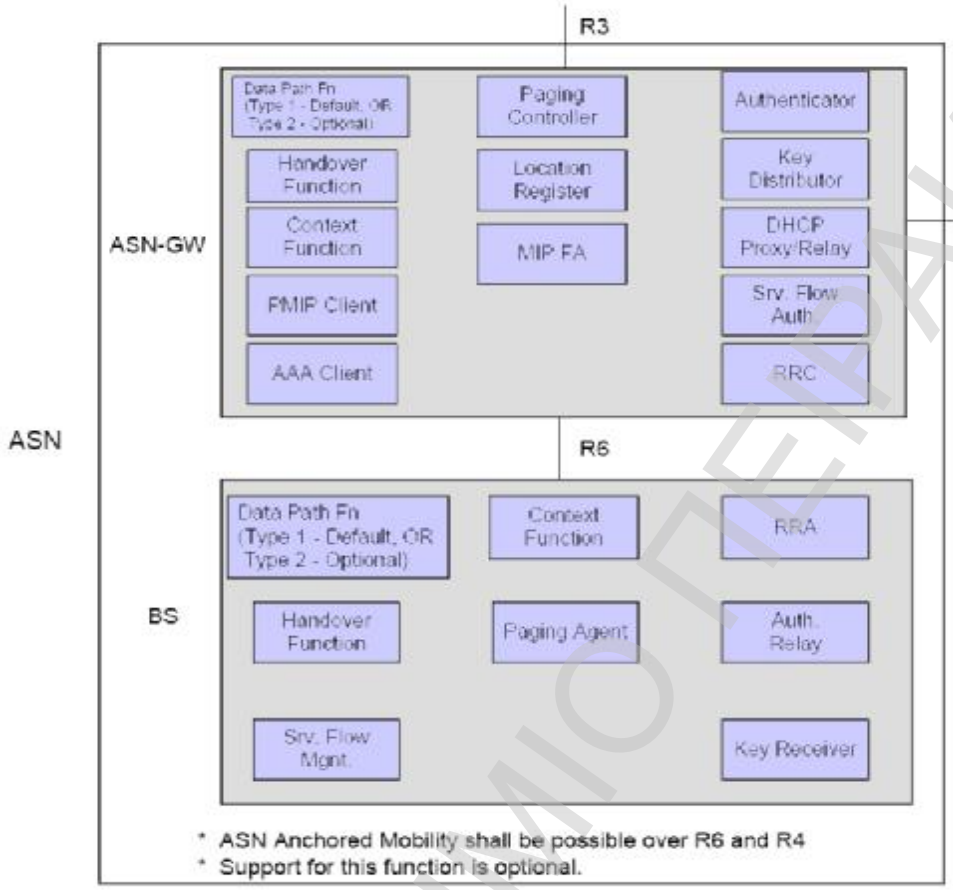


Figure 6: *Functional View of ASN Profile A*

Table 1 illustrates the reference points over which intra-profile intra-ASN interoperability is achieved in accordance with Profile A.

Function Categories	Function	ASN Entity Name	Exposed Protocols, Primitives, API	Associated RP
Security	Authenticator	ASN GW	Auth Relay Primitives	R6
	Auth Relay	BS	Auth Relay Primitives	R6
	Key Distributor	ASN GW	AK Transfer Primitives	R6
	Key Receiver	BS	AK Transfer Primitives	R6
IntraASN Mobility	Data Path Fn (Type 1 or 2)	ASN GW & BS	Data Path Control Primitives	R6
	Handover Fn	ASN GW & BS	HO Control Primitives	R6
	Context Server & Client	ASN GW & BS		R6
L3 Mobility	MIP FA	ASN GW	Client MIP	R6
	MIP AR	ASN-GW	Client MIP	R6
Radio Resource Management	RRC	ASN GW	RRM Primitives	R6
	RRA	BS	RRM Primitives	R6
Paging	Paging Agent	BS	Paging & Idle Mode Primitives	R6
	Paging Controller	ASN GW	Paging & Idle Mode Primitives	R6
QoS	SFA	ASN GW	BS	R6
	SFM	BS		

Table 1: *Profile A Interoperability Reference Points*

2.1.4.2 Profile B

Profile B ASNs are characterized by unexposed intra-ASN interfaces and hence intra-ASN interoperability is not specified. However, Profile B ASNs shall be capable of interoperating with other ASNs of any profile type via R3 and R4 reference points. Inter-ASN anchored mobility shall be possible via R4.

Mapping of ASN functions is not specified for Profile B ASNs and as such there can be several different realizations of a Profile B implementation. These include, for example, implementations where all the ASN functions are located within a single

physical device such as an *Integrated BS network entity* (IBS), and ones where ASN functionality is distributed over multiple network nodes.

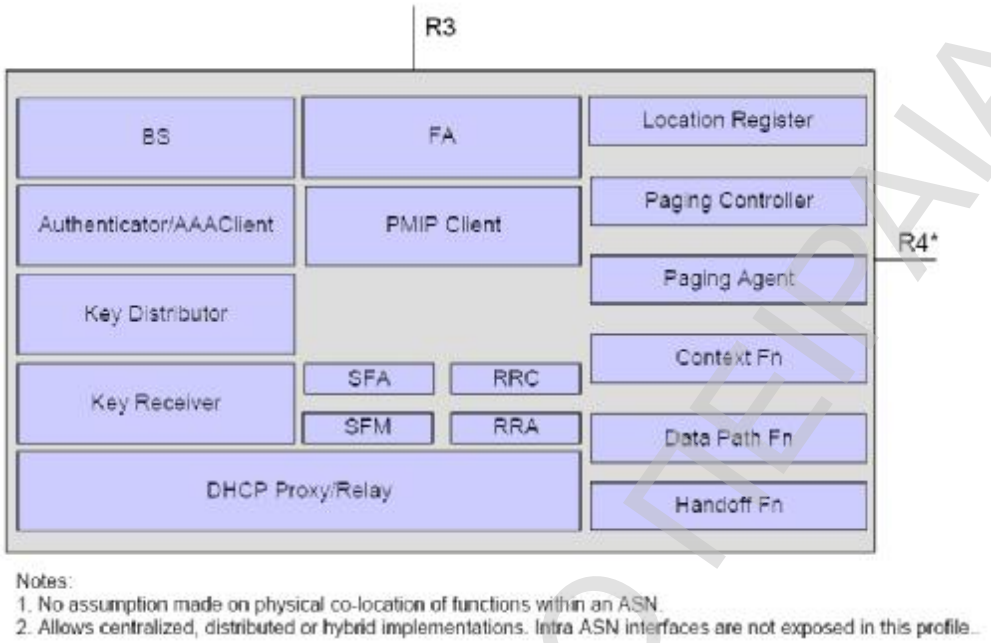


Figure 7: *Functional View of ASN Profile B*

2.1.4.3 Profile C

According to Profile C, ASN functions are mapped into ASN GW and BS as shown in Figure 8. Key attributes of Profile C are:

- HO Control is in the Base Station.
- RRC is in the BS that would allow RRM within the BS. An "RRC Relay" is in the ASN GW, to relay the RRM messages sent from BS to BS via R6.
- As in Profile A, ASN Anchored mobility among BSs SHALL be achieved by utilizing R6 and R4 physical connections.

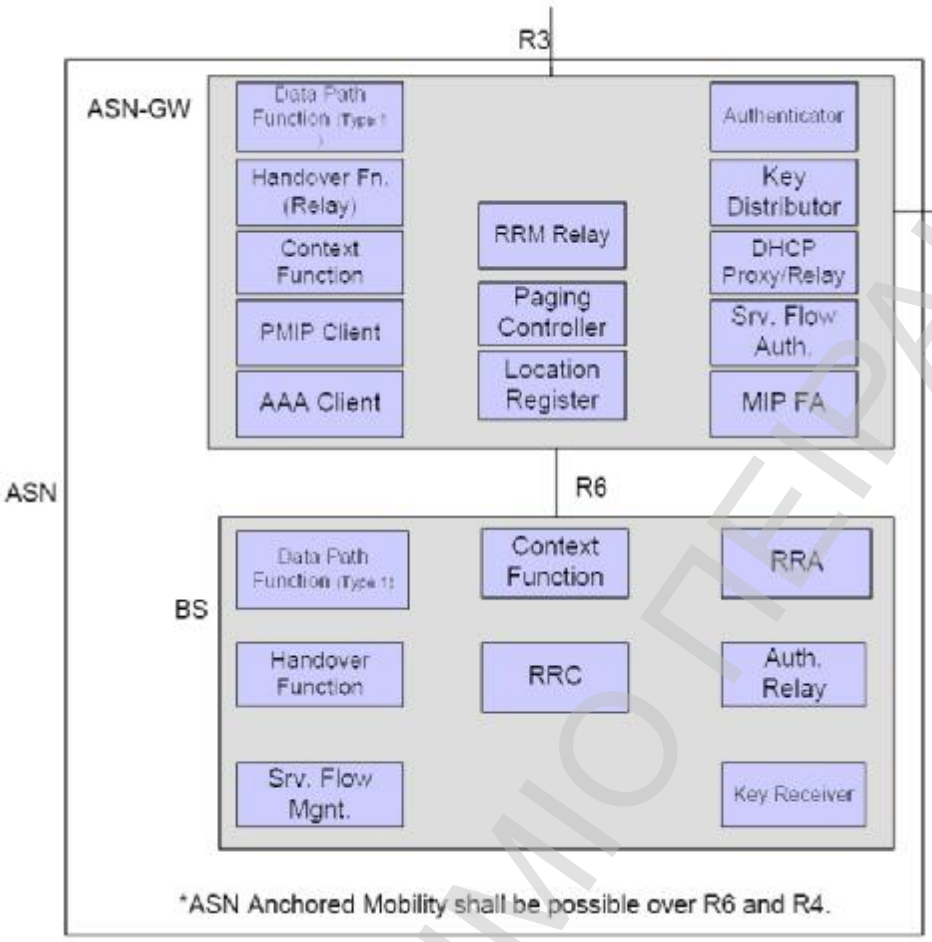


Figure 8: *Functional View of ASN Profile C*

Table 2 illustrates the reference points over which intra-profile intra-ASN interoperability is achieved in accordance with Profile C.

Function Categories	Function	ASN Entity Name	Exposed Protocols, Primitives, API	Associated RP
Security	Authenticator	ASN GW	Auth Relay Primitives	R6
	Auth Relay	BS	Auth Relay Primitives	R6
	Key Distributor	ASN GW	AK Transfer Primitives	R6
	Key Receiver	BS	AK Transfer Primitives	R6
IntraASN Mobility	Data Path Function (Type 1)	ASN GW & BS	Data Path Control Primitives	R6
	Handover Fu	ASN GW & BS	HO Control Primitives	R6
	Context Server & Client	ASN GW & BS		R6
L3 Mobility	MIP FA	ASN GW	Client MIP	R6
	MIP AR	ASN-GW	Client MIP	R6
Radio Resource Management	RRC	BS	RRM Primitives	R6
	RRA	BS	None (BS internal)	-
Paging	RRC Relay	ASN GW	RRM Primitives	R6
	Paging Agent	BS	Paging & Idle Mode Primitives	R6
	Paging Controller	ASN GW	Paging & Idle Mode Primitives	R6
QoS	SFA	ASN GW	QoS Primitives	R6
	SFM	BS		

Table 2: *Profile C Interoperability Reference Points*

2.1.5 ASN Functional Protocols

Protocol Layering of WiMAX considers end-to-end protocol layering. Data and control packets are forwarded from the MS to the CSN in uplink. The traffic is concentrated in the ASN GW and forwarded to the CSN and same way, concentrated in the ASN GW for downlink and distributed to the MSs residing in different BSs. IP packets use IP convergence sublayer (IP-CS) or Ethernet convergence sublayer (ETH-CS) over IEEE 802.16e. The IP-CS with IP-in-IP encapsulation between BS and ASN GW is considered in most designs. Bridging is also another way of routing packet within ASN.

Network Discovery and Selection implements manual or automatic selection of the appropriate network. MS first discovers all the NAPs where each has an Operator ID embedded into Base Station ID and transmitted with DL-MAP of each frame. And MS continue to listen the channel for SII-ADV signal which system identity information advertisement to advertise NSP IDs. The MS selects one NSP from the list according to an algorithm and performs network entry and provide its identity and its home NSP domain with a network access identifier (NAI). The ASN selects the next AAA hop from the realm portion of the NAI.

IP Address Assignment is done through DHCP or AAA: ASN hosts DHCP relay or DHCP proxy respectively. In order to deliver the point of attachment IP address to MS. For IPv6 there is access router in ASN to obtain globally routable IP address. The MS gets the care-of-address (CoA) from ASN and home address (HoA) from CSN. Authentication and Security Architecture implements 802.16e security with IETF EAP framework. AAA framework is used for service flow authorization, mobility management and policy control. AAA framework is based on pull model in which supplicant sends a request to ASN and ASN forwards it to AAA server. The AAA return with appropriate response to ASN which set up the service and inform the MS. The elements are depicted in Figure 9.

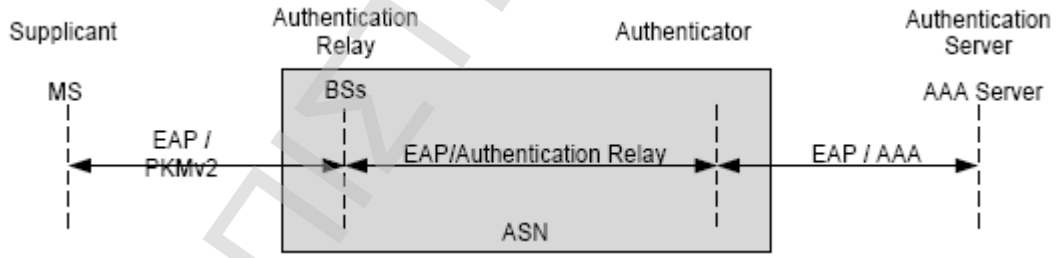


Figure 9 : *Authentication Relay Inside the ASN*

User and device authentication is supported with PKMv2 and EAP. PKMv2 is between MS and BS and BS relays this EAP messages to ASN GW where AAA client encapsulates the EAP and forwards to AAA server in the CSN over RADIUS. EAP-AKA, EAP-TLS, EAP-SIM, EAP-PSK, EAP-TTLS are the supported EAP types. Both user and device authentication is performed with double-EAP and device credentials are in the form of digital certificate, secret key, or X.509 certificate.

Quality of Service Architecture in WiMAX complements the QoS framework in IEEE 802.16e-2005 QoS model. The QoS provides rich set of variety: per user and per

service flow basis differentiated levels; admission control; bandwidth optimization. QoS provides static and dynamic service flow creation. For each service there is provisioned, admitted, and active states. When flow is in active state it starts getting the service. Entities are Policy function and AAA server residing in CSN, Service flow management residing in BS, and Service flow authorization residing in ASN GW.

Mobility Management implements mobility with the ASN and across the. ASN-anchored mobility is when MS moves within the same Foreign Agent domain residing in ASN GW. Control signals use R6 and R8 reference points and data path shift happens in ASN GW with new R6 to target BS when handover is complete. CSN-anchored mobility additional to ASN-anchored mobility triggers the FA change through Home Agent. Now, R3 and R4 reference points also become active.

Radio Resource Management is responsible to fully utilize the network by information gathering and implementing decisions. The information such as radio-related measurements; base station spare capacity reports are concentrated to assist handover decision and load balancing decisions.

Paging and Idle Mode Operation is responsible to maintain a track and alert for MS when it is in idle mode for battery power saving reasons. Paging is executed to alert MS when there is an incoming message. MS is tracked when it is in the idle mode and information is stored to a location register (LR). Granularity of track is bigger than cell size since a paging group (PG) is composed of multiple cell and when a MS moves across paging groups, location update occurs via R6 and/or R4. Paging Controller (PG) in ASN GW retrieves the location from LR and alerts the paging agent in (PA) in BS to signal to MS.

2.1.6 CSN Reference Model

The *Connectivity Service Network* (CSN) consists of all the functions/equipment that enable IP connectivity to WiMAX subscribers. As a consequence, the CSN includes the following functions:

- User connection authorization and Layer 3 access (IP address allocation for user sessions, AAA proxy server and functions)
- QoS management (policy and admission control based on user profiles)

- Mobility support based on Mobile IP (Home Agent (HA), function for inter-ASN mobility)
- Tunneling (based on IP protocols) with other equipment/networks (ASN-CSN tunneling support, inter-CSN tunneling for roaming)
- WiMAX subscriber billing
- WiMAX services (Internet access, location-based services, connectivity for peer-to-peer services, provisioning, authorization and/or connectivity to IMS).

2.2 Network Functionalities

2.2.1 Mobility Management

The WiMAX mobility-management architecture was designed to

- Minimize packet loss and handoff latency and maintain packet ordering to support seamless handover even at vehicular speeds
- Comply with the security and trust architecture of IEEE 802.16 and IETF EAP RFCs during mobility events
- Supporting macro diversity handover (MOHO) as well as fast base station switching (FBSS)7
- Minimize the number of round-trips of signaling to execute handover
- Keep handover control and data path control separate
- Support multiple deployment scenarios and be agnostic to ASN decomposition
- Support both IPv4- and IPv6-based mobility management and accommodate mobiles withmultiple IP addresses and simultaneous IPv4 and IPv6 connections
- Maintain the possibility of vertical or intertechnology handovers and roaming between NSPs
- Allow a single NAP to serve multiple MSs using different private and public IP domains owned by different NSPs
- Support both static and dynamic home address configuration
- Allow for policy-based and dynamic assignment of home agents to facilitate such features as route optimization and load balancing

The WiMAX network supports two types of mobility:

(1) *ASN-anchored mobility* and (2) *CSN-anchored mobility*.

ASN-anchored mobility is also referred to as intra-ASN mobility, or micromobility. In this case, the MS moves between two data paths while maintaining the same anchor foreign agent at the northbound edge of the ASN network. The handover in this case happens across the R8 and/or R6 reference points. ASN-anchored handover typically involves migration of R6, with R8 used for transferring undelivered packets after handover. It is also possible to keep the layer 3 connection to the same BS (anchor BS) through the handover and have data traverse from the anchor BS to the serving BS throughout the session.

CSN-anchored mobility is also referred to as inter-ASN mobility, or macromobility. In this case, the MS changes to a new anchor FA, this is called FA migration. The new FA and CSN exchange signaling messages to establish data-forwarding paths. The handover in this case happens across the R3 reference point, with tunneling over R4 to transfer undelivered packets. Figure 10 illustrates the various possible handover scenarios supported in WiMAX.

Mobility management is typically triggered when the MS moves across base stations based on radio conditions. It may also be triggered when an MS wakes up from idle mode at a different ASN or when the network decides to transfer R3 end points for an MS from the serving FA to a new FA for resource optimization. In many cases, both ASN- and CSN-anchored mobility may be triggered. When this happens, it is simpler and preferable to initiate the CSN-anchored mobility after successfully completing the ASN-anchored mobility.

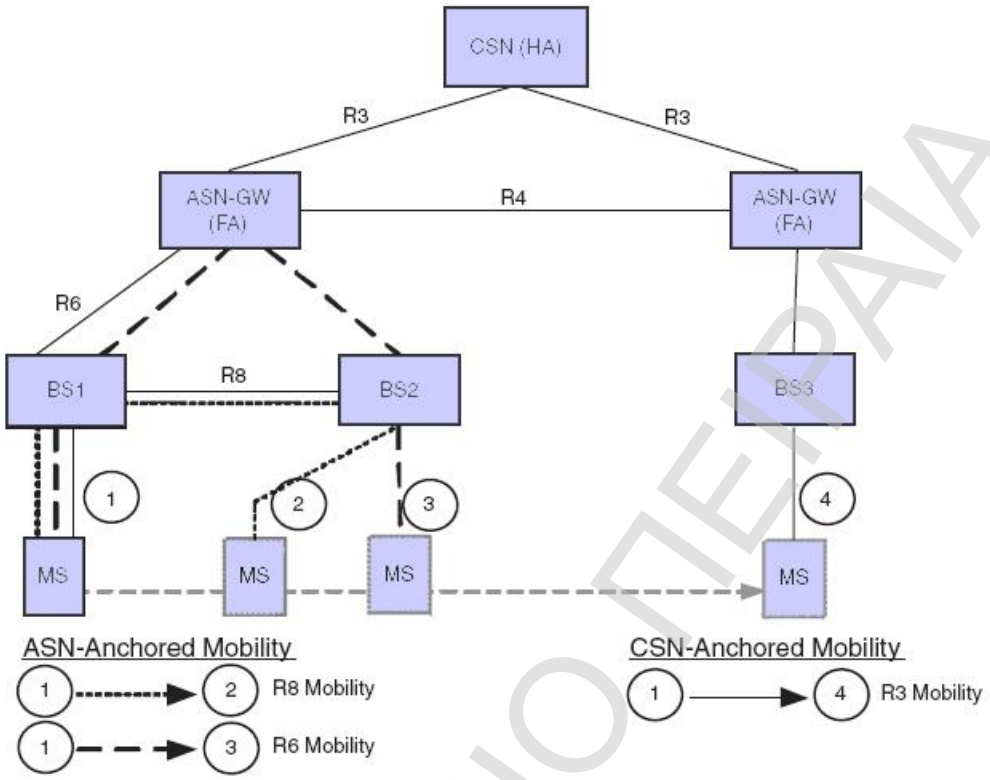


Figure 10: *Handover scenarios supported in WiMAX*

2.2.1.1 ASN - Anchored Mobility

ASN-anchored mobility supports handoff scenarios in which the mobile moves its point of attachment from one BS to another within the same ASN. This type of movement is invisible to the CSN and does not have any impact at the network- or IP-layer level. Implementing ASN anchored mobility does not require any additional network-layer software on the MS.

The WiMAX standard defines three functions that together provide ASN-anchored mobility managment: data path function, handoff function, and context function.

1. The *data path function (DPF)* is responsible for setting up and managing the bearer paths needed for data packet transmission between the functional entities, such as BSs and ASN GWs, involved in a handover. This includes setting up appropriate tunnels between the entities for packet forwarding, ensuring low latency, and handling special needs, suchas multicast and broadcast. Conceptually, four DPF entities are defined in WiMAX:

a. *anchor data path function,* which is the DPF at one end of the data path that anchors the data path associated with the MS across handovers;
b. *serving data path function,* which is the DPF associated with the BS that currently has the IEEE 802.16e link to the MS;
c. *target data path function*, which is the DPF that has been selected as the target for the handover; and
d. *relaying data path function*, which mediates between serving, target, and anchor DPFs to deliver the information.

2. The *handoff (HO) function* is responsible for making the HO decisions and performing the signaling procedures related to HO. The HO function supports both mobile and network initiated handovers, FBSS, and MDHO. Note, however, that FBSS and MDHO are not supported in Release 1. Like the DPF, the HO function is also distributed among many entities, namely, *serving HO function, target HO function,* and *relaying HO function*. The serving HO function controls the overall HO decision operation and signaling procedures, signaling the target HO function to prepare for handover and informing the MS. Signaling between the serving and target HO functions may be via zero or more relaying HO functions.
A relaying HO may modify the content of HO messages, and impact HO decisions.
The WiMAX standard defines a number of primitives and messages such as HO Request, HO Response, and HO Confirm, for communication among the HO functions.

3. The *context function* is responsible for the exchange of state information among the network elements impacted by a handover. During a handover execution period, there may be MS-related state information in the network and network-related state information in the MS that need to be updated and/or transferred. For example, the target BS needs to be updated with the security context of the MS that is being handed in.The context function is implemented using a client/server model. The *context server* keeps the updated session context information, and the *context client,* which is implemented in the functional entity that has the IEEE 802.16e air link, retrieves it during handover. There could be a *relaying context function* between the context server and context client. The session-context information exchanged between the context client and server may include MS NAI, MS MAC address, anchor ASN GW

associated with the MS, SFID and associated parameters, CID, home agent IP address, CoA, DHCP Server, AAA server, security information related to PKMv2 and proxy MIP, and so on.

2.2.1.2 CSN - Anchored Mobility

CSN-anchored mobility refers to mobility across different ASNs, in particular across multiple foreign agents. It is assumed that for the current release FAs belong to the same NAP.

CSN-anchored mobility requires IP-layer mobility management. Mobile IP (MIP) is the IETF protocol for managing mobility across IP subnets. Mobile IP is used in WiMAX networks to enable CSN-anchored mobility.

The WiMAX network defines two manifestations of MIP implementations for supporting CSN anchored mobility. The first one is based on client MIP meaning that the MS needs to have a MIP client (Mobile Node – MN). The other is based on having a proxy MIP in the network, that is to say the MN is implemented in the ASN on behalf of the MS. No particular software (change in IP stack) is needed to be present in the MS with proxy MIP. Both proxy MIP and client MIP are mandatory for the current specification of WiMAX. Coexistence may be also supported, though in this case the MS should support either mobile IP with client MIP or regular IP with proxy MIP.

2.2.2 Radio resource management

The RRM function is aimed at maximizing the efficiency of radio resource utilization and is performed within the ASN in the WiMAX network. Tasks performed by the WiMAX RRM include (1) triggering radio-resource-related measurements by BSs and MSs, (2) reporting these measurements to required databases within the network, (3) maintaining one or more databases related to RRM, (4) exchanging information between these databases within or across ASNs, and (5) making radio resource information available to other functional entities, such as HO control and QoS management.

RRM function in WiMAX architecture introduces a concept of Radio Resource Agent (RRA) and Radio Resource Controller (RRC) functional elements and signaling between RRA and RRC and between RRC and RRC.

Implementation of RRM is optional because a) many RRM tasks are executed autonomously and locally in each BS without any interaction to other RRM Functional Entities in the ASN b) some RRM related signalling is implicitly included in signalling between other ASN functions, e.g. Handover function, QoS function. If RRM is supported, then ASN SHALL have at least one RRC. The radio resource agent (RRA) resides in each BS and collects and maintains radio resource indicators, such as received signal strength, from the BS and all MSs attached to the BS. The RRA also communicates RRM control information, such as neighbor BS set and their parameters, to the MSs attached to it. The radio resource controller (RRC) is a logical entity that may reside in each BS, in ASN GW, or as a stand-alone server in the ASN. The RRC is responsible for collecting the radio resource indicators from the various RRAs attached to it and maintaining a "regional" radio resource database. When the RRC and the RRA are implemented in separate functional entities, they communicate over the R6 reference point. Multiple RRCs may also communicate with one another over the R4 reference point if implemented outside the BS and over the R8 reference point if integrated within the BS.

Each RRA in the BS is also responsible for controlling its radio resources, based on its own measurement reports and those obtained from the RRC. Control functions performed by the RRA include power control, MAC and PHY supervision, modification of the neighbor BS list, assistance with the local service flow management function and policy management for service flow admission control, and assistance with the local HO functions for initiating HO.

Standard procedures and primitives are defined for communication between the RRA and the RRC. The procedures may be classified as one of two types. The first type, called information reporting procedures, is used for delivery of BS radio resource indicators from the RRA to the RRC, and between RRCs. The second type, called decision-support procedures from RRC to RRA is used for communicating useful hints about the aggregated RRM status that may be used by the BS for various purposes. Defined RRM primitives include those for requesting and reporting link-level quality per MS, spare capacity available per BS, and neighbor BS radio resource status.

Figure 11 shows two generic reference models for RRM as defined in WiMAX. The first one shows the split-RRM model, where the RRC is located outside the BS; the second shows the RRC collocated with RRA within the BS. In the split-RRM case, RRAs and RRC interact across the R6 reference point. In the integrated RRM model, the interface between RRA and RRC is outside the scope of this specification, and only the information reporting procedures, represented with dashed lines, are standardized. The decision-support procedures, shown as solid lines between RRA and RRC in each BS, remain proprietary. Here, the RRM in different BSs may communicate with one another using an RRM relay in the ASN GW. The split model and the integrated model are included as part of ASN profiles A and C, respectively.

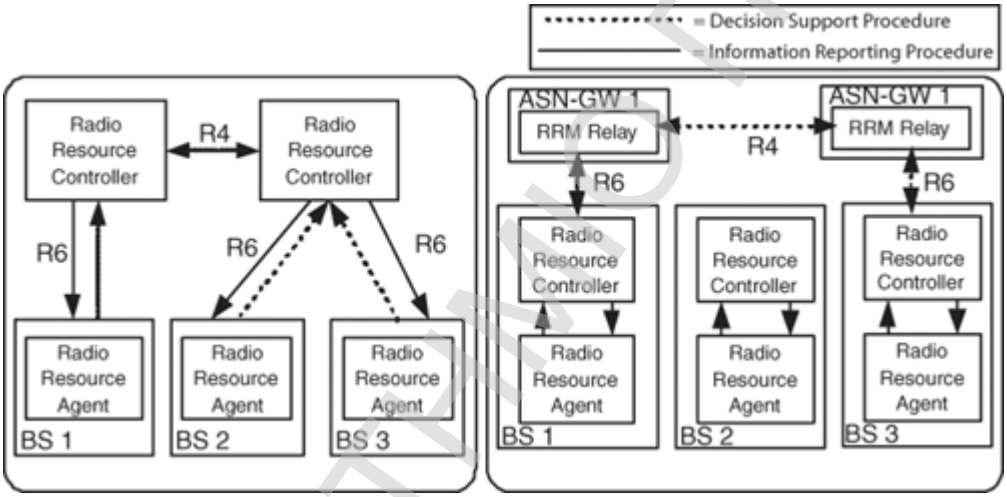


Figure 11: *Generic reference models for RRM: (a) split RRM and (b) integrated RRM*

2.2.3 Paging and Idle-Mode Operation

In order to save battery power on the handset, the WiMAX MS goes into idle mode when it is not involved in an active session. Idle mode in mobile WiMAX is a mechanism that allows the MS to receive broadcast DL transmission from the BS without registering itself with the network.

Paging refers to procedures used by the network to seek an MS in idle mode in the coverage area of a predefined set of Base Station(s) identified by a Paging Group (as per IEEE 802.16e specification). In addition, Paging Update refers to procedures to obtain location update or network entry from an MS in idle mode. Paging procedures

are implemented using Paging MAC message exchanges between MS and BS, under the control of a higher-layer paging management functions.

Paging Controller (PC): Paging Controller is a functional entity that administers the activity of idle-mode MS in the network. It is uniquely identified by a PC ID (6 bytes) in IEEE 802.16e and may be either collocated with the BS or separated from it across an R6 reference point. Each idle-mode MS requires a single PC containing its updated location information. This PC is referred to as the anchor PC. Additional PCs in the network may, however, participate in relaying paging and location management messages between the Paging Agent and the anchor PC. These additional PCs are called relay PCs.

Paging Agent (PA): The Paging Agent is a BS functional entity that handles the interaction between the PC and the IEEE 802.16e–specified paging-related functions. A Paging Agent is co-located with BS. The interaction between PA and PC is subject to standardization only when it is performed through R6 reference point

Location Register (LR): The Location Register LR is a distributed database that contains information about the idle mode MSs. The information for each MS includes, but is not limited to, current paging group ID, paging cycle, paging offset, and service flow information. An instance of the LR is associated with every anchor PC. When an MS moves across paging groups, location update occurs across PCs via R6 and/or R4 reference points, and the information is updated in the LR associated with the anchor PC assigned to the MS.

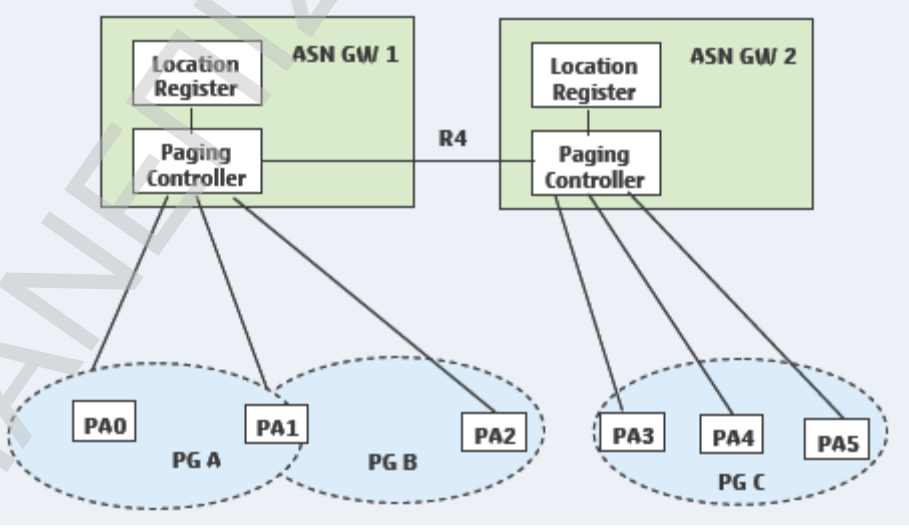


Figure 12: *Paging Network Reference Model*

Paging Groups comprise a set of Base Stations. An access network (i.e. NAP) may consist of one or more Paging Groups. A NAP may comprise one or more Paging Controllers. Each Idle MS in the NAP is assigned a single Paging Controller, called Anchor PC. An MS in idle mode must be accessible in the network during Paging Intervals for Paging and Location Updates. PC and LR entries are created when MS enters idle mode. User plane tunnel between FA, data path and BS are removed. LR is updated when idle MS crosses the boundary of Paging Group and is deleted when MS exits idle mode.

An MS in idle mode periodically monitors the downlink transmission of the network to determine the paging group of its current location. If it moves to a new paging group, an MS performs a paging group update, during which it informs the network of the current paging group in which it is present. When, due to pending downlink traffic, the network needs to establish a connection with an MS in idle mode, the network needs to page the MS only in all the BSs belonging to the current paging group of the MS. Without the concept of the paging area, the network would need to page the MSs in all the BSs within the entire network. Each paging area should be large enough so that the MS is not required to perform a paging area update too often and should be small enough so that the paging overhead associated with sending the page on multiple BSs is low enough.

2.2.4 Network Discovery and Selection

The procedure is applicable to the first time use, initial network entry, network re-entry, or when an MS transitions across NAP coverage areas. It is assumed that an MS will operate in an environment where multiple networks are available and candidate for the MS to connect to and also, multiple service providers which are offering various services over the available networks. WiMAX networks are required to support either manual or automatic selection of the appropriate network, based on user preference. The solution proposed by the WiMAX standard to facilitate these operations consists of four main procedures: NAP discovery, NSP discovery, NSP enumeration and selection, and ASN attachment.

NAP discovery: According to this process, an MS discovers all available NAPs within a wireless coverage area. The MS scans and decodes the DL-MAP of ASNs on all

detected channels. The most significant 24 bits of the “Base Station ID” value (within DL-MAP message) are used as the "Operator ID", which is actually the “NAP Identifier” (NAP-ID). During NAP discovery and in order to increase the efficiency of the procedure, information including previously detected and retained values, and/or other stored information such as channel, centre frequency etc could be useful to the MS.

NSP discovery: This procedure enables the MS to discover all NSPs that provide service in a given ASN. So, in addition to NAP ID, an MS is also provided with a list of available NSP Identifier (NSP-IDs) in order to have a complete view and information on the network prior to making a selection decision. NSP-IDs is a unique 24-bit identifier or 32-byte NAI and may be formatted as a globally assigned identifier or a combined MCC+MNC.

The MS discovers the NSPs by listening to a unsolicited, periodic transmission of NSP IDs broadcast by the ASN during initial scan or network entry. NSP-IDs may be also sent by the BS on demand by MS. Alternatively, the MS could maintain a configuration with a list of NSPs. The list of NSP IDs SHALL contain only NSPs that are either directly connected to the NAP's network or have a direct business relationship with NAP.

In case of NAP+NSP deployment (i.e. a NAP managed/owned by a single NSP), there is not need for separate provision of an NSP-ID. NAP-ID is sufficient.

If the configuration list in the MS does not match the NSP-IDs in the network broadcast, the MS should get the list from the network.

The NSP-IDs received from the detected networks are mapped to an NSP realm either by using stored configuration information in the MS or by making a query to retrieve it from the ASN.

NSP enumeration and selection: An NSP Enumeration list produced after NSP discovery process and is used by the MS for both automatic and manual NSP selection. In the former case, the MS selects an NSP from the list, based on information obtained from the service area and stored configuration. In the latter case, the list is presented to the user for selection and seems useful for a user to check its connectivity with a detected network or be serviced in a "pay for use" manner.

ASN attachment: Once an NSP is selected, the MS indicates its selection by attaching to an ASN associated with the selected NSP and by providing its identity and home NSP domain in the form of a NAI.

An MS indicates its NSP selection by attaching to an ASN associated with the selected NSP, and by providing its identity and home NSP domain in form of NAI. The ASN uses the realm portion of the NAI to route AAA transactions for the MS. The MS SHALL use a decorated NAI with additional (IETF RFC 4282) when the information need to be routed via another mediating realm (e.g., a visited NSP).

Configuration information: As stated before, configuration information stored in MS is used to assist network entry discovery and selection and should include, among others, items as follows:

- Ordered, prioritized NAP and NSP Identifiers Lists for both manual and automatic network selection
- NAP/NSP mapping list indicating the supported NSPs per NAP.
- NSP Change Count indicating whether the list of supported NSPs per NAP is changed
- Mapping table between 24-bit NSP identifiers and corresponding realm of the NSPs
- Physical information useful in NAP Discovery including channel, centre frequency, and PHY profile
- Security parameters related to ASN attachment phase e.g. user credentials for authentication with a NSP purposes.
- Network deployment mode indicating NAP+NSP mode or NAP sharing mode.

2.2.5 Authentication and Security Architecture

The WiMAX authentication and security architecture is designed to support all the IEEE 802.16e security services, using an IETF EAP-based AAA framework. In addition to authentication, the AAA framework is used for service flow authorization, QoS policy control, and secure mobility management.

2.2.5.1 AAA Architecture Framework

The WiMAX Forum recommends using the AAA framework based on the pull model, interaction between the AAA elements as defined in RFC 2904. The pull sequence consists of four basic steps.

1. The supplicant MS sends a request to the network access server (NAS) function in the ASN.
2. The NAS in the ASN forwards the request to the service provider's AAA server. The NAS acts as an AAA client on behalf of the user.
3. The AAA server evaluates the request and returns an appropriate response to the NAS.
4. The NAS sets up the service and tells the MS that it is ready.

Figure 13 shows the pull model as applied to the generic WiMAX roaming case. Here, steps 2 and 3 are split into two sub-steps, since the user is connecting to a visited NSP that is different from the home NSP. For the non-roaming case, the home CSN and the visited CSN are one and the same. It should be noted that the NAP may deploy an AAA proxy between the NAS(s) in the ASN and the AAA in the CSN, especially when the ASN has many NASs, and the CSN is in another administrative domain. Using an AAA proxy enhances security and makes configuration easier, since it reduces the number of shared secrets to configure between the NAP and the foreign CSN.

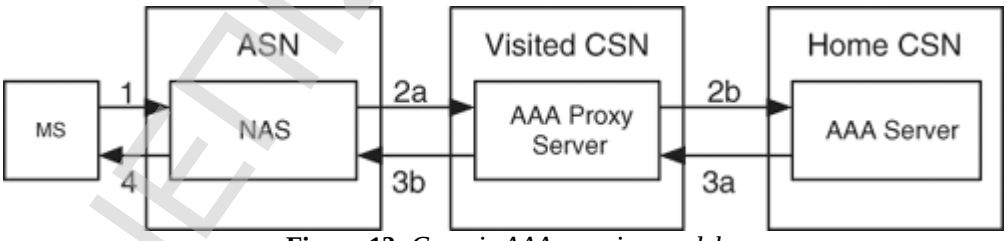


Figure 13: *Generic AAA roaming model*

In the WiMAX architecture, the AAA framework is used for authentication, mobility management, and QoS control. The NAS is a collective term used to describe the entity that performs the roles of authenticator, proxy-MIP client, foreign agent, service flow manager, and so on. The NAS resides in the ASN, though the implementation of the various functions may reside in different physical elements within the ASN.

2.2.6 IP Address Assignment

IPv4 address management is referring to point-of-attachment (PoA) IP addresses. PoA addresses are dynamic and are delivered to the MS via DHCP. (Alternatively, the home CSN may allocate IP addresses to an ASN via AAA, which in turn are delivered to the MS via DHCP. In this case, the ASN will have a DHCP proxy function as opposed to a DHCP relay function).

The release 1 of the standard does not deal with overlapping IP addresses.

When an MS is an IP gateway or host, a PoA IP address is allocated to the gateway or the host, respectively. If the MS acts as a layer 2 bridge (ETH-CS), IP addresses may be allocated to the hosts behind the MS.

For fixed access, the IP address a) has to be routable in the CSN and ASN, b) must be allocated from the CSN address space of the home NSP and c) may be either static or dynamic. Static IP addresses may be assigned by manual provisioning in the MS or via DHCP, while dynamic IP addresses are based on DHCP. The DHCP server resides in CSN domain that allocates the PoA address. A DHCP relay exists in the network path to the CSN. The DHCP proxy may reside in ASN and retrieves IP host configuration information during access authorization (i.e. during AAA exchange).

For nomadic access, dynamic IP address assignment based on DHCP is assumed. The DHCP server resides in home or visited CSN domains. As in fixed case, the DHCP proxy may reside in ASN and retrieves IP host configuration information during access authorization (i.e. during AAA exchange).

For portable and mobile access, dynamic allocation from either the home or the visited CSN is allowed, depending on roaming agreements and the user subscription profile and policy. In case of Proxy-MIP, PoA IP address assignment (Mobile IP home address) is based on DHCP. In case of Client-MIP, IP addressing will be based on MIP [RFC 3344] instead of DHCP.

The functional entity residing in ASN may be either a DCHP relay or a DCHP proxy. DHCP relay is used when MS initiates DHCP and address need to be retrieved from DHCP server. DHCP Proxy is used when MS initiates the DHCP and address is available during authentication from AAA server.

3. WiMAX ASN Gateway Design

3.1. WiMAX ASN Gateway Data Plane Overview

This chapter presents a functional description of the ASN GW Data Plane, based on the WiMAX standards.

3.1.1 Data Plane Functionality

The main functions of the data plane subsystem are summarized as follows:

GRE encapsulation-decapsulation: GRE tunneling is used for the data traffic in the R6 and the R4 reference points. The WiMAX specifications state that:

- no checksum will be used in the GRE packets;
- the "Key" option is used for the Data Path ID;
- the "Sequence Number" option may be used for handover optimizations and that Packet Header Suppression (PHS) may also be used.

IP over IP encapsulation-decapsulation: IP over IP is the proposed tunneling method for data traffic over the R3 reference point. However it is stated that GRE can also be used in R3 without the "Key" and "Sequence Number" options.

Packet classification: Several types of classification are needed for the traffic that the data plane will handle. The classification can be actually performed in different stages. In the first stage for example the control traffic can be identified from the data traffic, based on the fact that control messages are not encapsulated and use the UDP protocol with specific ports. The second stage can perform GRE (or IPoIP) classification, and the third stage can perform inner packet classification (both for the IP and the Ethernet convergence sublayer).

QoS Marking: The data traffic will be passed through a quality of service module to mark the packets so as to allow QoS discrimination per flow.

Traffic management: Traffic will be switched/routed between MS (R6), CSN (R3) and other ASN GWs (R4). The ASN GW plays the role of the FA in order to support MIP. The WiMAX specification states that the possible granularities can be per

service flow, per MS, or per BS. The granularity of the tunnels is defined and handled by the Data Path function located in the control plane.

Accounting support: User data packet or octet counting will be supported. The counting can be performed per user service flow. Counting of control packets or octets is optional.

Handover support: The data plane can (optionally) provide support for handover optimizations. This includes multicasting and buffering.

Security: R6 and R4 have an end-to-end secure channel, including privacy. The channel security may be implemented using physical security, IPsec or SSL, VPNs, etc. R3 may not have an end-to-end secure channel. In this case, lower layers are assumed to be insecure and the signaling protocols and data traffic provide their own security when needed.

3.2 WiMAX ASN Gateway Control Plane Overview

3.2.1 Control Plane Functionality

3.3.2.1 General description

The WiMAX network reference model defines multiple profiles for the ASN in order to accommodate varying network operator requirements and the vendors' preference for different network architectures. Interoperability among ASN elements (BSs and ASN GW) is supported among all products that comply with the specifications for the same ASN profile. Each ASN profile calls for a different decomposition of functions within the ASN. ASN profile B calls for a single entity that combines the BS and the ASN GW. Profiles A and C split the functions between the BS and the ASN GW slightly differently, specifically functions related to mobility management and radio resource management.

Profiles A and C both use a hierarchical model with a topology similar to that used in cellular networks and that is well suited to support full mobility. In profile A, the handover function is in the ASN GW; in profile C, it is in the BS, with the ASN GW performing only the handover relay function. Also, in profile A, the radio resource controller (RRC) is located in the ASN GW, allowing for RRM across multiple BSs. In profile C, the RRC function is fully contained and distributed within the BS.

The separation of the radio functionality and network management residing in the ASN gateway, native in profile C, facilitates inter-vendor interoperability as it allows network operators to select a different vendor for each function and so avoid conflicts and duplications. Furthermore, profile C does not require a separate ASN gateway for the radio management functions and thus, gives the opportunity to fixed operators to use existing access servers (e.g. BRAS) instead of deploying an ASN GW gateway.

In general, the ASN gateway, through its control plane functionality,

- Ø provides ASN location management and paging
- Ø acts as a server for network session and mobility management
- Ø performs admission control and temporary caching of subscriber profiles and encryption keys

- Ø acts as an authenticator and AAA client/proxy, delivering RADIUS/DIAMETER messages to selected CSN AAA
- Ø provides mobility tunnel establishment and management with BSs
- Ø acts as a client for session/mobility management
- Ø performs service flow authorization (SFA), based on the user profile and QoS policy
- Ø provides foreign-agent functionality
- Ø performs routing (IPv4 and IPv6) to selected CSNs.

Functional Category	**Related Standards**	**Function**	**ASN Entity Name**		
			Profile A	**Profile B**	**Profile C**
Security	WiMAx architecture, Stage 2, Stage 3, IEEE 802-16e, RFCs 2904, 4017, 2104, 3748, 2716, 4187, 2759,4282, 2138, 2869 , 3579 , 3588 .	Authenticator	ASN GW	ASN	ASN GW
		Authentication relay	BS	ASN	BS
		Key distributor	ASN GW	ASN	ASN GW
		Key receiver	BS	ASN	BS
Mobility	WiMAx architecture, Stage 2 ,Stage 3, IEEE 802-, RFCs 3344, 3024, 3012, 2794, 2131, 2132, 3046,3993, 3542 .	Data path function	ASN GW & BS	ASN	ASN GW & BS
		Handover control	ASN GW	ASN	BS
		Context server and client	ASN GW & BS	ASN	ASN GW & BS
		MIP foreign agent	ASN GW	ASN	ASN GW
Radio resource management	WiMAx architecture, Stage 2], Stage 3, IEEE 802].	Radio resource controller	ASN GW	ASN	BS
		Radio resource agent	BS	ASN	BS
	WiMAx architecture,	Paging agent	BS	ASN	BS

Paging	Stage 2, Stage 3, RFC 3344, (RFC 966])	Paging controller	ASN GW	ASN	ASN GW
QoS	WiMAx architecture, Stage 2, Stage, IEEE 802.16e, RFC 2748.	Service flow authorization	ASN GW	ASN	ASN GW
		Authenticator	ASN GW	ASN	ASN GW
		Service flow manager	BS	ASN	BS

Table 3: *ASN functional entities*

Accordingly, in the following Figure 14, a generic block diagram for the ASN GW control plane is shown. The block diagram proposes a possible decomposition of the main ASN GW control plane functionality into various software modules. Note that this block diagram has been constructed in a profile agnostic manner.

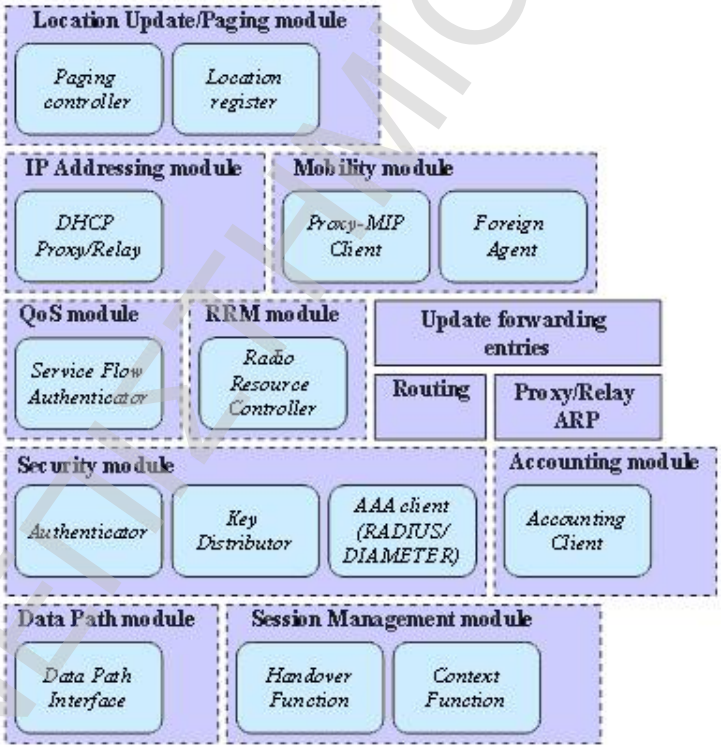


Figure 14: *ASN GW Control Plane Block diagram*

3.3.2.2 Description of the ASN GW Control Plane blocks

In the following, the modules that comprise the ASN GW control plane block diagram depicted in Figure 14 are described. These modules residing in ASN GW participate in one or more WiMAX control functions.

Data Path Module

The Data Path module will be responsible for setting up user data bearer in user data path and track session information on this path. It will control the traffic and may also support both IP and Ethernet convergence sub-layers for bearer. This module will be also responsible for managing (creating/deleting) GRE tunnels in the respective R6/R4 reference points and accordingly, switching traffic from GRE tunnels to IPoIP (MIP) tunnels. It will be also used to support handover functionality in terms of relaying paths to other ASN GWs and might be also used to render optimized handovers by offering bi-casting and/or buffering capabilities.

Session Management Module

The Context Function within Session management module will be responsible for creating, deleting, modifying and reading MS context (AK context, Network context, MAC context etc). It will be also responsible for transferring contexts to/from ASN GWs/BS. MS context will be kept in failure safe memory areas.

The Handover Function on the other hand will control handovers in both micro (ASN anchored) and macro (CSN mobility cases). In the micro- case, it will be responsible for selecting the target BSs from a candidate list, propagated via HO messages, negotiate the HO and finally select the target BS. It also collaborates with Data Path function for configuring the user data path in case bicasting and/or buffering capabilities are offered to optimize the HO procedure. Furthermore, in the macro-case, this module must be able to handle probable FA re-anchoring (Relocation of FA with updated CoA).

Security Module

This module implements functionality pertinent to authentication and security. It contains Authenticator, Key Distributor and the AAA client. The Authenticator is actually the entity defined in the EAP specifications as the end of the link initiating EAP authentication during MS network entry. The Key distributor is responsible for holding security keys, either fetched or locally derived during the EAP exchange. AAA client is responsible for the communication with the AAA server in the CSN over the respective AAA protocol, thus it also has the responsibility to map the Authentication relay protocol to AAA protocol for transfers to AAA server.

The security module may support multiple types of authentication e.g. Device/User/Both Device&User authentication however, only User authentication might be adequate since, according to the current release of NWG specifications, device authentication is an optional procedure.

Accounting Module

This module will be responsible for the accounting procedures based in RADIUS protocol. It will offer both Offline (post-paid) and Online (pre-paid) Accounting: In Offline accounting user will not be charged while he is being served, but duration based charging information are generated and stored in the form of UDRs In Online accounting user is continuously monitored against its remaining credits provided by HAAA server. Once he consumes its prepaid balance, he must be denied to be served.

Routing

This module may be used to facilitate the packet forwarding by populating various tables like L3 routing table, multicast table etc. Routing tables will be used for routing packets into NAP and CSN if L3 transport is used for R3, R4 and R6 interfaces.

Proxy/Relay ARP

Proxy/Relay ARP module is responsible for MS MAC address hiding. In this way, ASN GW is configured to act on behalf of MS and reply to the external ARP requests for MS or relay ARP requests generated by MS.

Update forwarding entries

This module is responsible for updating (adding/modifying/deleting) the forwarding entries stored in the form of a CAM in the switch board. The control plane through this module is aware of which data plane blade serves each active node and may fill the respective CAM in the switch board with the necessary entries. When events like network exit or handover occur, it deletes or changes the entries as required.

QoS module

The main part of the QoS module will be the Service Flow Authenticator either Serving or Anchor and will be responsible for admission control based on local policies and resource availability, for handling (creating/modifying/deleting) pre-provisioned and dynamic service flows, for acquiring (through AAA client) policies stored in the HAAA, for performing policy enforcement using a local policy database and an associated local policy function (LPF) and for data path (pre)registration/(pre)deregistration in association with Handover Function.

RRM module

The RRM module will be represented by the Radio Resource Controller entity and will be present only when profile A is considered. This module will be responsible for collecting radio resource indicators from the controlled BSs , to maintain a local radio resource database and accordingly, to assist WiMAX functions which impact radio resources e.g. QoS, service flow admission control, mobility management etc.

IP addressing module

This module will be responsible for managing the IP address allocation for MS. This module implements DHCP Relay or a DHCP Proxy and uses them depending on the way that the IP address is obtained by MS. DHCP Relay is used when address needs to be retrieved from DHCP server in the CSN, whereas DHCP Proxy is used when the address has been already retrieved from AAA server during authentication. DNS establishments will be also supported by this module. The IP addressing module will be also interfacing with the mobility module especially for coordinating MIP registration and IP address assignment in the portable/mobile access scenario.

Mobility module

The mobility module implements Proxy-MIP and Foreign Agent functionality in order to support mobility of the MS in multiple access networks including roaming situations. This module is responsible for advertising MIP to support Client-MIP, for updating mobility context related to session management module. This module is also interfacing with IP addressing module as stated before where, Proxy-MIP client is responsible for MIP registration and de-registration with HA.

Location update / Paging module

This module will be responsible for paging and idle mode operation of MS. It will implement the Paging Controller entity that handles one or more paging groups, connects to other paging controllers in the network, triggers the paging agents in BS(s) and updates location register whenever MS moves in/out of IDLE mode or when MS updates its own location. The Location Register will be a distributed database containing information about Idle mode MSs.

3.3 WiMAX ASN Gateway Management Plane

3.3.1 Management Plane functionality

The management plane will be responsible for managing the ASN GW. It will allocate functionalities into management functional areas divided as per ITU-T X.700 standard in Fault, Configuration, Accounting, Performance and Security management, also known as FCAPS. The way in which the ASN GW management is distributed around those functional areas is described in the following sections (see also Figure 15).

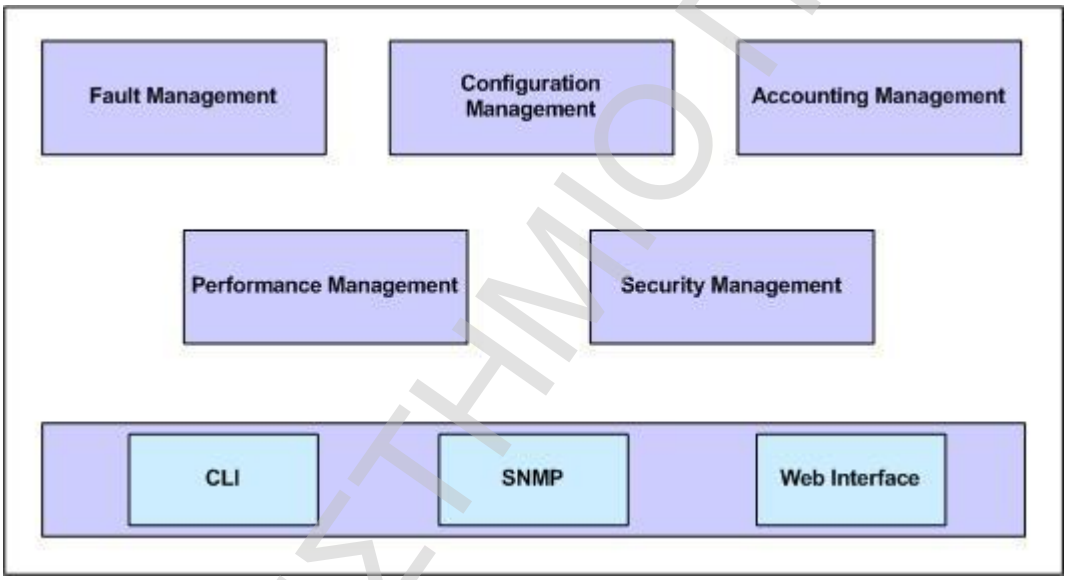


Figure 15: *ASN GW Management Plane Block diagram*

Fault Management

Fault management will include the development of functions that address alarm surveillance, testing and fault isolation. Alarm surveillance allows the reporting alarms with different levels of severity along with the possible cause of alarm. Testing will be needed for tracking the progress and retrieving/reporting the results. Fault detection and isolation functions permit the managing system to use techniques such as alarm correlation and diagnostics testing to determine the location and root cause of the fault so that necessary corrective action may be taken. Special care will be

devoted to the case of failover, where appropriately defined actions will manage to transfer the ongoing load to the redundant blade(s).

Configuration Management

Configuration management functions, as the name reveals, will be responsible for the starting-up and configuration of managed components of the ASN GW. They will manage availability on resources and services and also monitor and control their state and status information. Configuration management will have the task to add/delete/modify/reset resources and their parameters, and to properly configure the available physical/logical interfaces, ports, temperature thresholds and/or other system specific parameters e.g. DHCP server, AAA server etc. Last but not least, software upgrade processes will be also controlled by the configuration management functionality.

Accounting Management

Typically, this functional area includes collecting usage data for the resources used in providing a service, prior to generating an invoice by applying a tariff associated with the service. Such management function is judged as needless and will be not explicitly considered in the framework of this study. However, minor accounting management functions may be implemented for statistical and logging purposes.

Performance Management

Performance management related functions will address issues concerning the monitoring of performance parameters e.g. delays, errors etc, the collection and processing of such resource performance information and the derived data statistics for obtaining insight to the behaviour and effectiveness of the managed resources. Setting proper thresholds for the different measured parameters and determining the alarms that will be produced in case of threshold crossing will be also developed. Performance management functions will be also responsible for applying control mechanisms to prevent traffic congestions e.g. by balancing the load among the data (control) plane blades.

Security Management

Security management will address issues concerning the controlled access to system resources, facilities and information and the protection of the managed system against intentional or accidental abuse, unauthorized access, and communication loss. The provisioned security management functions will be able to manage user authentication, to support accounts including the option to give variant access rights into specific groups of users and to handle the details (rules) of the access. Security auditing will specify the type of event reports that should be contained in a log that is to be used for evaluating the security mechanisms and their performance. Specific security alarm notifications need to be defined for informing about security violation attempts, along with parameters and semantics.

3.3.2 Data Model

The ASN GW node organizes and maintains all the available managed objects in a so-called configuration tree. A configuration agent lying within the ASN GW is responsible for advertising this structure to the configuration clients, thus rendering the ASN GW manageable.

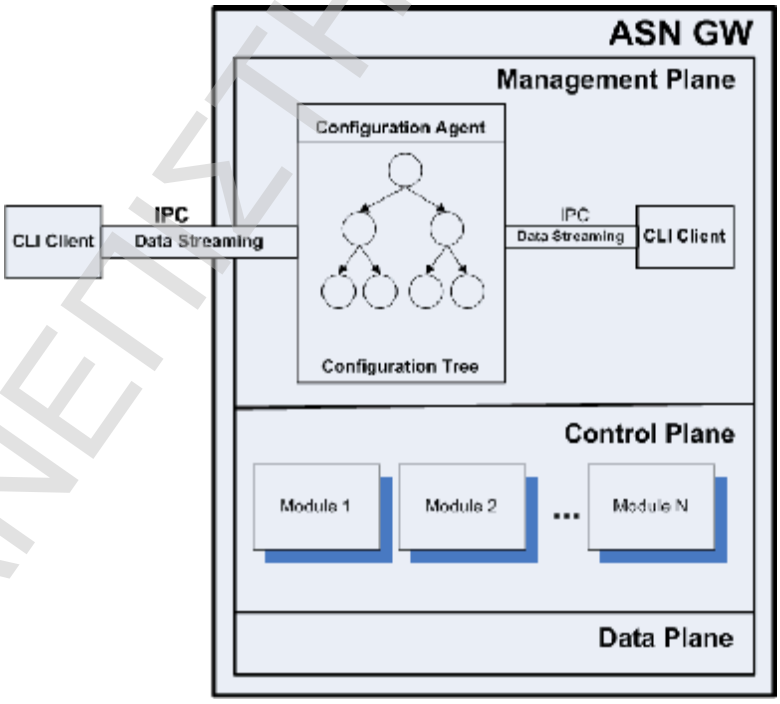


Figure 16: *ASN GW Architecture Design*

In such a hierarchical tree structure all nodes can be uniquely addressed and accessed by specifying the exact path resulting to that node, starting from the root. The configuration tree actually defines the management view of the ASN GW, comprising of several managed objects. A managed object is actually a subtree within the configuration tree. Such subtrees comprise each of a set of nodes which are related in some way. A managed object may also consist of one single node.

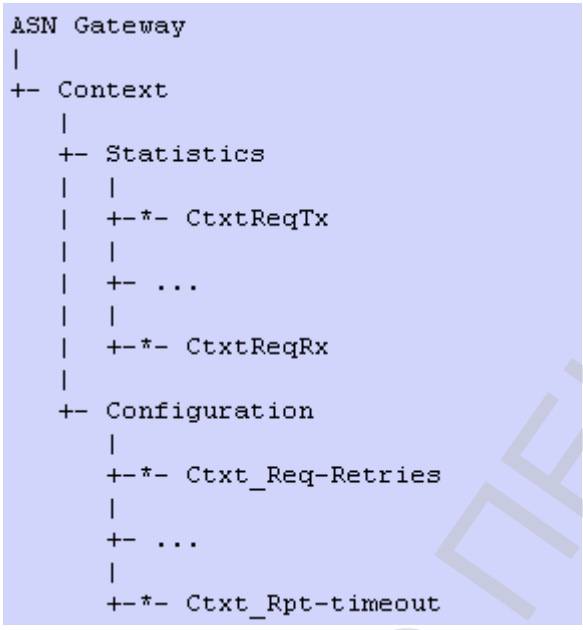
A portion of a typical configuration tree might look like this:

```
Root
|
+- ASN Gateway
|
+- Host Config
|
+- ACLs
|
+ ... (etc) ...
```

Under the ASN Gateway node the managed objects can be organized like this:

```
Root
|
+- ASN Gateway
   |
   +- Authentication
   |  |
   |  + ... (more nodes) ...
   |
   +- MIP
   |
   +- PMIP
   |
   +- DHCP
   |
   +- Context
   |
   +- Data Path
   |
   +- AAA Client
   |
   +- Paging
   |
   +- Hand-Over Relay
   |
   +- ... (more nodes) ...
   |
   +- RRM Relay
```

Under the Context node, the subtree might look like this:



None of these example trees depict the exhaustive list of nodes, managed objects, or parameters albeit it can be easily perceived that the complete list will comprise a set of nodes that are needed to manage the functional modules of the ASN GW control plane.

A node is the basic element in the configuration tree and it represents the managed objects that can be manipulated by various management actions. Two kinds of nodes are considered in the configuration tree: "*Interior nodes*" (or simply "*nodes*") and "*leaf nodes*" or just "*leaves*". An interior node can potentially have an unlimited number of child nodes (interior and leaf), but cannot store any value. On the other hand, a leaf doesn't have any child nodes, but represents parameters and stores values for those parameters. In addition, the node in which leaves are attached is called the "edge node" for these leaves.

In the example of the context subtree, above, the *Context* node is an interior node, as so are the *Configuration* and *Statistics* nodes. Under the *Configuration* node, leaves for the appropriate parameters are attached and in the same manner, the *Statistics* node holds the leaves pertinent to statistical values. The *Statistics* and the *Configuration* nodes are edge-nodes for their respective leaves.

Every leaf node has a "*name*" and a "*type*". The leaf's type controls the kind of value it can store. Leaf types will be defined such as: *text*, *integer*, *timestamp*, *IP address*, *MAC address*, *binary object*, *reference*, *pointer*, etc. Every interior node has also a "*name*" and belongs to a "*class*" or "*type*" which specifies:

- The names and types (classes) of interior nodes it can have as children (if any), as well as whether any of them are mandatory.
- The names and types of leave nodes it can have as children as well as if any of them are mandatory
- A set of optional class-specific operations (hooks) pertinent to the node and corresponding to the tree manipulation primitives. These operations allow the definition of class-specific actions to the tree-manipulation primitives. Specifically hooks for the following primitives will be provided: validate, create, delete and possibly other. The role of these node-type-specific operations will be discussed below.

In addition, every interior node has two sets of access-control flags associated with it. One set regulating read-access and the other regulating write access. Their role will be discussed below.

Note

Special performance issues apply to leaf nodes holding statistics counters. While accessing and retrieving the value of a statistics counter is an operation not unlike the value-retrieval from other leaf nodes, updating a statistics counter is an operation that occurs (or can occur) very often (several times per second, or even once for every processed packet). Special optimizations for updating these nodes will be considered and discussed in a future release of this document.

3.3.3 Detailed Node Type Definition

Every interior node is associated (belongs to) a Node Type. Node types (or Classes) are specified using Node Type Definitions with one or more Node Type Definitions. A Node Type Definition has the following attributes:

- **Name:** Every node type registered has a unique name.
- **Leaf Node Definitions:** A node type contains a set of definitions. Specifying the leaf nodes that nodes of this node type are allowed (or required) to have and the characteristics of this nodes
- **Child Node Definitions:** A node type contains a set of definitions specifying the child nodes (internal nodes) that nodes of this node type are allowed (or required) to have and the characteristics of those child nodes (including, in turn, their node types)
- **Operations:** A set of optional class-specific operations (hooks) pertinent to the node and corresponding to the tree manipulation primitives as described above.

Leaf Node Definition:

A leaf node definition (within a node type definition) contains the following information:

- **Name:** The name of the leaf node.
- **Type:** The type of the leaf node. One of
    - STRING (a sequence of characters)
    - BINARY OBJECT (an unstructured data buffer)
    - INTEGER (32 bit unsigned value)
    - IP ADDRESS (32 bit value, most significant first)
    - MAC ADDRESS(48 bit value, most significant
    - PATH (a valid absolute or relative path leading to a Configuration tree node)
- **Default Value**
- **Mandatory Status:** When a leaf node has the mandatory property set, it means, that the leaf node must exist in order for the tree to be valid.

Child Node Definition:

A Child node definition (within a node type definition) contains the following information:

- **Name:** Every node type registered has a unique name.
- **Type:** It specifies the type of the interior node.
- **Mandatory Status:** When a Child node has the mandatory property set, it means, that the Child node must exist in order for the tree to be valid.

Example of Node Type Definitions:

```
**Node Type:**
       *Name:* "Interfaces"
       *Leaf Node Definitions:* <empty>
       *Child Node Definitions:*
                  **Name:** "eth*"
                  **Type:** "Ethernet_If"
                  **Mandatory Status:** "Not_Mandatory"

                  **Name:** "PPP*"
                  **Type:** "PPP_If"
                  **Mandatory Status:** "Not_Mandatory"
       *Operations:* <empty>

**Node Type:**
       *Name:* "Ethernet_If"
       *Leaf Node Definitions:*
                  **Name:** "IP_Address"
                  **Type:** "IP_ADDRESS"
                  **Default value:**"No"
                  **Mandatory Status:** "Mandatory"
```

3.3.4 Relational Database

Our configuration tree is manipulated in memory and will be saved for maintaining persistence. Two different ways for saving this hierarchical data structure (configuration tree) will be provided and are described as follows:

- **File-backed storage mode:** in this mode, the configuration tree will be saved in a file in a hard disk. The save procedure starts whenever the agent terminates its operation or alternatively, whenever a user requests for saving the tree in a file under a specific name. A restore procedure will be also provided so that the user can restore the configuration tree from a file.
- **Relational-database storage mode:** in this mode, the configuration tree will be stored in a relational database. The reason behind this choice stems from the requirement to perform complex queries to the configuration tree, a possibility that is natively facilitated by an RDBMS.

In this database we can represent the structure of the nodes, leafs and their values. When the agent starts, the configuration tree is loaded from this database. The database will always be updated. This update procedure of the database is not synchronous with the changes in the configuration tree. When we change the content of the configuration tree with the primitives, we also have to change the content of the database. The procedure, starting by client issuing a primitive and ending with the application of the primitive to both the configuration tree and to its relational representation in the database. This procedure might look like this:

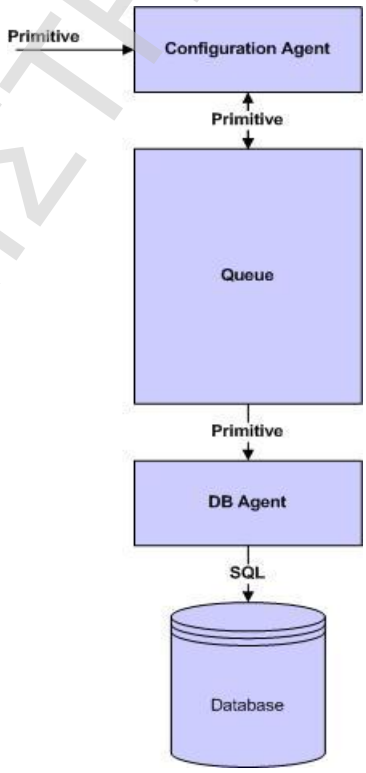


Figure 17: *Relational-database storage mode Procedure*

When the primitive is sent to the configuration agent, it performs the changes to the in-memory tree and pushes the primitive in a queue to be served by a special asynchronous entity (e.g. thread) inside the configuration agent called the "database agent". The database agent is responsible to perform the changes in the database. Many primitives derive from different sources as the configuration agent can be connected with different clients. The database agent must know from which source the primitive comes from.

We will use transactions. With transactions, we have the ability to execute a set of queries as a unit. The results of a transaction can be committed or rolled-back. Each primitive has its own transaction and as a result we can secure that the changes in the database will be atomic as in the case of the configuration tree. The configuration tree and the database must be synchronized when the agent starts. The tree is also synchronized when the database agent fails to process and commit a primitive. In this case, the queue is discarded and the configuration tree and the database are synchronized. After the synchronization, the configuration agent can continue receiving primitives from the clients. Although this synchronization process is time consuming and complex, we do not expected to happen often.

The configuration agent maintains multiple sessions to the database. This is necessary for processing locking operations *where by* several primitives must be executed and committed to the database atomically. This is done by performing the actions for these primitives within their own transaction which coexists with transactions used for processing the primitives outside the lock context.

In our database we store all the nodes of the configuration tree. Our database will have one table for all the nodes of the configuration tree and one table per Node-Type for each Edge Node of the configuration tree. One possible structure of the tables in the database is the following:

Figure 18: *Relational Database Structure of Tables*

The "Nodes" table will have as columns: (id, pid, name, path, class, read flags, write flags):

- The id is a unique identifier of each interior node in the configuration tree.
- The pid is the id of the parent node.
- The name defines the name of each node.
- The path is the absolute path of the node in the tree. It is optional but it is convenient if you want to find a node quickly.
- The class is the interior's node class.
- The read flags and the write flags define the access control flags which are associated with the node.

We will define one table per Edge Node Type. In the "Edge Node Type" table, we have one row for every leaf in the tree that has as a type the "Edge Node Type".

Each table of these will have as columns: (id, leaf_id, name, path, mandatory, default_value, value):

- The id is the unique identifier of the Edge node.
- The leaf_id is the unique identifier of each leaf of the Edge node.
- The name defines the name of each leaf node.
- The path is the absolute path of the leaf node in the tree. It is optional but it is convenient if you want to find a leaf quickly.
- The manatory column defines is the leaf node is mandatory or optional.
- The default_value column defines the default value of the leaf node.
- The value is the value of the leaf node.

In the "Nodes" table, "id" is the primary key and "pid" is a foreign key to the "id" attribute of the same table. In the "Edge Node Type" table, the "id" attribute is a

foreign key to the "id" attribute of the "Nodes" table and the "leaf_id" is the primary key. The relations between the tables may look like this:

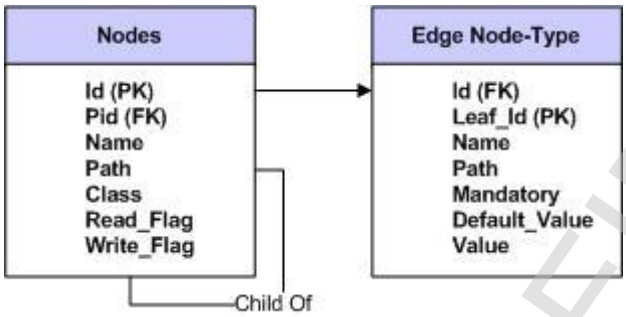


Figure 19: *Relations Between the Tables*

3.3.4.1 Application of primitives to the database

The configuration agent will push in the queue only the write primitives. All write primitives are executed atomically within a transaction. The configuration agent translates the configuration tree primitives to database primitives understood by the database agent- before insert them to the queue. A tree manipulation primitive can correspond to one or more database primitives. For example the create tree manipulation primitive, is translated to one "insert" database primitive and possibly to several update database primitives. On the other hand the modify tree manipulation primitive is translated to a single "update" database primitive. The database primitives and their operation in the database is the following:

Database primitive Insert

The configuration agent passes to the database agent the following information: the id of the parent node, the name of a new node, the read flags and the write flags. With these attributes the database agent can add a new row (new node) in the "Nodes" table. Also, it can add the read and write flags in the "Nodes" table.

Database primitive Delete

The configuration agent passes to the database agent the following information: the list of nodes ids in the subtree we want to delete. With these attributes the database agent can remove one by one all the ids from the "nodes" table. After the delete from the "nodes" table, the database agent also deletes the respective rows from the other tables according to the type of the node. In case it does not provide the list of the ids, the configuration agent passes instead the id of the subtree's root node. When the configuration agent passes only the pid we must use recursive queries to find the ids that we want to delete from the database.

Database primitive Update

The configuration agent passes to the database agent the following information: the id of the edge node, the ids of leafs and their values. With these attributes the database agent can change the value of leafs according to the leaf-types (e.g. leaf-integer, etc.).

Database primitive Set-flags

The configuration agent passes to the database agent the following information: the id of the node, the read flags and the write flags. With these attributes the database agent can change the access control (read flags and write flags) columns in the "interior nodes" table.

3.3.5 Redundancy and fault tolerance

Fault tolerance will be ensured by incorporating redundancy in the designed system. Specifically, the configuration agent/tree will be duplicated with one agent (tree) playing the role of the "master" and a second one playing the role of the "slave", respectively. The design can be also extended to include N slave agents as replicas of the master one. The "slave" agents replicate the “master” tree in a synchronous manner. As a result, the procedure of storing configurations will adhere to a simplified model of a two-phase commit protocol, which can be described as follows:

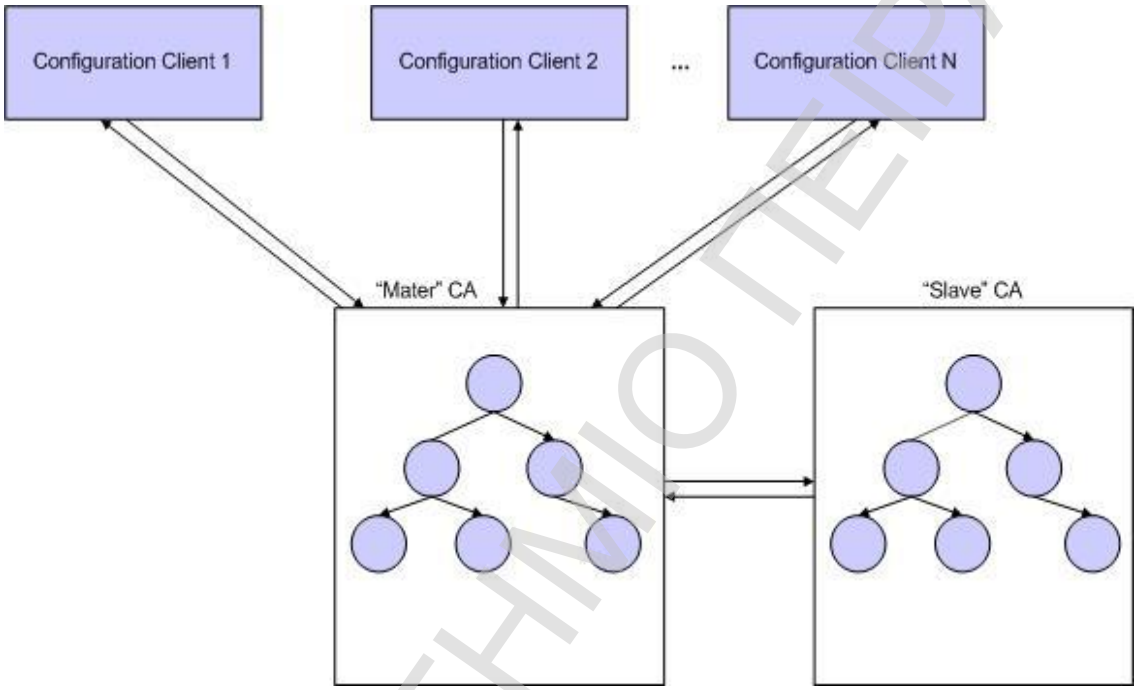


Figure 20: ***Redundancy and fault tolerance Model***

The configuration agent receives a primitive from the configuration client. It applies the primitive to the "master" configuration tree and the changes are successful. The "master" configuration agent makes the changes to the "master" configuration tree but it does not commit them. It will commit the changes only if it will receive a "success" message from the "slave" configuration agent. Then, the "master" configuration agent sends the primitive encapsulated in a message to the "slave" configuration agent. The "slave" agent applies the primitive. If the changes are successful, it sends a "success" message to the "master" agent who commits the changes and the "master" configuration agent sends a "success" message to the configuration client. If the changes are failed, it sends a "failure" message to the "master" agent who does a roll-back to the previous state and the "master" configuration agent sends a "failed" message to the configuration client.

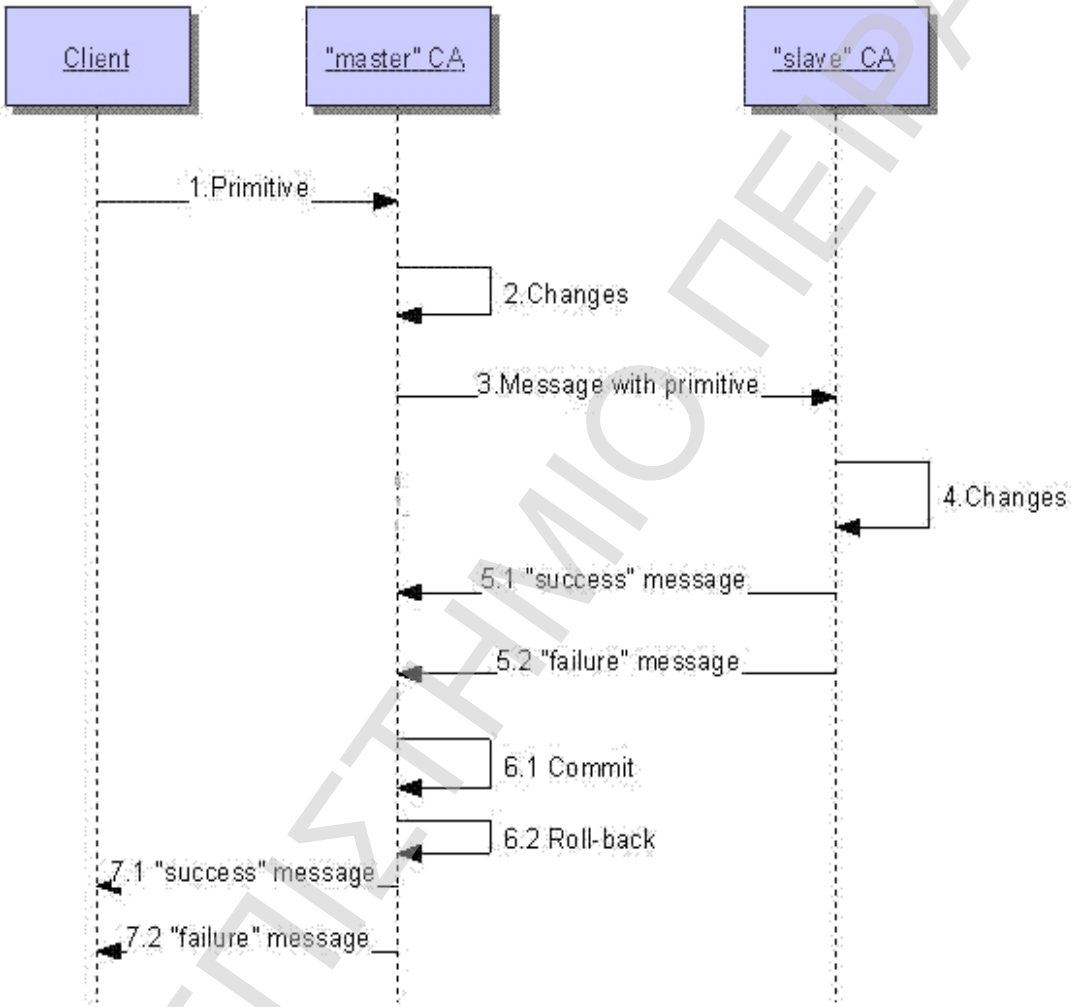


Figure 21: ***Master CA – Slave CA Synchronization Workflow***

As the two trees work in a synchronous manner, we ensure that in the end of every operation the two trees are in the same state and if an operation fails, it is applied to neither the "master" nor the "slave" configuration tree.

3.3.6 Security and Authorization

Special focus will be placed on managing access to the configuration tree according to specified privileges given to users and/or groups of users as well as to other entities accessing the configuration tree.

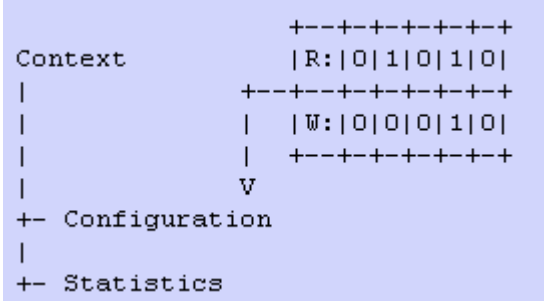
There are three Modes:

- An Agent Based Mode
- A Radius Based Mode
- A TACACS Based Mode

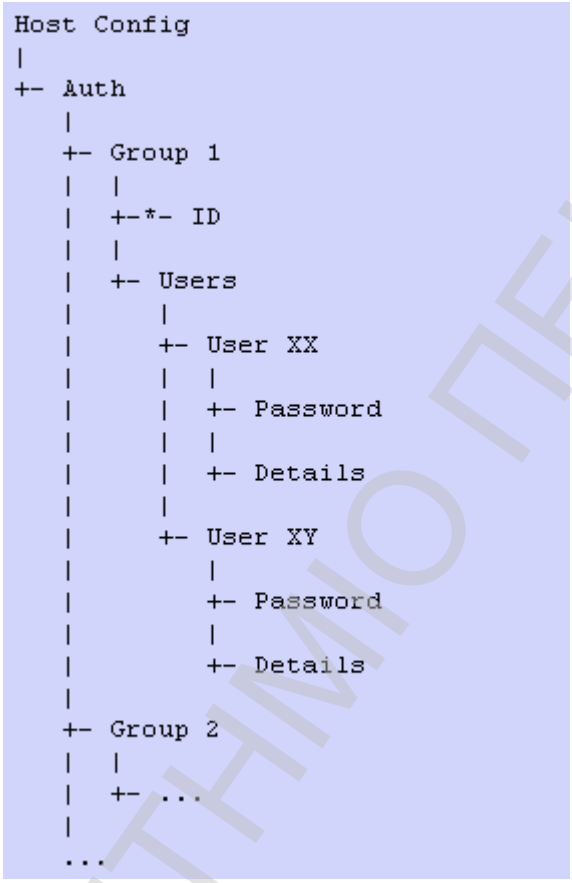
Agent Based Mode

Every entity (typically client) accessing the configuration tree is assigned, upon admission, a token used to regulate its access to the tree nodes. For most clients this token is assigned during the authentication phase when they supply their credentials to the agent. For special entities accessing the tree (such as local clients, or the agent itself) no authentication is required and the authorization will be done by assigning automatically a token upon admission. For performance reasons 32 two such tokens are defined and the token assigned to an entity is decided based on the entity's identity or the credentials supplied by it. More than one entity can be assigned the same token.

As mentioned above, every node in the tree (internal node) has two sets of access-control flags associated with it. The first set of flags (the read-set) governs which tokens are allowed to perform the Get, Describe, and Children primitives to it. The second set of flags (the write-set) governs which tokens are allowed to perform all the primitives on the node:



For the purpose of assigning tokens to credentials an Auth node will be included in the tree. One possible structure of the Auth node is the following:



There are the groups (each corresponding to one of the 32 tokens) under the authorization subtree. Under each group, there is an edge-node for each set of credentials (each user). The name of the edge node is the user name (login-name). Under these edge nodes there are password and details (real name or other information) leaf nodes for the user.

Radius Based Mode

In Radius Based Mode, authentication will take place externally and be provided by a RADIUS server, and the authorization will be provided locally based on two sets of access- control flags as in Agent Based Mode.

Every time the Configuration Agent receives a login request, it creates an "Access–Request" packet. This packet must contain username and password (mandatory attributes) of the configuration client which accessing the configuration tree. This packet is submitted to the RADIUS server. If no response is returned within the length of time, the packet is resent a number of times. Once the RADIUS server receives the "Access–Request" packet, it validates the configuration agent. If the configuration agent is valid, the RADIUS server consults a database to find the user whose name and password matches to the username and password in the "Access–Request" packet. If any condition is not met, the RADIUS server sends an "Access-Reject" response indicating that this user request is invalid. If all conditions are met, the list of configuration values for the user is placed into an "Access-Accept" response. For the external clients that use the RADIUS Authentication to access the configuration tree, a token is assigned through the Authentication phase. The client receives its token with the "Access-Accept" packet.

TACACS Based Mode

In TACACS Based Mode Authentication and Authorization will take place externally by using a TACACS server.

Authentication Phase

Every time the Configuration Agent receives a login request, it creates a "Start" packet. The "Start" packet describes the type of authentication to be performed, and may contain the username and some authentication data, such as password, privileged level, etc. In response to a "Start" packet, the TACACS Server sends a "Reply" packet. The "Reply" packet indicates whether the authentication has finished, or whether it should continue. If the "Reply" packet indicates that authentication should continue, then it will also indicate what new information is requested. The client will get that information and return it in a "Continue" packet. The TACACS Server will answer to the Configuration Agent with a "Reply" packet.

Authorization Phase

The Configuration Agent sends a "Request" packet that contains a fixed set of fields which describe the authenticity of the user or process. Also, the "Request" packet contains a variable set of arguments which describe services and options for which the authorization is requested. Every time the user sends a request (e.g. CLI command) to access the Configuration Tree, the Configuration Agent sends the "Request" packet with attributes which describe the authenticity of user, the type (write/read) of the primitives that corresponds to the user's request, and the path that describes to which Configuration Tree's Object the primitives should be applied. The "Response" packet contains a variable set of response arguments (attribute-value pairs) which can restrict or modify the user's access rights.

3.3.7 Tree manipulation Primitives

To ensure consistency and validity the manipulation-of and access-to the configuration tree (be it internally by the configuration agent itself, or externally by its clients) is always performed through a set of primitives. Primitives are applied to the configuration tree atomically. If their application fails its effects are rolled-back and the tree is left in a state as if the primitive was never issued. For reasons of access-control and locking the primitives are categorized in three classes: *read primitives, write primitives* and *auxiliary primitives.*

add (write primitive)

add <node> <name> [<read flags> <write flags>]

Create and attach a node named <*name*> as a child of the node specified by <*node*>. The new node is created with the access-control flags specified by <*read flags*> and <*write flags*>, or, if these arguments are missing, with the same access flags as the parent node. The type (class) of the attached node is deduced from the parent node and the child node's name. If the type cannot be deduced then the application fails. This operation may leave the tree in an invalid state, therefore the parent node (or an ancestor thereof) must be write-locked before performing it.

remove (write primitive)

remove <node>

Remove the subtree rooted at <*node*>. This operation may leave the tree in an invalid state, therefore the parent node (or an ancestor thereof) must be write-locked before performing it.

create (write primitive)

create <node> <name> <aguments> [<read flags> <write flags>]

Create and attach a node named <*name*> as a child of the node specified by <*node*>. The new node is created with the access-control flags specified by <*read flags*> and <*write flags*>, or, if the arguments are missing, with the same access flags as the parent node. The new node is created by calling node's class-specific create operation and passing it the arguments supplied in <*arguments*> this may result in the creation of additional nodes and leafs. The type (class) of the attached node is deduced from the parent node and the child node's name. If the type cannot be deduced then the application fails. This primitive is only applicable for node-types that have the create operation defined. After a create operation, the validate primitive is implicitly applied to the newly created node (or subtree), and if it fails the effects of create are rolled-back. This operation will always leave the tree in a valid state therefore it can be performed without locking the node.

delete (write primitive)

delete <node>

Remove the subtree rooted at <*node*>. Before deleting the node call the class-specific delete operation. This primitive is only applicable for node-types that have the delete operation defined. The delete class-specific operation is usually a nop-function serving as a place-holder to indicate that a specific node-class can be safely deleted. After a delete operation, the validate primitive is implicitly applied to the parent node (or subtree), and if it fails the effects of create are rolled-back (NOTE: this step may

be omitted). This operation will always leave the tree in a valid state therefore it can be performed without locking the node.

set (write primitive)

set <node> <leaf> <value>...

Used to set the values of one or more leaf nodes (parameters) of the edge node *<node>*. The values of one or more leafs can be set with a single primitive application. If the value for a leaf is NULL the leaf is removed. After a set operation, the validate primitive is implicitly applied to the edge node, and if it fails the effects of set are rolled-back. This operation will always leave the tree in a valid state therefore it can be performed without locking the node.

modify (write primitive)

modify <node> <leaf> <value>...

Used to set the values of one or more leaf nodes (parameters) of the edge node *<node>*. The values of one or more leafs can be set with a single primitive application. If the value for a leaf is NULL the leaf is removed. Unlike the set primitive after a modify operation, the validate primitive is not applied to the edge node. This operation may leave the tree in an invalid state, therefore the edge node (or an ancestor thereof) must be write-locked before performing it.

authset (write primitive)

authset <node> <read flags> <write flags>

Set the authorization flags of a node. This operation will always leave the tree in a valid state therefore it can be performed without locking the node.

get (read primitive)

get <node> [<leaf> ...]

Return the values of the leafs named by the optional arguments attached to the edge node <*node*>. If no leaf names are specified then return the values of all leafs.

validate (read primitive)

validate <*node*>

Perform validity checks on the node <*node*> to verify its conformance to the node's type (class). Then call the class-specific validate operation (if defined) to perform any additional checks. For every child node apply the validate primitive recursively. Return the result of the tests: "Node valid" if the all tests succeeded or "node invalid" if any of the tests failed.

children (read primitive)

children <*node*>

Returns the interior nodes that are rooted to the node <node>.

typeof (read primitive)

typeof <*node*>

Returns the type (class) of the node.

describe-type (read-primitive)

describe-type <*type*>

Return meta-information regarding a node-type (class) including: The name of the type, the allowed names and types (classes) of children nodes for a node of this type as well as if any of them are mandatory, the names and types of allowed leaf nodes, the expected signature of the create (and possibly other) class-specific operation(s) and so on.

lock-write (write primitive)

lock-write <node>

The lock-write primitive is used to obtain atomic access to the branch of the configuration tree rooted at <*node*>. Atomicity is required in order to prevent two different users from having simultaneous write access to the same node as well as to allow modifications to the tree in a "transactional" manner. This way the tree transforms from one valid state to the next as the result of the application of a set of primitives without being visible while in the intermediate (and possibly invalid or inconsistent) states. Any modifications (write-primitives) performed to the subtree after it has been write-locked will become visible to other entities only after the subtree is unlocked. While the subtree is write-locked by an entity it cannot be read-locked or write-locked by another unless it is first unlocked. Write primitives cannot be applied on the subtree by other entities while it's write-locked. Read primitives can be applied (but they will not "see" the effects of the write primitives applied by the entity holding the lock unless the lock is released).

Note

In order to keep the modifications to the subtree invisible to entities other than the one holding the write-lock, the subtree is effectively copied when the write-lock is acquired and subsequent write primitives are applied to the copy.

lock-read (read primitive)

lock-read <node>

The lock-read primitive is applied by an entity to a branch of the configuration tree in order to prevent other entities from acquiring a write-lock to the branch and from applying write primitives to it. Multiple entities can acquire the read lock to a subtree. The lock is considered released when all entities have released it. lock-release (read primitive) lock-release <node> Release the lock held on the branch rooted at node <*node*> held by the applying entity. If no lock is held by the entity the operation does nothing. If a read-lock is held, the lock is released and if no other entity is holding a

read-lock the subtree is unlocked. If a write-lock is held then: Perform a validity check on the subtree by applying the validate primitive. If the check fails, unlock the subtree and, at the same time, roll-back any changes performed since the subtree was locked (in effect: discard the modified copy). If the validity check succeeds then unlock the subtree and, at the same time, commit (make visible) all changes performed since the subtree was locked (in effect: replace the master subtree with the modified copy) .

register-notification (read-primitive)

register-notification <node> <channel-id>

Applied to the node <*node*> by an entity to inform the configuration agent that the entity wishes to be notified about changes performed to the subtree rooted on the node. The notifications will be sent the the channel identified by <*channel-id*>

save (auxiliary primitive)

save <file-name>

The save primitive is used for saving the configuration tree in a file with a specific name, whenever file-backed storage mode is utilized. The results that this operation may return are: "save succeed" if the save of the configuration tree was successful or "save failed" if the save failed. This operation will always leave the tree in a valid state; therefore it can be performed without locking in a node.

restore (auxiliary primitive)

restore <file-name>

It is used to restore the configuration tree from a file specified by the <file-name>, whenever file-backed storage mode is utilized. The results that this operation may return are: "restore complete" if the restore of the configuration tree was successful or "restore failed" if the restore failed. This operation will always leave the tree in a valid state; therefore it can be performed without locking in a node.

3.3.8 Command Protocol

The Configuration Agent communicates with external and internal (local) clients (or/and data plane modules), through a stream sockets. Both Internet-domain, and Unix-domain sockets are supported. Both Unix-domain and Internet-domain ports are simultaneously active. Unix-domain sockets are used by internal clients, and Internet-domain sockets are used by external clients. Three ports are provided: One Internet-domain port and Two Unix-domain ports:

- The Internet-domain port, where external clients can connect and send commands to the Configuration Agent using TCP.
- The Unix-domain self-authenticated port, where data plane modules can connect to the Configuration Agent in order to access the Configuration Tree.
- And the Unix-domain local port where local clients executing on the same machine can connect and send commands to the Configuration Agent.

General Packet Structure

The interaction between the clients and configuration agent is strictly stop-and-wait, meaning that the client must wait for a command to be responded before sending the next one. Request Packet Format

All the command packets sent from clients to Configuration agent share the same <COMMAND_PCK> packet format. A header consisting of the packet length encoded as a 4 bytes unsigned integer, followed by the body of the packet consisting of "packet length" octets. The first octet of the body is the command opcode, identifying the command type and the arguments depend on the command opcode (see **Command Operation Codes** below).

Formally:

```
<COMMAND_PCK> : { <HEAD>, <BODY> }
<HEAD> : {<PACKET_LENGTH>}
<PACKET_LENGTH> : The length of the packet (4 bytes)
<BODY> : {<OPCODE>, <ARGS>}
<OPCODE>: The command operation code (opcode), identifying the command type
<ARGS>: The arguments of the command, depending on the value of the
        <OPCODE> filed
```

The <BODY> of the <COMMAND_PCK> packet is analytically described in the "Detailed Description of Commands Format" section.

Response Packet Format

For all commands, there is a response <RESPONSE_PCK> packet format. A header consisting of the packet length encoded as a 4 bytes unsigned integer, followed by the body of the packet consisting of "packet length" octets. The first octet of the body is the response respcode, identifying the response type and the data depend on the respcode.

Formally:

```
<RESPONSE_PCK>: ( <HEAD>, <BODY> )
<HEAD>: (<PACKET_LENGTH>)
<PACKET_LENGTH>: The length of the packet (4 bytes)
<BODY>: (<RESPCODE>, <DATA>)
<RESPCODE>: The Response operation code (respcode),
<DATA>: The response data, depending on the value of the <RESPCODE> field
```

The <BODY> of the <RESPONSE_PCK> packet is analytically described in the "Detailed Description of Response Format" section.

Command Operation Codes (opcodes)

- 0x01 --> LOGIN
- 0x02 --> CREATE
- 0x03 --> ADD
- 0x04 --> REMOVE
- 0x05 --> DELETE
- 0x06 --> SET
- 0x07 --> MODIFY
- 0x08 --> AUTHSET
- 0x09 --> GET
- 0x0a --> VALIDATE
- 0x10 --> CHILDREN
- 0x11 --> TYPEOF
- 0x12 --> DESC_TYPE
- 0x13 --> LOCK_WRITE
- 0x14 --> LOCK_READ
- 0x15 --> REGISTER_NOTIF

Command Response Codes (respcodes)

- 0x00 --> STATUS
- 0x01 --> LOGIN
- 0x02 --> CREATE
- 0x03 --> ADD
- 0x04 --> REMOVE
- 0x05 --> DELETE
- 0x06 --> SET
- 0x07 --> MODIFY
- 0x08 --> AUTHSET
- 0x09 --> GET
- 0x0a --> VALIDATE
- 0x10 --> CHILDREN
- 0x11 --> TYPEOF
- 0x12 --> DESC_TYPE
- 0x13 --> LOCK_WRITE
- 0x14 --> LOCK_READ
- 0x15 --> REGISTER_NOTIF

Data Transmission format

String: A sequential collection of Unicode characters that is used to represent text. The zero value will be placed at the end of a String to indicate the end of it (Null-terminated String).

Binary Object: A Data buffer, which format will be {<buffer_length>,<buffer_data>}. The <buffer_length> encoded as a 4 bytes unsigned Integer32.

Integer8: an unsigned 8 bit value

Integer16: an unsigned 16 bit value

Integer32: an unsigned 32 bit value

IP Address: Value of 4 bytes. In Network Byte Order.

MAC Address: Value of 6 bytes. In Network Byte Order.

Path: an absolute path that represents the node location at the Configuration Tree. It is constituted from node names (in hierarchical structure) separated by "." character.

List of Commands

The Commands that can be sent by the clients to Configuration Agent are showed below. Each of the following commands will be encapsulated in the <BODY> of the <COMMAND_PCK> packet.

- C_LOGIN
- C_CREATE
- C_ADD
- C_REMOVE
- C_DELETE
- C_SET
- C_MODIFY
- AUTHSET
- C_GET
- C_VALIDATE
- C_CHILDREN
- C_TYPEOF
- C_DESCTYPE
- C_LOCK
- REGISTER_NOTIF

Detailed Description of Commands Format

Detailed format of the commands sent by Clients to the Configuration Agent is described below.

Command: Login

Sent by an external client to start the authentication process.

Syntax:

```
<C_LOGIN>: { <LOGIN>, <USERNAME>, <PASSWORD> }
<LOGIN>: Command Operation Code 0x01. 1 byte.
<USERNAME>: Field Type: String
<PASSWORD>: Field Type: String
```

Replied with: <R_STATUS>

Command: Create

Sent by a client to attach a new child Node specified in the <NAME> field to the Node specified by a <NODE> field. The field <ARGS> specifies the arguments of the specific Node Type.

Syntax:

```
<C_CREATE> :{ <CREATE>, <NODE>, <NAME>, <ARGS> <READ/WRITE FLAGS>}
<CREATE>: Command Operation Code 0x02. 1 byte.
<NODE>: The absolute path leading to a Configuration tree node.
    Field Type: Path.
<NAME>: The name of the node that should be attached as a child to
    the node specified by <NODE> field. Field Type: String.
<ARGS>: { <NARGS>, <ARG>, <ARG> ... }
<NARGS>: The number of the arguments <ARG> that following.
    Field Type: Integer8
<ARG>: An Argument of the Node Type of the created node specified
    in <NAME> field. Field Type: String.
<READ/WRITE FLAGS>: Field Type: Integer8
```

Replied with: <R_STATUS>

Command: Add

Sent by a client to attach a new child node specified in the <NAME> field to the Node specified by a <NODE> field.

Syntax:

```
<C_ADD>: {<ADD>, <NODE>, <NAME>, <READ/WRITE FLAGS> }
<ADD>: Command Operation Code 0x03. 1 byte.
<NODE>: The absolute path leading to a Configuration tree node.
    Field Type: Path.
<NAME>: A Leaf Node Name. Field Type: String.
<READ/WRITE FLAGS>: Field Type: Integer8
```

Replied with: <R_STATUS>

Command: Remove

Sent by a client to remove the subtree rooted at specified <NODE> field from the Configuration Tree.

Syntax:

```
<C_REMOVE> :{ <REMOVE>, <NODE> }
<REMOVE>: Command Operation Code 0x04. 1 byte.
<NODE>: The absolute path leading to a Configuration tree node.
    Field Type: Path.
```

Replied with: <R_STATUS>

Command: Delete

Sent by a client to delete the subtree rooted at specified <NODE> field from the Configuration Tree.

Syntax:

```
<C_DELETE>: { <DELETE>, <NODE> }
<DELETE>: Command Operation Code 0x05. 1 byte
<NODE>: The absolute path leading to a Configuration tree node.
    Field Type: Path.
```

Replied with: <R_STATUS>

Command: Set

Sent by a client to set values to the leaf nodes of the Edge Node specified at the <NODE> field.

Syntax:

```
<C_SET>: { <SET>, <NODE>, <ARGS> }
<SET>: Command Operation Code 0x06. 1 byte
<NODE>: The absolute path leading to a Configuration tree node.
    Field Type: Path.
<ARGS>: { <NARGS>, <ARG>, <ARG> ... }
<NARGS>: The number of the arguments <ARG> that following.
    Field Type: Integer8
<ARG>: { <LEAF>, <VALUE> }
<LEAF>: A Leaf Node Name. Field Type: String.
<VALUE>: This field contains the value of the <LEAF>. The Type of
    this value depend on the Leaf Nodes' Type, it's packed in a
    transmission format.
```

Replied with: <R_STATUS>

Command: Modify

Sent by a client to set values to the leaf nodes of the Edge Node specified at the <NODE> field.

Syntax:

```
<C_MODIFY>:( <MODIFY>, <NODE>, <ARGS> )
<MODIFY>: Command Operation Code 0x07. 1 byte
<NODE>: The absolute path leading to a Configuration tree node.
    Field Type: Path.
<ARGS>: ( <NARGS>, <ARG>, <ARG> ... )
<NARGS>: The number of the arguments <ARG> that following.
    Field Type: Integer8
<ARG>: ( <LEAF>, <VALUE> )
<LEAF>: A Leaf Node Name. Field Type: String.
<VALUE>: This field contains the value of the <LEAF>. The Type of
    this value depend on the Leaf Nodes' Type, it's packed in a
    transmission format.
```

Replied with: <R_STATUS>

Command: AuthSet

Sent by a Client to set the authorization Read/Write Flags of the specified Node at <NODE> field.

Syntax:

```
<C_AUTHSET> :( <AUTHSET>, <NODE>,<READ/WRITE FLAGS> )
<AUTHSET>: Command Operation Code 0x08. 1 byte
<NODE>: The absolute path leading to a Configuration tree node.
    Field Type: Path.
<READ/WRITE FLAGS>: Field Type: Integer8
```

Replied with: <R_STATUS>

Command: Get

Sent by a Client requesting the values of the leaf Nodes <LIST_LEAF> attached to the Edge Node specified in <NODE> field. If the <LIST_LEAF> field is empty, the

Configuration Agent will return the values of all leaf Nodes attached to the Edge Node.

Syntax:

```
<C_GET>: ( <GET>, <NODE>, <LIST_LEAF> )
<GET>: Command Operation Code 0x09. 1 byte
<NODE>: The absolute path leading to a Configuration tree node.
    Field Type: Path.
<LIST_LEAF>: One or More Leaf Node Names.
    Field Type: String. Optional Field
```

Replied with: <R_GET> or <R_STATUS> In case any error occurred, The Configuration Agent will reply with <R_GET> to the client. Otherwise, the client will receive the <R_STATUS>.

Command: Validate

This "command" verifies if the changes applied to the node are in accordance with its Node-Type.

Syntax:

```
<C_VALIDATE>:( <VALIDATE>, <NODE> )
<VALIDATE>: Command Operation Code 0x0a. 1 byte
<NODE>: The absolute path leading to a Configuration tree node.
    Field Type: Path.
```

Replied with: <R_STATUS>

Command: Children

When received by a Configuration agent, it sends to a client a list of interior node names that have the specified node as a parent Node.

Syntax:

```
<C_CHILDREN>:( <CHILDREN>, <NODE>)
<CHILDREN>: Command Operation Code 0x10. 1 byte
<NODE>: The absolute path leading to a Configuration tree node.
    Field Type: Path.
```

Replied with: <R_CHILDREN> or <R_STATUS> In case any error occurred, The Configuration Agent will reply with <R_CHILDREN> to the client. Otherwise, the client will receive the <R_STATUS>.

Command: TypeOf

When received by a Configuration agent, it sends to a client the Node-Type of the specified Node.

Syntax:

```
<C_TYPEOF>:{<TYPEOF>, <NODE> }
<TYPEOF>: Command Operation Code 0x11. 1 byte
<NODE>: The absolute path leading to a Configuration tree node.
    Field Type: Path.
```

Replied with: <R_TYPEOF> or <R_STATUS> In case any error occurred, The Configuration Agent will reply with <R_TYPEOF> to the client. Otherwise, the client will receive the <R_STATUS>.

Command: DescribeType

When received by a Configuration Agent, it sends to a client the meta-information regurding a Node Type.

Syntax:

```
<C_DESCTYPE>:{ <DESCTYPE>, <TYPE> }
<DESCTYPE>: Command Operation Code 0x12. 1 byte
```

Replied with: <R_DESCTYPE> or <R_STATUS> In case any error occurred, The Configuration Agent will reply with <R_DESCTYPE> to the client. Otherwise, the client will receive the <R_STATUS>.

Command: Lock

Syntax:

```
<C_LOCK>: {<LOCK>, <TYPE>, <NODE> }
<LOCK>: Command Operation Code 0x13. 1 byte
<TYPE>: Two Integer8 values. One for the Read Lock, and the other for
    the Write lock.
<NODE>: The absolute path leading to a Configuration tree node.
    Field Type: Path.
```

Replied with: <R_STATUS>

Command: RegisterNotification

Sent by a client in order to inform the configuration agent that it wishes to be notified about the changes performed to the subtree rooted to the node.

Syntax:

```
<REGISTER_NOTIF>: { <REGISTER>, <NODE>, <CHANNEL_ID> }
<REGISTER>: Command Operation Code 0x15. 1 byte
<NODE>: The absolute path leading to a Configuration tree node.
    Field Type: Path.
<CHANNEL_ID>: Field Type: Integer32
```

Replied with : <R_STATUS>

Detailed Description of Response Format

Response: Status

For all commands that expect no data in return, there is one response packet, a <R_STATUS> response. <R_STATUS> packet is sent by the Configuration Agent as a generic response for commands that expect no data, and as an error response to any requested command. The arguments of the response are a single-byte error code (zero is success, non-zero is error) followed by a short message describing the reason of the failure.

Syntax:

```
<R_STATUS>: {<STATUS>, <ERRCODE>, <MESSAGE> }
<STATUS>:  Response Operation Code 0x00. 1 byte
<ERRCODE>: Completion code. 1 byte. 0 for success, any other value
    for error
<MESSAGE>: Error message. Field Type: String
```

The following error codes are currently defined. **<ERRCODE> :**

```
0x00 --> "No Error"
0x01 --> "Some error"
0x02 --> "Out of memory"
```

Response: Get

Syntax:

```
<R_GET>: { <GET>, <ARGS> }
<GET>: Response Operation Code 0x09. 1 byte
<ARGS>: { <NARGS>, <ARG> , <ARG> . . . }
<NARGS>: The number of the arguments <ARG> that following.
    Field Type: Integer8
<ARG>: { <LEAF>, <VALUE> }
<LEAF>: A name of the leaf node. Field Type: String
<VALUE>: A value of the leaf node specified in <LEAF> field. The Type
    of this value depend on the Leaf Nodes' Type, it's packed in a
    transmission format.
```

Response: Children

Syntax:

```
<R_CHILDREN>: ( <CHILDREN>, <NODES> )
<CHILDREN>: Response Operation Code 0x10. 1 byte
<NODES>: ( <NNAMES>, <NAME> , <NAME> . . . )
<NNAMES>: The number of Node names <NAME> that following.
     Field Type: Integer8
<NAME>: A Node Name. Field Type: String
```

Response: TypeOf

Syntax:

```
<R_TYPEOF>: ( <TYPEOF>, <CLASS> )
<TYPEOF>: Response Operation Code 0x11. 1 byte
<CLASS>: The Node Type (Class) of the Node. Field Type: String
```

Response: DescribeType

Syntax:

```
<R_DESCTYPE>: ( <DESCTYPE>, <TYPE_NAME>, <NODES>, <LEAFS> )
<DESCTYPE>: Response Operation Code 0x12. 1 byte
<TYPE_NAME>: A name of a Node Type. Field Type: String
<NODES>: ( <NNODES>, <NODE>, <NODE> . . .)
<NNODES>: The number of interior Nodes <NODE> that following.
    Field Type: Integer8
<NODE>: ( <NODE_NAME>, <NODE_TYPE>, <MANDATORY> )
<NODE_NAME>: The allowed name of a child Node. Field Type: String
<NODE_TYPE: The allowed Type of a child node specified in
    <NODE_NAME>. Field Type: String
<MANDATORY>: Field type: An Integer8.
<LEAFS>: ( <NLEAFS>, <LEAF>, <leaf> . . .)
<NLEAFS>: The number of Leaf Nodes <LEAF> that following.
    Field Type: Integer8
<LEAF>: ( <LEAF_NAME>, <LEAF_TYPE>, <MANDATORY> )
<LEAF_NAME>: The allowed name of a leaf Node. Field Type: String
<LEAF_TYPE>: The allowed Type of a leaf node specified in
    <LEAF_NAME>. Field Type: String
<MANDATORY>: Field type: An Integer8
```

5. References

1. WiMAX Forum. WiMAX end-to-end network systems architecture. Stage 2: Architecture tenets, reference model and reference points.

2. WiMAX Forum. WiMAX end-to-end network systems architecture. Stage 3: Detailed protocols and procedures.

3. Fundamentals of WiMAX, Understanding Broadband Wireless Networking. Jeffrey G.Andrews, Arunabha Chosh, Rias Muhamed

4. WiMAX: Technology for Broadband Wireless Access, Loutfi Nuaymi

5. WiMAX/MobileFi Advanced Research and Technology, Edited by Yang Xiao

6.WiMAX - A Wireless Technology Revolution. G. S. V. Radha, Krishna Rao, G. Radhamani

7. WiMax Overview. Sanida Omerovic

8. RFC 2865 Remote Authentication Dial In User Service (http://www.faqs.org/rfcs/rfc2865.html)

9. ftp://ftpeng.cisco.com/pub/tacacs/tac-rfc.1.78.txt

10. RFC 2138 - Remote Authentication Dial In User Service (RADIUS).

11. RFC 2869, RADIUS Extensions.

12. RFC 3579, RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP).

13. RFC 2904: "AAA Authorization Framework", IETF; August 2000.