KONGSBERG

CoNSIS

Implementation of the NetAH (Military Authentication Header/AH*/QAH) Technical Report

Document ID: 2/1559/2-FCPR10127 Rev A

Summary

A primary goal of the Network Authentication Header (NetAH) is to protect the QoS signaling in the outer IP header. NetAH can also be used to distinguish authorized from un-authorized nodes when building a protected network. The concept was described in an article in the IEEE Communication Magazine in 2008. This report describes the implementation.

The NetAH is a modified version of the standard IPsec Authentication Header (AH). In contrast to standard IPsec, it also protects the QoS signaling in the outer IP header. It is implemented as a hop-by-hop extension header. It is compatible with standard IPsec in the sense that it can be used in addition to this.

Tactical communication nodes from Kongsberg were used as experimental platforms. The NetAH can be ported to other experimental platforms. A NetAH patch is available for Linux 2.6.39.4. It comes with a patch for IPsec-Tools version 0.7.3.

Table of Contents

1. Introduction ... 6
1.1 Scope ... 6
1.2 Identification ... 6
1.3 Related work ... 6
1.4 Abbreviations and definitions ... 7
1.5 Document overview ... 8
1.6 References and related documents ... 8
2. Background information ... 9
2.1 Military Authentication Header, AH* - NetAH ... 9
2.1.1 Motivation ... 9
2.1.2 Standard IPsec ... 9
2.1.3 NetAH ... 10
2.1.4 Compatibility requirements between AH and NetAH ... 11
2.1.5 Combining ESP and NetAH ... 11
2.2 The experimental platforms: the Kongsberg TacLAN communication nodes ... 13
2.2.1 System overview ... 13
2.2.2 SR600 and WM600 ... 14
2.2.3 TR600 ... 14
3. The NetAH implementation ... 15
3.1 Architecture ... 15
3.2 NetAH header formats ... 15
3.3 Outbound processing of the NetAH ... 18
3.3.1 Specification ... 18
3.3.2 Implementation of the outbound processing of NetAH in the Linux IP stack ... 19
3.4 Hop-by-hop processing of NetAH ... 20
3.4.1 Specification of hop-by-hop processing of NetAH ... 20
3.4.2 Implementation of the hop-by-hop processing of NetAH ... 20
3.5 Inbound processing of NetAH ... 21
3.5.1 Specification ... 21
3.5.2 Implementation of the in-bound processing of NetAH ... 21
3.6 Possible future improvements ... 21
4. Testnetwork configuration of IPsec, QoS and routing ... 23
4.1 IPsec ... 23
4.1.1 IPsec configuration file for peer hosts ... 23
4.1.2 IPsec configuration for hosts along the route ... 25

4.1.3 Implementation of NetAH in setkey 27
4.2 Quality of Service 28
4.2.1 Classification of IP traffic 28
4.2.2 Shaping of IP traffic 28
4.2.3 Mapping between DSCP values and radio priority queues 28
4.3 Routing 30
5. Test network topology 31
6. Test scenarios 35
6.1 PC and SR/WM setup 38
6.1.1 Manipulation of DSCP field 38
6.1.2 IPERF 38
6.2 Test procedures and results 39
7. Test network – code updates 41
7.1 Updated Linux Kernel code 41
7.2 Function call chains 42

Figures

Figure 1 IPsec AH transport and tunnel mode 9
Figure 2 Authentication Header (AH) format 9
Figure 3 IPv6 header format – fields not protected by AH in grey 10
Figure 4 IPv6 header format – fields not protected by NetAH in grey 11
Figure 5 Military AH combined with ESP 11
Figure 6 End to end NetAH transport link 12
Figure 7 NetAH transport link without priority update along the route 12
Figure 8 Overview of TacLAN environment and Kongsberg products 13
Figure 9 System overview: SR600 for dismounted soldiers and WM600 for vehicles 14
Figure 10 WM600 and SR600 14
Figure 11 Tactical router TR600 14
Figure 12 IPv6 and Military Authentication Header (NetAH) format 16
Figure 13 Implementation of NetAH handling in Linux IP stack 19
Figure 14 Example of configuration file for setkey at NetAH/ESP peer hosts. 24
Figure 15 Configuration file for setkey at NetAH/ESP IP hosts along the route. 26
Figure 16 tc configuration script used for air interface (uhf0) (SPEED = 400 Kbit, ie. optimized for 920 Kbit/sec) 29
Figure 17 Firewall script used by iptables in radios 30
Figure 18 Test network (IPsec view) 32
Figure 19 Test network (IP address view) 33

Figure 20 Test network (Routing domain view) 34
Figure 21 Test scenario 1: Reference setup 36
Figure 22 Test scenario 2: Verify use of NetAH via COTS router, no priority differences 36
Figure 23 Test scenario 3: Verify prioritizing of high priority packets when congestion occurs and using NetAH 36
Figure 24 Test scenario 4: Verify use of NetAH via TR600 tactical router 37
Figure 25 Test scenario 5: Demonstrate consequences of manipulated AH 37
Figure 26 Test scenario 6: Verify handling of packets with manipulated NetAH 37

Tables

Table 1 NetAH fields 16
Table 2 Overview of modified files 27
Table 3 Test scenarios 35
Table 4 Test procedures 39
Table 5 Overview of updated kernel code 41

1. Introduction

Standard IPsec leaves the mutable fields of the outer IP header unprotected. This makes the QoS signaling in the IP header prone to attacks. Reference [7] proposes a modified version of the IPsec authentication header that also protects the QoS signaling in the outer header. This report describes the implementation of the concept.

The modified AH is referred to as the military authentication header, AH*, QAH or Network Authentication Header (NetAH). This report adopts the term NetAH. The code listings also uses qah.

CoNSIS focuses on tactical networks, and the NetAH was implemented on Kongsberg UHF broadband tactical communication nodes that were to be used as experimental platforms in the wireless ad hoc test network in addition/extension to the fixed CoNSIS network. The Kongsberg UHF broadband communication nodes include vehicle mounted WM600s and SR600 soldier radios. The Kongsberg broadband radios include the basic functionality of IPv6, IPsec, DiffServ, OLSR or Wireless OSPF and OSPF required in the CoNSIS experimentation. The Kongsberg tactical router, TR600, was also used in the tests.

The NetAH functionality can be ported to other experimental platforms. A NetAH patch is available for Linux 2.6.39.4. It comes with a patch for IPsec-Tools version 0.7.3.

1.1 Scope

The scope of the document is to describe the work and results from the implementation and demonstration of the NetAH on Kongsberg UHF broadband tactical communication platforms. An introductory overview of the NetAH concept and the experimental platforms is also included.

The document is intended both for readers that are unfamiliar with IPsec and the NetAH concept as well as the more experienced software developers. The experienced audience can skip the introductory parts.

1.2 Identification

This document is the delivery from the “Military Authentication Header, AH*” activity provided by Kongsberg Defence & Aerospace (KDA) under Norwegian national funding within the CoNSIS project and under contract between Forsvarets Forskningsinstitutt (FFI) and KDA [9]. FFI coordinates the Norwegian participation in the CoNSIS project. A number of private subcontractors have been engaged to provide parts of the work. KDA is one of the subcontractors. This report one out of multiple deliveries in the Coalition Network for Secure Information Sharing (CoNSIS) project.

1.3 Related work

The concept of NetAH was introduced in the IEEE Communications Magazine article ”QoS signaling in IP-based Military Ad HoC Networks”[7]. The article was a result of a joint work of KDA and FFI during the GOSIKT project. The article describes the concept. This report describes the implementation of the concept.

1.4 Abbreviations and definitions

AH	Authentication Header
AH*	Military authentication header –other term for NetAH
BGP	Border Gateway Protocol
BITS	Bump in the stack
BITW	Bump in the wire
COTS	Components Off The Shelf
DS	Differentiated Services
DSCP	Differentiated Services Code Point
ECN	Explicit Congestion Notification
ESN	Extended Sequence Number
ESP	Encapsulating Security Payload
FFI	Forsvarets Forskningsinstitutt
GOSIKT	Grunnleggende teknologier Og trender innen SIKkerheT (Norwegian)
ICV	Integrity Check Value
IPComp	IP Payload Compression Protocol (IPComp)
IPsec	IP security protocol
KDA	Kongsberg Defence & Aerospace
KAME	Project name. KAME was a joint effort of six organizations in Japan which aimed to provide a free IPv6 and IPsec
MOU	Memorandum Of Understanding
NetAH	Network Authentication Header
OLSR	Optimized Link State Routing protocol
OSPF	Open Shortest Path First
QAH	QoS Authentication Header – other term for NetAH
QoS	Quality of Service
RIP	Routing Information Protocol
SA	Security Association
SAD	Security Association Database
SPD	Security Policy Database
SPI	Security Parameter Index
UHF	Ultra High Frequency

1.5 Document overview

The report starts with an introduction to the NetAH concept and the experimental platforms used. Then the requirements and details on the implementation are described. The report also specifies the test network and results from the verification tests.

1.6 References and related documents

[1] S. Bradner, and V. Paxson, "IANA Allocation Guidelines For Values In the Internet Protocol and Related Headers," RFC2780, 2005

[2] T. Clausen (Ed.), and P. Jacquet (Ed.), "Optimized Link State Routing Protocol (OLSR)," RFC 3626, 2003.

[3] R. Coltun, D. Ferguson, J. Moy and A. Lindem (Ed.), "OSPF for IPv6," RFC5340,2008

[4] CoNSIS MOU, 03-06-09

[5] S. Deering, and R. Hinden. 1998. "RFC 2460. Internet Protocol, Version 6 (IPv6) Specification."

[6] F. Fenner, "Experimental Values in IPv4, IPv6, ICMPv4, ICMPv6, UDP and TCP Headers," RFC 4727, 2006

[7] A. M. Hegland, and E. Winjum,"QoS signaling in IP-based Military Ad HoC Networks," IEEE Communications Magazine, November 2008.

[8] S. Kent. 2005. "RFC 4302. IP Authentication Header."

[9] Technical Arrangement - TA Number 2009.01 between FFI and KDA Concerning KDA deliverables to the CoNSIS project, Annex B to Collaboration Agreement

[10] T. Narten, "Assigning Experimental and Testing Numbers Considered Useful," RFC 3692, 2004

[11] T. Narten, "Guidelines for Writing and IANA Considerations Section in RFCs,"RFC5226, 2008.

[12] R. Ogier, and P. Spagnolo, "Mobile Ad Hoc Network (MANET) Extension of OSPF Using Connected Dominating Set (CDS) Flooding," RFC 5614, 2009.

[13] A. Shacham, B. Monsour, R. Pereira, and M. Thomas, " IP Payload Compression Protocol (IPComp)", RFC 3173, 2001

2. Background information

This section outlines the NetAH concept and provides an overview of the experimental platforms.

2.1 Military Authentication Header, AH* - NetAH

2.1.1 Motivation

Proper QoS provisioning is important in military networks – not least tactical ad hoc networks. The QoS qualifiers are placed in the IP header to allow the routers to act on them. Without proper protection, this also paves the way for malicious nodes to change the traffic flow in the network. IPsec does not protect these fields. NetAH was therefore proposed to protect QoS signalling in the IP header.

2.1.2 Standard IPsec

IPsec relies on the *Authentication Header* (AH) and the *Encapsulating Security Payload* (ESP) protocols for protection of the IP payload. AH assures integrity and data origin authentication. ESP provides confidentiality in addition. IPsec *transport mode* encapsulates the payload, and uses the original IP header for routing decisions. *Tunnel mode* adds a new IP header as illustrated in Figure 1. In tunnel mode, the DSCP value and ECN bits are copied from the inner to the outer header. The IPv6 Flow Label can be copied or constructed depending on whether IPsec is implemented in an end-host or in a separate IPsec gateway, respectively. (Constructed Flow Labels is required in the latter case in order to ensure unique labels.) IP header *options* are never copied from the inner to the outer header. The authentication header format is shown in Figure 2.

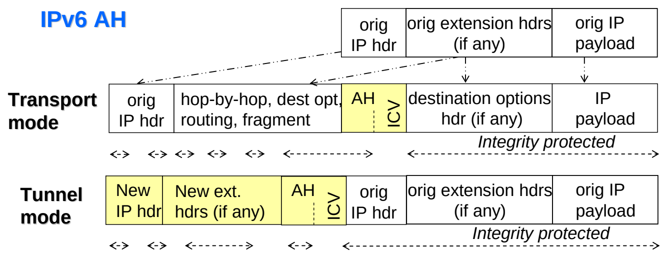

Figure 1 IPsec AH transport and tunnel mode

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Next Header   |  Payload Len  |          RESERVED             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Security Parameters Index (SPI)               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Sequence Number Field                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                Integrity Check Value-ICV (variable)           |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2 Authentication Header (AH) format

ESP does not cover the outer IP header. AH, on the other hand, includes the immutable and predictable fields of the outer IP header. But the IPv6 Traffic Class (DS and ECN), Flow Label and HopLimit are mutable, and are therefore not protected. Fields not protected by the AH are highlighted in grey color in Figure 3.

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version| Traffic Class |           Flow Label                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Payload Length        |  Next Header  |   Hop Limit   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                         Source Address                        +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                      Destination Address                      +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3 IPv6 header format – fields not protected by AH in grey

- The IPv6 header fields are classified as follows:
    - Immutable:
      Version, Payload Length, Next Header, Source Address, Destination Address (without Routing Extension Header)
    - Mutable but predictable:
      Destination Address (with Routing Extension Header)
    - Mutable:
      DSCP (6 bits), ECN (2 bits), Flow Label, Hop Limit.

2.1.3 NetAH

NetAH is parallel to the standard IPsec AH, but includes the Traffic Class (DS and ECN fields), IPv6 flow labels as illustrated in Figure 4 plus relevant options or extension headers in the integrity protection.

The NetAH prevents undetectable modifications of the QoS qualifiers in the IP header as the datagram traverses the network. Assuming a symmetric group key; all group members can verify the message authenticity. The authorized nodes (group members) can modify the QoS qualifiers, and update the ICV of the AH* accordingly. But modifications made by externals will be detected.

On reception of a datagram, the NetAH enabled nodes check the ICV before forwarding decisions are made.

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version| Traffic Class |           Flow Label                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Payload Length        |  Next Header  |   Hop Limit   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                         Source Address                        +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                      Destination Address                      +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4 IPv6 header format – fields not protected by NetAH in grey

2.1.4 Compatibility requirements between AH and NetAH

Processing of AH and NetAH shall not affect each other in a network where both AH and NetAH traffic is present. This means that:

1. Standard IPsec shall not handle NetAH packets.
2. Extended IPsec with NetAH shall differentiate between standard AH packets and NetAH packets.

Both requirements are achieved by the use of *Next Header* field. Standard AH is indicated in the IPv6 header by setting the *Next Header* field to 0x33 (AH). NetAH is indicated by using a Hop-by-hop options IPv6 extension header as described in section 2.2.

2.1.5 Combining ESP and NetAH

The NetAH can be combined with ESP, as illustrated in Figure 5. ESP assures end-to-end confidentiality and integrity of user data through a symmetric pair-wise key that is different from the NetAH group key. The NetAH ensures IP header authenticity in transit. The NetAH is calculated over the entire datagram, including ESP and the IP header, excluding the HopLimit field. The NetAH can co-exist with standard IPsec implementations, and can be tunnelled through ordinary IPsec enabled networks. Nodes using ordinary IPsec implementations will forward the datagram towards its destination on the basis of the IP header destination address without examining the NetAH.

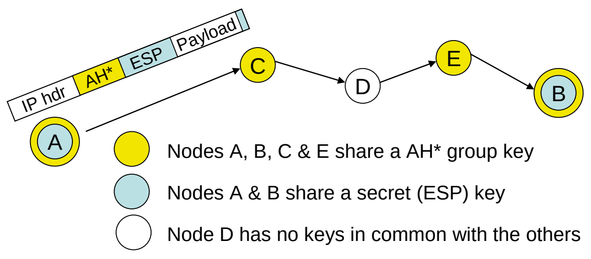
IP hdr
AH*
ESP
Payload
A
C
D
E
B
Nodes A, B, C & E share a AH* group key
Nodes A & B share a secret (ESP) key
Node D has no keys in common with the others

Figure 5 Military AH combined with ESP

A test setup is illustrated in Figure 6. An ESP tunnel together with a NetAH is established between the two peripherals at each end of the link. All hosts in grey colour share the same NetAH key, while only the source and destination hosts know the ESP key. The QoS priority held by the DS field could be set and/or updated by the source or by a radio or a router holding the common NetAH key along the route, depending of the QoS policy of the network.

In the example shown in Figure 6, a default priority is set in the DS field and the NetAH created by the source peripheral and verified by all of the hosts holding the common NetAH key. The grey coloured router may manage and control the priority by updating the DS field according to the QoS policy of the network. When updating the DS field, the NetAH is updated using the NetAH key. The white coloured router is used as a transport node, but not fully trusted regarding quality of service manipulation. Thus, the common NetAH key is not assigned to the white router. In case the DS field is manipulated, it will be detected by the next host with a valid NetAH key.

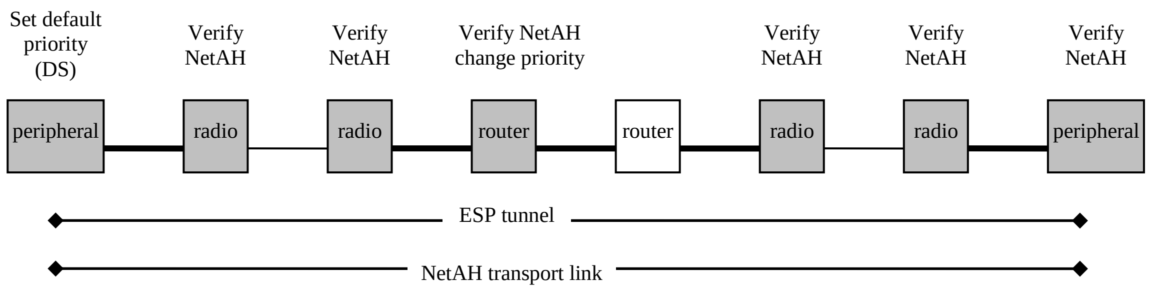

Figure 6 End to end NetAH transport link

In our system, NetAH is implemented in the radios but not in the external peripherals (hosts) as shown in Figure 7. Here the peer hosts sets the priority in the DS field when creating an IP packet. The connected radio creates the NetAH and the intermediate nodes along the route are able to verify the NetAH. The capability of updating the DS field and NetAH using the NetAH key is not included in the current implementation.

In case the DS field is manipulated by the white router, it will be detected by the next host with a valid NetAH key. Depending on policy, the datagram is discarded or forwarded best effort.

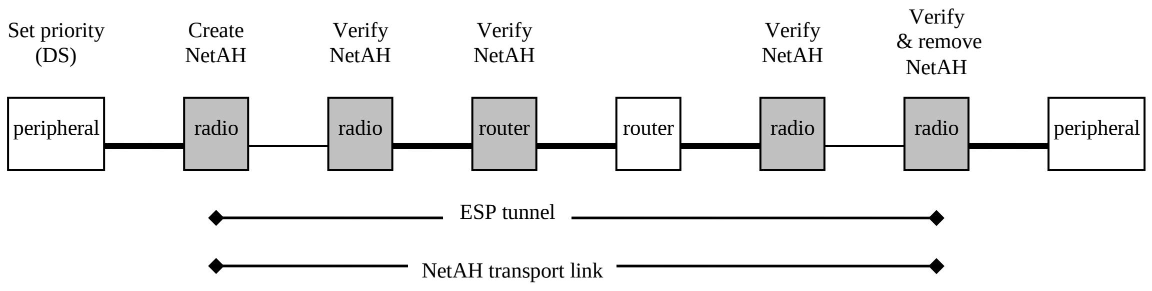

Figure 7 NetAH transport link without priority update along the route

2.2 The experimental platforms: the Kongsberg TacLAN communication nodes

2.2.1 System overview

The UHF soldier IP radio SR600, the WM600 vehicle IP radio, and the tactical router TR600 are all members of the Kongsberg TacLAN product range. TacLAN is a broadband communication infrastructure to support systems in vehicles, in Command Posts and on individuals in the lower echelon as outlined in Figure 8. The TacLAN system provides new system elements that support a wide variety of services – both wired and wireless.

To enhance interoperability in international military operations the TacLAN communication infrastructure is based on IP communication protocols. TacLAN provides both wired and wireless interfaces.

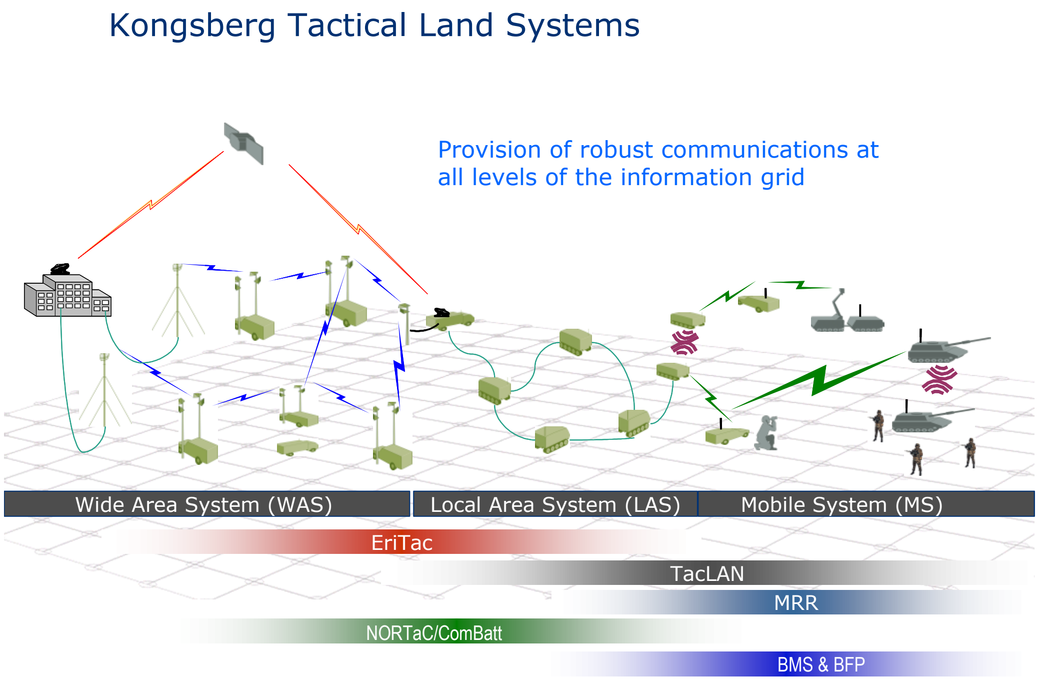
Kongsberg Tactical Land Systems
Provision of robust communications at all levels of the information grid
Wide Area System (WAS)
Local Area System (LAS)
Mobile System (MS)
EriTac
TacLAN
MRR
NORTaC/ComBatt
BMS & BFP

Figure 8 Overview of TacLAN environment and Kongsberg products

Information Exchange Requirements increases as the call for real- or near-real time information from the lower and highest tactical levels increases. Mobility is a challenge. The TacLAN system supports unit mobility, platform mobility, terminal mobility and individual mobility at a dynamic battlefield. A number of products support the TacLAN communication infrastructure in Figure 8.

The SR600, WM600 and TR600 solve a range of scenarios. Figure 9 shows one.

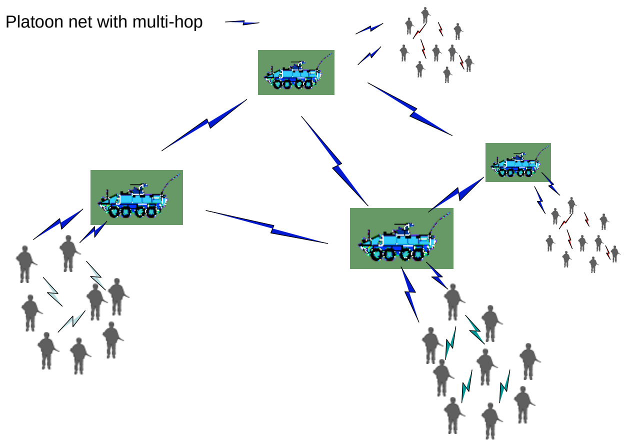

Figure 9 System overview: SR600 for dismounted soldiers and WM600 for vehicles.

2.2.2 SR600 and WM600

Figure 10 shows the WM600 and SR600. The SR600 is designed for dismounted soldier use. The WM600 is intended for vehicle mounting. The basic features of the two are the same. But the RF output power ranges from 10mW to 5W for the WM600 and 10mW to 1W for SR600. Both operate in the MIL UHF Band I: 225-400MHz. The data rates are from 100kbps to 2.5 Mbps. The radios include a Linux kernel and provide multi-hop ad hoc network routing. They are software defined, and are designed for the lower tactical echelons' operational requirements – including coverage.

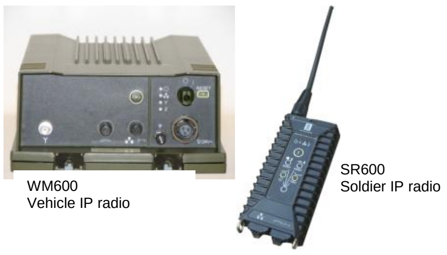

Figure 10 WM600 and SR600

2.2.3 TR600

The TR600 is illustrated in Figure 11. It makes the warfighter able to exploit the different communication links available in a vehicle or Command Post. The TR600 enable IP routing of data over a variety of links such as HF-radio, modems, IP aware as well as legacy Combat Net Radios, the WM600/SR600 broadband radios and a variety of SATCOM links – including BGAN and similar commercial systems. The TR600 is an integrated part of the TacLAN system. It supports OSPF, RIP and BGP.

Figure 11 Tactical router TR600

3. The NetAH implementation

3.1 Architecture

In Ipv6 the IPsec's protocols and capabilities are integrated with the IP layer. This allows all IPsec security modes and capabilities to be provided just as easily as regular IP. No extra hardware or architectural layers are needed.

The operating system of the radios is Linux. IPsec-Tools is a part of KAMEs IPsec utilities to the Linux 2.6 IPsec implementation. NetAH is implemented as modifications of the Linux kernel and IPsec-Tools with the following main changes:

1. Updated outbound processing. This is carried out at the start of a NetAH transport link.
2. Added hop-by-hop processing, including verification of the IP packet at a host along the NetAH transport link.

   For hop-by-hop processing the NetAH key together with the security associations (SA) have to be known to the hosts along the route. The distribution mechanism of SAs and keys is not covered by this report.
3. Updated inbound processing. This is carried out at the end of the NetAH transport link.
4. Added handling of IP packet with failing NetAH verification, either as a part of the Hop-by-hop processing or inbound processing.

3.2 NetAH header formats

Figure 12 shows the NetAH hop-by-hop options header in an IPv6 encapsulation. Table 1 details the various fields.

The NetAH header is identical to the AH header with the exception of the Payload Len (8 bits) is replaced with the HdrExtLen field (8bits), and the 8 bits Option type and 8 bits OpdDataLen take the place of the16 bits Reserved fields of the standard AH header.

The NextHeader field in the IPv6 header contains the value zero for Hop-by-hop options rather than the value used when traditional AH extension header follows.

The NetAH extension header must be processed by each intermediate node along the delivery path. This means that the NetAH extension header must be implemented as a hop-by-hop options header as these are the only ones that are processed by intermediate nodes. Extension headers such as AH and ESP are only processed by the end hosts.

The Hop-by-hop option is placed immediately after the standard IP header, before other extension headers. The presence of the Hop-by-hop option is indicated by the value zero in the *Next Header* field of the IPv6 header.

The two most significant bits of the option type field were set to ‘00’. By default, packets with unrecognized hop-by-hop extensions would otherwise be discarded. When set to ‘00’ the intermediate node will skip over this option and continue processing the header even though it does not recognize the NetAH hop-by-hop options header.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version| Traffic Class |           Flow Label                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Payload Length        | Next Header=00|   Hop Limit   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                         Source Address                        +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                      Destination Address                      +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Next Header   |  Hdr Ext Len  | Option type=1E| Opt Data Len  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Security Parameters Index (SPI)               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Sequence Number Field                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                Integrity Check Value-ICV (variable)           |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 12 IPv6 and Military Authentication Header (NetAH) format

Table 1 NetAH fields

NetAH hop-by-hop fields	
Next Header:	8-bit selector. Identifies the type of header immediately following the Hop-by-Hop Options header
Hdr Ext Len	8-bit unsigned integer. Length of Hop-by-hop Options header in 8-octet units, not including the first 8 octets.
Option type:	8-bit identifier of the option type**. For NetAH set to: 0x1E (00011110)**
Opt Data Len:	8-bit unsigned integer. Length of the Option Data field of this option, in octets
SPI	32-bit security parameter index
Sequence Number field	32-bit sequence number
ICV	Variable integrity check value. The chosen Linux kernel uses **96 bits.**

The third most significant bit in the options type field was set to ‘0’ to indicate the option data do not change en route. That is, the three most significant bits of the NetAH options type field were set to ’000’.

Regarding the 5 least significant bits: An option type for experimentation that does not collide with others is needed. The Hop-by-hop options are defined by IANA. The number assignment is found at http://www.iana.org/assignments/ipv6-parameters. See also RFC2780 [1] and RFC3692 [10]. We chose option type set to 0x1E[1] for the experimentation with NetAH. RFC 3692 recommended a range of numbers specifically earmarked for testing and experimentation purposes - similar to those called "Private Use" in RFC 5226[11]. Furthermore, RFC 4727[6] reserves some ranges of numbers for experimentation purposes and describes the numbers that have already been reserved by other documents.

[1] It is only appropriate to use these values in explicitly-configured experiments; they MUST NOT be shipped as defaults in implementations. See RFC3692 for details.

3.3 Outbound processing of the NetAH

3.3.1 Specification

An outline of the AH processing for outbound packets specified by RFC 4302 is described below:

1. Security Association Lookup
2. Sequence Number Generation
3. Integrity Check Value Calculation
4. Fragmentation

The NetAH processing is similar to the AH processing, except the NetAH format and the ICV calculation are different.

The NetAH format is described in section 3.2.

For outbound NetAH processing the Integrity Check Value Calculation is modified. The NetAH ICV is computed over:

- IP or extension header fields before the NetAH header that are either immutable in transit or that are predictable in value upon arrival at the endpoint for the NetAH SA.
- The following mutable fields in addition to the fields covered for AH: DSCP, ECN and Flow Label.
- the NetAH header (Next Header, Payload Len, Reserved, SPI, Sequence Number (low-order 32 bits), and the ICV (which is set to zero for this computation), and explicit padding bytes (if any))
- everything after NetAH is assumed to be immutable in transit
- (the high-order bits of the ESN (if employed), and any implicit padding required by the integrity algorithm)

Fields which may be modified during transit, i.e. not covered by the ICV, and the ICV field itself, are set to zero and included in the ICV computation. When modifying AH to NetAH, the fields to be included in the ICV are not zeroed prior to the calculation.

3.3.2 Implementation of the outbound processing of NetAH in the Linux IP stack

The NetAH implementation is based on the AH implementation. In brief the following modifications have been carried out to implement NetAH:

- Addition of mutable fields for calculation of ICV, i.e. DSCP, ECN and Flow Label.
- Creation of header according to NetAH format instead of AH format.
- Addition of an internal protocol type for NetAH for differentiation from the other IP protocols in the Linux kernel. This is an addition to the existing IPsec protocols, i.e. ESP (IP protocol number 50), AH (51) and IPComp (108). The selected IP protocol number is 254 that is reserved for experimentation and testing. For the other IPsec protocols, the internal protocol number is the same as the header number. In our case the Hop-by-hop header is used and this has the protocol number 0. The number 0 is reseved for other uses in the Linux kernel. Thus the new kernel internal NetAH protocol number has to differ from the header number used in the IP-packet.

Figure 13 illustrates where the NetAH modifications have been implemented in the Linux IP stack. Outbound IP traffic follows the solid arrow on the right side. When a packet is received from the TCP/UDP layer, the IPsec policy is read from the Security Policy Database (SPD) based on the source and destination (1) and NetAH is created according to the SPD and the Security Association Database(SAD) (2). For Hop-by-hop processing, the matching SPI is found (3) and the NetAH ICV is verified (4) before it is forwarded to the correct network interface.

Inbound processing contains the same two steps (3) and (4) as Hop-by-hop processing and then the IPsec policy is read (5) and the NetAH is removed (6)

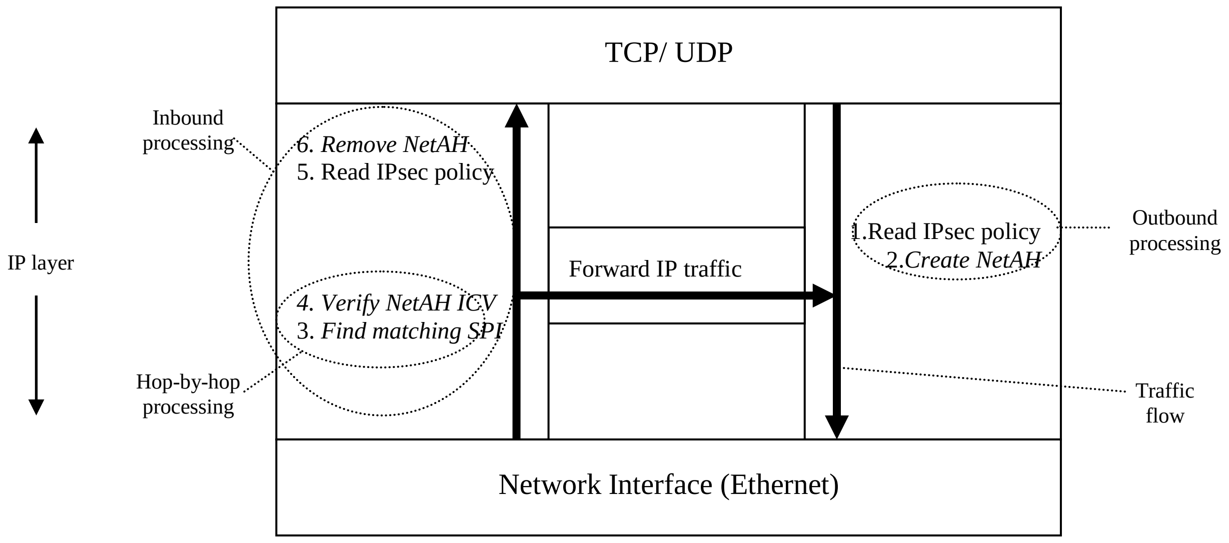

Figure 13 Implementation of NetAH handling in Linux IP stack

3.4 Hop-by-hop processing of NetAH

3.4.1 Specification of hop-by-hop processing of NetAH

The hop-by-hop processing of NetAH along the route shall be in accordance to the following steps:

1. Reassembly: The need for reassembly of fragmented IP packets along the route may be considered during future works.
2. Security Association Lookup

   For hop-by-hop processing the SPD and the SAD holding the NetAH key together with the security associations (SA) have to be known to the hosts along the route. Triggering of hop-by-hop processing of NetAH shall be based on:

   - Next header = 0x00 indicating hop-by-hop processing.
   - The *option type* field in the NetAH = 0x1E
   - Search for SPI as for standard AH.

     In case the SPI is not specified, the packet shall be handled in the same way as for failing ICV as described in step 4. In the current implementation the traffic class field is set to zero (lowest level) and the packet is forwarded.
3. Sequence number verification.
4. Integrity Check Value Verification.

   For NetAH the ICV calculation has to cover the mutable fields in the same way as for the ICV calculation carried out during the outbound packet processing. In case the ICV verification fails, the IP packet has to be handled according to a pre configured policy as described in section 3.6. In the current implementation the traffic class field is set to zero (lowest level).

3.4.2 Implementation of the hop-by-hop processing of NetAH

The NetAH implementation is based on the AH implementation. The following modifications have been carried out to implement NetAH:

- Hop-by-hop processing of NetAH is implemented as an extension of the hop-by-hop processing mechanism in the Linux kernel. The extension is based on some of the mechanisms used in inbound NetAH processing.
- When the hop-by-hop extension header is detected (Next Header = 0) the following is carried out:
  - Detection of The NetAH *option type* field = 0x1E
  - Detection of the SPI in the IP packet and search for a matching SPI in the SAD. In case the SPI is not found in the SAD, the traffic class field is set to zero (lowest level).
  - Verification of ICV. In case the verification fails the traffic class field is set to zero (lowest level)
  - Packet is forwarded for standard handling by the Linux kernel IP layer.

Reassembly of fragmented IP packets is not implemented. The need for reassembly of fragmented IP packets along the route may be considered for future updates.

Sequence number verification is not implemented. This may be implemented in a later release.

3.5 Inbound processing of NetAH

3.5.1 Specification

An outline of the AH processing for inbound packets specified by RFC 4302 [6] is described below:

1. Reassembly
2. Security Association Lookup.
3. Sequence Number Verification.
4. Integrity Check Value Verification

The NetAH processing shall be similar to the AH processing described above, except from the NetAH format and the ICV calculation.

The NetAH format shall be according to the description in section 3.2.

For inbound NetAH processing the Integrity Check Value Calculation shall be modified in the same way as for outbound processing described in section 3.3.1.

3.5.2 Implementation of the in-bound processing of NetAH

The NetAH implementation is based on the AH implementation and the NetAH hop-by-hop implementation. In brief, the following is implemented:

- The hop-by-hop header is handled in the same way as for hop-by-hop processing described in section 3.4 above. The packet is then routed through the IP layer to the NetAH inbound processing.
- The NetAH inbound processing is equal to the AH inbound processing except that the ICV calculation is skipped since this is already carried out by the hop-by-hop processing mechanism. Further the NetAH is removed from IP-packet. The remaining packet is processed according to any following header(s) using standard implementation and eventually sent to the addressed protocol above the IP-layer.

3.6 Possible future improvements

Later releases may be improved by the following additions and modifications:

- **Configurable handling of NetAH with failing ICV**

  Handling of IP packet with failing NetAH verification may either be a part of the Hop-by-hop processing or inbound processing.

  According to RFC 4302 [6], the receiver MUST discard a received IP datagram as invalid if the ICV test fails. However, when NetAH is used in combination with ESP it could be configurable how to handle failing ICV verification. A failing NetAH ICV may happen due to any intended or unintended change in the IP packet except from the *Hop Limit* field. Packets with manipulated content in the inner ESP packet will fail by the ESP ICV verification. Thus, if the packet fails the NetAH verification and passes the ESP verification, the content of interest that may have been changed is the NetAH fields *Traffic Class* (DSCP and ECP) and *Flow Label*.

Further handling of an IP packet with failing NetAH ICV and matching ESP ICV shall follow one of the alternatives according to a pre configured policy:

a) Discard the packet

b) Forward the packet with a pre configured priority, most likely low priority (e.g. *Best Effort*).

Configurable policy is currently not implemented. Packets with failing NetAH ICV is forwarded with the lowest priority as described in b).

- **Reassembly of IP packets for hosts along the route**

  The need for reassembly of fragmented IP packets along the route may be considered for future updates.

- **Sequence number verification**

  Sequence number verification may be implemented in a later release in order to prevent replay DOS attacks.

- **Skip NetAH ICV verification in the end host**

  In case the policy is to forward packets with failing NetAH ICV, it is not needed to calculate the ICV in the end host.

- **Skip NetAH ICV verification prior to high bandwidth hops**

  The objective of NetAH is to protect high priority traffic via narrow bandwidth (radio) links. Thus, the ICV verification may not be needed after a radio hop prior to a higher bandwidth link.

4. Testnetwork configuration of IPsec, QoS and routing

Preparation of the experimental platforms included configuration of IPv6, routing, IPsec and enabling of DiffServ. Amongst other the IPsec-tools was rewritten to support NetAH.

4.1 IPsec

IPsec is managed by using setkey, which is a part of the IPsec-tools application. setkey is invoked with a filename of a configuration file as input parameter. For standard IPsec the processing of AH and ESP is only active in the peer hosts of the AH and ESP links. For NetAH, IPsec NetAH also has to be active in the hosts along the route. An example of an IPsec configuration file for peer hosts is described in section 4.1.1, while IPsec configuration files for hosts along the route is described in section 4.1.2.

4.1.1 IPsec configuration file for peer hosts

The input file to setkey for NetAH and ESP is shown in Figure 14. Two NetAH/ESP transport links were defined:

- fc10:f510:3:21::2 to fc10:f510:3:20::2
- fc10:f510:3:22::2 to fc10:f510:3:20::2

The configuration file consists of two main sections: the definition of the security policy database (SPD) and the definition of the security association database (SAD).

The SPD entries are created by the spdadd command which holds the source and destination addresses or address ranges. The –P option defines the direction of the policy entry (inbound or outbound), the handling of the traffic (e.g. IPsec), the protocol (e.g. ESP) and mode (transport or tunnel). The example shows a bundle of both ESP and NetAH. IPsec has to be activated and configured in both ends of a tunnel or transport link.

The SAD entries are created by the add command which holds the source and destination addresses, the protocol (qah, ah, esp or ipcomp), the security parameter index (SPI), the algorithm and the key used by the algorithm. The SAD entries have to correspond to the policies defined by the SPD entries.

In the example the configuration supports both traffic originated in the host itself (e.g. a radio) and traffic originated in the subnet originated at the LAN (wired) side of the radio.

The configuration file may be applied both at the SR600(A) radio (address fc10:f510:3:21::2) and the SR600(B) radio (address fc10:f510:3:22::2) in the test setup.

The configuration file for the SR600(C) radio (address fc10:f510:3:20::2) is equal, except that the direction of the –P options in the SPD entries are swapped, i.e. 'in' is replaced with 'out' and vice versa.

The configuration file for NetAH is similar to a AH setup except that the AH setup uses the 'ah' token instead of 'qah'.

```
flush;
spdflush;

## Security Policy Database (SPD)

spdadd  fc10:f510:3:82::/64   fc10:f510:3:77::/64   any -P out ipsec esp/tunnel/fc10:f510:3:82::1-fc10:f510:3:77::1/require qah/transport//require;
spdadd  fc10:f510:3:77::/64   fc10:f510:3:82::/64   any -P in  ipsec esp/tunnel/fc10:f510:3:77::1-fc10:f510:3:82::1/require qah/transport//require;

spdadd  fc10:f510:3:82::/64   fc10:f510:3:7d::/64   any -P out ipsec esp/tunnel/fc10:f510:3:82::1-fc10:f510:3:7d::1/require qah/transport//require;
spdadd  fc10:f510:3:7d::/64   fc10:f510:3:82::/64   any -P in  ipsec esp/tunnel/fc10:f510:3:7d::1-fc10:f510:3:82::1/require qah/transport//require;

## Security Assosiation Database (SAD)
## sha1 key
#    src                 dst                 protocol spi      encr_algo  key 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
add  fc10:f510:3:82::1  fc10:f510:3:77::1      qah 0x300 -A hmac-sha1  0x3ffe05014819fff819fff3f3ffe05014819ffff;
add  fc10:f510:3:77::1  fc10:f510:3:82::1      qah 0x301 -A hmac-sha1  0x3ffe05014819fff819fff3f3ffe05014819ffff;

add  fc10:f510:3:82::1  fc10:f510:3:7d::1      qah 0x304 -A hmac-sha1  0x3ffe05014819fff819fff3f3ffe05014819ffff;
add  fc10:f510:3:7d::1  fc10:f510:3:82::1      qah 0x305 -A hmac-sha1  0x3ffe05014819fff819fff3f3ffe05014819ffff;

## des key
#    src                 dst                 protocol spi                encr_algo  key 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
add  fc10:f510:3:82::1  fc10:f510:3:77::1     esp 0x400 -m tunnel -E  des-cbc    0x3ffe05014819fff8;
add  fc10:f510:3:77::1  fc10:f510:3:82::1     esp 0x401 -m tunnel -E  des-cbc    0x3ffe05014819fff8;
add  fc10:f510:3:77::1  fc10:f510:3:7d::1     esp 0x402 -m tunnel -E  des-cbc    0x3ffe05014819fff8;
add  fc10:f510:3:7d::1  fc10:f510:3:77::1     esp 0x403 -m tunnel -E  des-cbc    0x3ffe05014819fff8;
add  fc10:f510:3:82::1  fc10:f510:3:7d::1     esp 0x404 -m tunnel -E  des-cbc    0x3ffe05014819fff8;
add  fc10:f510:3:7d::1  fc10:f510:3:82::1     esp 0x405 -m tunnel -E  des-cbc    0x3ffe05014819fff8;
```

Figure 14 Example of configuration file for setkey at NetAH/ESP peer hosts.

4.1.2 IPsec configuration for hosts along the route

Along the route, only the NetAH-key shall be available for trusted nodes. The ESP key shall only be available for the ESP tunnel peer hosts. An example of the setkey configuration file for hosts along the route is shown in Figure 15. As described in sections 3.4.2 and 3.6, the handling of traffic is currently only based on a matching SPI. Thus a SPD definition is not needed. For an extended implementation it could be useful to have the possibility to add policy statements in the SPD. These SPD statements could require traffic between given addresses (or all) in order for NetAH to keep its priority.

```
flush;
spdflush;


## Security Assosiation Database (SAD)
## sha1 key
#    src                 dst                 protocol spi      encr_algo  key 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
add  fc10:f510:3:82::1  fc10:f510:3:77::1      qah 0x300 -A hmac-sha1  0x3ffe05014819fff819fff3f3ffe05014819ffff;
add  fc10:f510:3:77::1  fc10:f510:3:82::1      qah 0x301 -A hmac-sha1  0x3ffe05014819fff819fff3f3ffe05014819ffff;

add  fc10:f510:3:77::1  fc10:f510:3:7d::1      qah 0x302 -A hmac-sha1  0x3ffe05014819fff819fff3f3ffe05014819ffff;
add  fc10:f510:3:7d::1  fc10:f510:3:77::1      qah 0x303 -A hmac-sha1  0x3ffe05014819fff819fff3f3ffe05014819ffff;

add  fc10:f510:3:82::1  fc10:f510:3:7d::1      qah 0x304 -A hmac-sha1  0x3ffe05014819fff819fff3f3ffe05014819ffff;
add  fc10:f510:3:7d::1  fc10:f510:3:82::1      qah 0x305 -A hmac-sha1  0x3ffe05014819fff819fff3f3ffe05014819ffff;
```

Figure 15 Configuration file for setkey at NetAH/ESP IP hosts along the route.

4.1.3 Implementation of NetAH in setkey

The setkey application has been extended by the following:

- Interpretation and signaling of the protocol NetAH in the SPD in addition to the existing ESP, AH and IP Payload Compression IPcomp[13]. The NetAH protocol is specified by the 'qah' token in the SPD part of the setkey configuration file.

  The protocol is a part of the *sadb_x_policy* message and for NetAH the *proto* parameter is 254.

- Interpretation and signaling of the protocol NetAH in the SAD in addition to the existing ESP based on RFC2406, ESP based on RFC1827, AH based on RFC2402, AH based on RFC1826, IPComp and TCP-MD5 based on RFC2385. The NetAH protocol is specified by the 'qah' token in the SAD part of the setkey configuration file.

  The protocol is a part of the *sadb_msg* message and for NetAH the the *sadb_msg_satype* parameter is 10.

Table 2 shows the files in the IPsec-tools package that have been modified in order to implement NetAH support:

Table 2 Overview of modified files

File	Change	New/ modified
/src/libipsec/policy_token.l	Added action on IPPROTO_QAH	mod
/src/libipsec/pfkey.c	Added handling of SADB_SATYPE_QAH	mod
/src/libipsec/ipsec_dump_policy.c	Added handling of IPPROTO_QAH	mod
/src/setkey/token.l	Added handling of IPPROTO_QAH	mod
/src/setkey/parse.y	#define SADB_SATYPE_QAH	mod

The .l and .y files are Lex/Yacc files which generates the C-code for the parser in IPsec-tools.

4.2 Quality of Service

4.2.1 Classification of IP traffic

The IP traffic is classified by *ip6tables*. See example in section 6.1.1 or www.netfilter.org.

Whereas ip6tables is also available in the radios, during our tests, the traffic was classified outside the tactical communication nodes.

4.2.2 Shaping of IP traffic

The IP traffic is shaped prior to transmission over the wireless media. The shaping is done on the basis of the classification provided by ip6tables.

Shaping by Linux traffic control tool *tc*. The *tc* tool consists of three parts:

1. Class definition: Example : `tc class add dev uhf0 parent 1: classid 1:10 htb rate 400Mbit ceil 400 kbps`
2. Queue discipline : Example : `tc qdisc add dev uhf0 parent 1:10 handle 10: pfifo limit 5`
3. Filtering : Example : `tc filter add dev uhf0 parent 1:0 protocol ipv6 prio1 u32 match u8 0x0c 0x0f at flowid 1:20`

The following command shows the statistics: tc –s –d qdisc show dev uhf0

4.2.3 Mapping between DSCP values and radio priority queues

The tactical communication nodes offer 7 different priorities with a corresponding Hierarchical Token Bucket (htb) queue. The IP traffic is mapped the queue based on the Differentiated Service Code Point (DSCP) values, i.e. the 6 most significant bits of the Traffic Class field in Figure 12. This field is used by the radios for packet classification purposes when prioritizing traffic prior to transmission over the air. See scripts in Figure 16 and Figure 17.

```
tc qdisc del dev uhf0 root  2>/dev/null
tc qdisc add dev uhf0 root handle 1: htb default 60 r2q 1
tc class add dev uhf0 parent 1:   classid 1:1 htb  rate $SPEED ceil $SPEED

# mainly for critical icmpv6 messages
tc class add dev uhf0 parent 1:1  classid 1:10 htb rate 15kbit ceil  $SPEED #CS7
# high priority networking protocols i.g. routing ++
tc class add dev uhf0 parent 1:1  classid 1:20 htb rate 10kbit ceil  $SPEED #CS6 (routing)
tc class add dev uhf0 parent 1:1  classid 1:30 htb rate 300kbit ceil $SPEED #CS3-CS5
tc class add dev uhf0 parent 1:1  classid 1:40 htb rate 25kbit ceil  $SPEED #CS2
tc class add dev uhf0 parent 1:1  classid 1:50 htb rate 25kbit ceil  $SPEED #CS1
tc class add dev uhf0 parent 1:1  classid 1:60 htb rate 25kbit ceil  $SPEED #BE

tc qdisc add dev uhf0 parent 1:10 handle 10: pfifo limit 10
tc qdisc add dev uhf0 parent 1:20 handle 20: pfifo limit 7
tc qdisc add dev uhf0 parent 1:30 handle 30: pfifo limit 200
tc qdisc add dev uhf0 parent 1:40 handle 40: pfifo limit 16
tc qdisc add dev uhf0 parent 1:50 handle 50: pfifo limit 16
tc qdisc add dev uhf0 parent 1:60 handle 60: sfq perturb 10

tc filter add dev uhf0 parent 1:0 protocol ipv6 prio 0 u32 match u8 0x0e 0x0f at 0 flowid 1:10
tc filter add dev uhf0 parent 1:0 protocol ipv6 prio 1 u32 match u8 0x0c 0x0f at 0 flowid 1:20
tc filter add dev uhf0 parent 1:0 protocol ipv6 prio 2 u32 match u8 0x0a 0x0f at 0 flowid 1:30
tc filter add dev uhf0 parent 1:0 protocol ipv6 prio 3 u32 match u8 0x08 0x0f at 0 flowid 1:30
tc filter add dev uhf0 parent 1:0 protocol ipv6 prio 4 u32 match u8 0x06 0x0f at 0 flowid 1:30
tc filter add dev uhf0 parent 1:0 protocol ipv6 prio 5 u32 match u8 0x04 0x0f at 0 flowid 1:40
tc filter add dev uhf0 parent 1:0 protocol ipv6 prio 6 u32 match u8 0x02 0x0f at 0 flowid 1:50
tc filter add dev uhf0 parent 1:0 protocol ipv6 prio 7 u32 match u8 0x00 0x0f at 0 flowid 1:60
```

Figure 16 tc configuration script used for air interface (uhf0) (SPEED = 400 Kbit, ie. optimized for 920 Kbit/sec)

```
# flush previously? defined rules, for debug purpose
iptables -t nat    -F
iptables -t mangle -F
iptables -F

ip6tables -t mangle -F
ip6tables -F

# disable ipv4 networking
iptables -P INPUT   DROP
iptables -P OUTPUT  DROP
iptables -P FORWARD DROP

# for debug purposes - don't keep them in the dark
iptables -A INPUT -i eth0      -j REJECT --reject-with icmp-net-prohibited
iptables -A INPUT -i uhf0 -j REJECT --reject-with icmp-net-prohibited

# ipv6 networking

# be restrict - by default drop all
ip6tables -P OUTPUT DROP
ip6tables -P INPUT  DROP
# but allow anything on the wired side
ip6tables -A INPUT  -i eth0 -j ACCEPT
ip6tables -A OUTPUT -o eth0 -j ACCEPT

# allow ipv6 networking on wired, but restamp high prio packets first
ip6tables -t mangle -A FORWARD  -i eth0 -m tos --tos 0xe0/0xe0 -j TOS --set-tos 0xa0
ip6tables -t mangle -A FORWARD  -i eth0 -m tos --tos 0xc0/0xc0 -j TOS --set-tos 0xa0

# give routing high priority
ip6tables -t mangle -A OUTPUT -o uhf0 -p 89 -j TOS --set-tos 0xc0

# allow icmp on wireless (could be more restrict)
ip6tables -A OUTPUT -o uhf0 -p icmpv6 -j ACCEPT
ip6tables -A INPUT  -i uhf0 -p icmpv6 -j ACCEPT

# allow ospf on wireless
ip6tables -A OUTPUT -o uhf0 -p 89 -j ACCEPT
ip6tables -A INPUT  -i uhf0 -p 89 -j ACCEPT

# allow multicast on wireless
ip6tables -A OUTPUT -o uhf0 -d ff08::/8 -j ACCEPT
ip6tables -A INPUT  -i uhf0 -d ff08::/8 -j ACCEPT

#allow link local on wireless (useful during debug)
ip6tables -A OUTPUT -o uhf0 -d fe80::/16 -j ACCEPT
ip6tables -A INPUT  -i uhf0 -d fe80::/16 -j ACCEPT

# finally allow qah (hop-by-hop)
ip6tables -A OUTPUT -o uhf0 -m ipv6header --header hop-by-hop,esp -j ACCEPT
ip6tables -A INPUT  -i uhf0 -m ipv6header --header hop-by-hop,esp -j ACCEPT

#finally, for debug purposes - don't keep them in the dark
ip6tables -A OUTPUT -o uhf0 -j REJECT --reject-with adm-prohibited
ip6tables -A INPUT  -i uhf0 -j REJECT --reject-with adm-prohibited
```

Figure 17 Firewall script used by iptables in radios

4.3 Routing

The tactical communication nodes are able to run OLSR[1] or OSPF-MANET [12] routing protocols for multi hop routing over the wireless interface. The quagga router was configured to run OSPFv3 Manet Designated Router from Boeing. See http://hipserver.mct.phantomworks.org/ietf/ospf/

OSPFv3 [3] was configured for the wired interface.

5. Test network topology

Figure 18 - Figure 20 show the test network topology. All three figures show the same test network with different views. Figure 18 shows the equipment and IPsec architecture. Figure 19 shows the IP address structure and Figure 20 shows the routing domains and protocols.

Both PC A and PC B communicate with PC C via two radio links and a router. Two ESP and AH transport mode links are established: PC A - PC C and PC B - PC C. With this topology it is possible to create congestion at the WM600(C)-SR600(C) radio link. Manipulated IP packets may then displace the intended high priority traffic. Further, protection of the high priority traffic by NetAH as a countermeasure can be tested.

The test was carried out with different routers, a standard COTS router and a TR600 router in order to verify that the NetAH is compliant to standard IP requirements.

The manipulation of IP packets were carried out in a router by using the Linux application ip6tables.

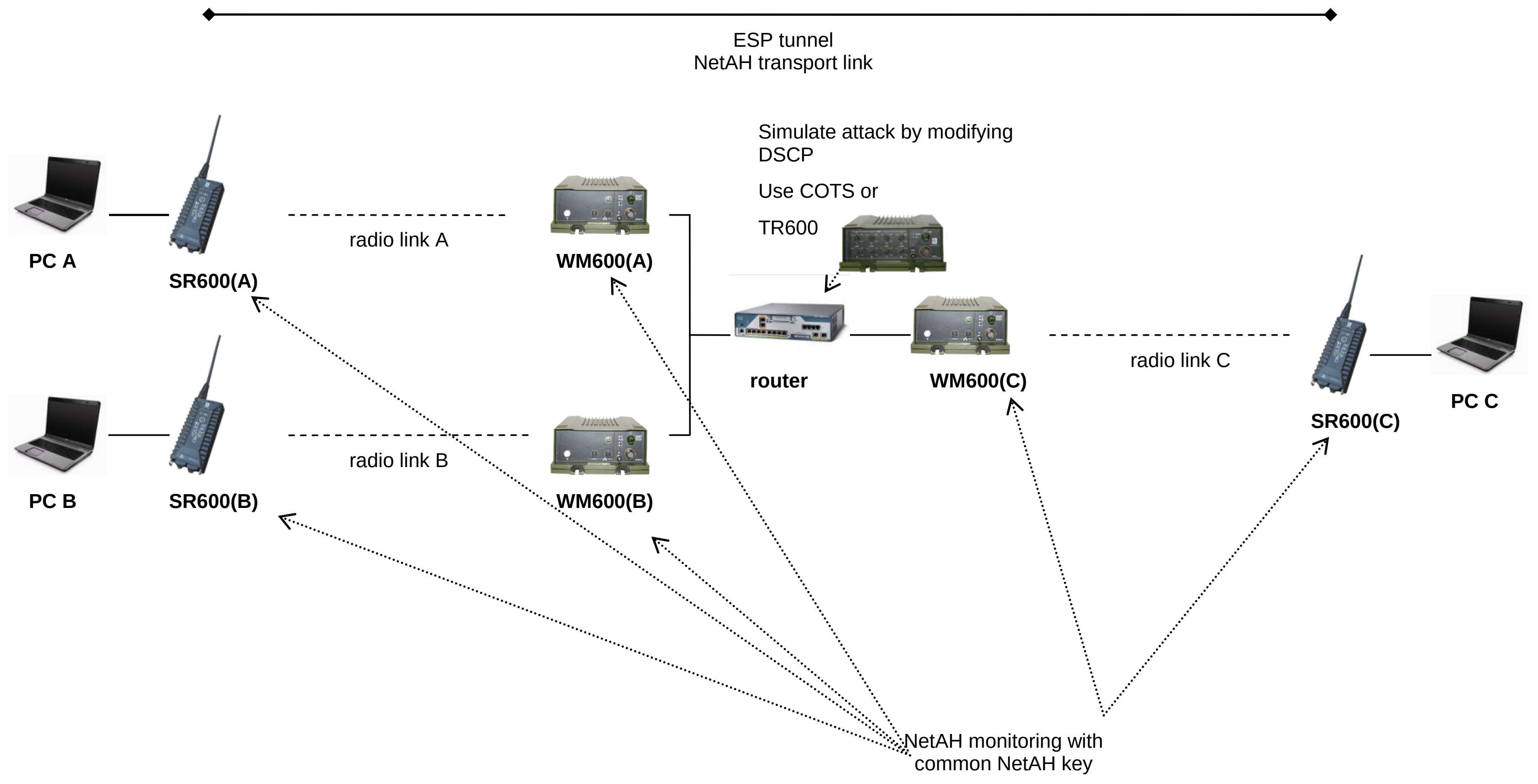

Figure 18 Test network (IPsec view)

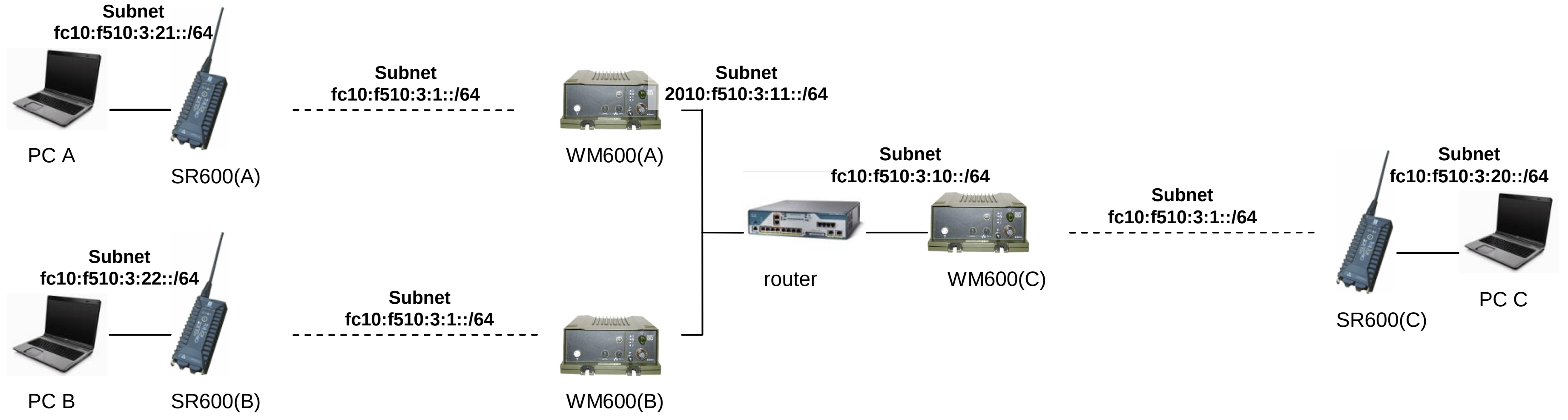

Figure 19 Test network (IP address view)

All radio subnets are fc10:f510:3:1::/64. All Ethernet subnets of the radios are fc10:f510:3:x::/64, where x is unique.

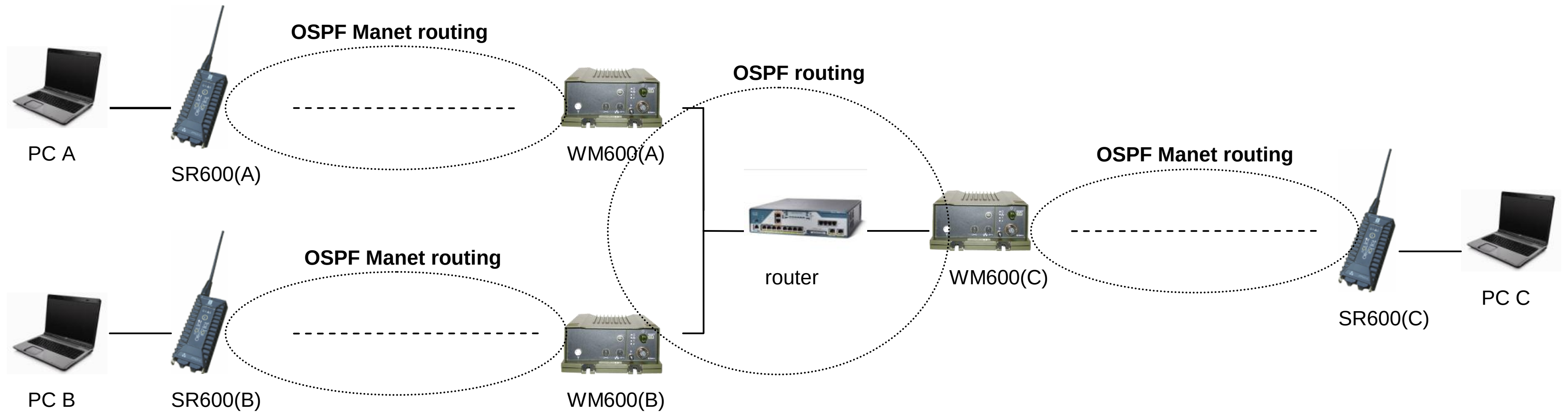

Figure 20 Test network (Routing domain view)

6. Test scenarios

The main objectives of the tests are:

1. Verify that the handling of the IP packets when the packets are passing through the standard IPv6 router is not affected by the NetAH.
2. Verify that NetAH IP packets are handled correctly. This means that manipulated NetAH packets are forwarded with lowest priority. When congestion occurs, the packets with failing NetAH will be delayed or dropped.
3. Demonstrate that use of NetAH has higher immunity against manipulation than standard AH

Six different test scenarios are identified in order to achieve the objectives of the tests. For all scenarios except from scenario 2 the radio link C will be a bottleneck for the traffic flow generated at PC A and PC B with PC C as the destination. Congestion will occur and the WM600(C) radio has to drop packets. All scenarios use TCP traffic generated by iperf, where iperf runs as client at PC A and PC B and as server at PC C.

The test scenarios are indicated in Figure 21 - Figure 26. Note that all test scenarios are according to the test setup described in section 5.

Figure 21 - Figure 26 are simplified in the sense that they only shows the PCs (the sources and destination of the user traffic), the router and the WM600(C) radio where the congestion may occur.

An overview of the purpose of each test scenario is shown in Table 3.

Table 3 Test scenarios

Scenario	Purpose	AH	Router
1	Reference setup. Find maximum throughput. The end to end throughput is expected to be higher than when using AH and NetAH	none	COTS
2	Verify that a COTS router handles the NetAH. No congestion.	NetAH	COTS
3	Verify that a COTS router handles the NetAH. Find maximum throughput. The end to end throughput is expected to be a bit lower than in scenario 1.	NetAH	COTS
4	Verify that the TR600 router is able to handle NetAH	NetAH	TR600
5	Demonstrate consequences of manipulated AH. The priority of the lowest priority traffic is set higher than the high priority traffic. Then the intended high priority traffic will be dropped when congestion occurs at the entrance to the bandwidth limited radio link.	AH.	TR600 or COTS manipulating IP packets
6	Demonstrate immunity against manipulation for NetAH. The priority of the lowest priority traffic is set higher than the high priority traffic in the same way as in scenario 5. When congestion occurs at the entrance to the bandwidth limited radio link, the manipulation will be detected and the manipulated packets will be dropped for the benefit of the high priority traffic.	NetAH	TR600 or COTS manipulating IP packets

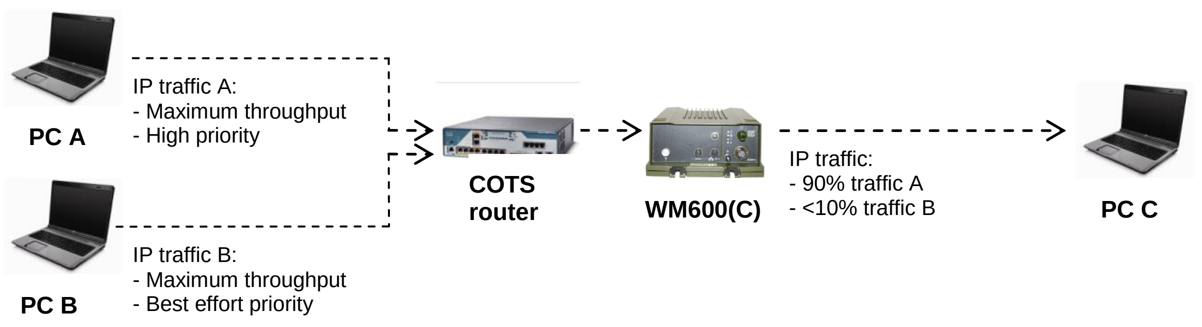

Figure 21 Test scenario 1: Reference setup

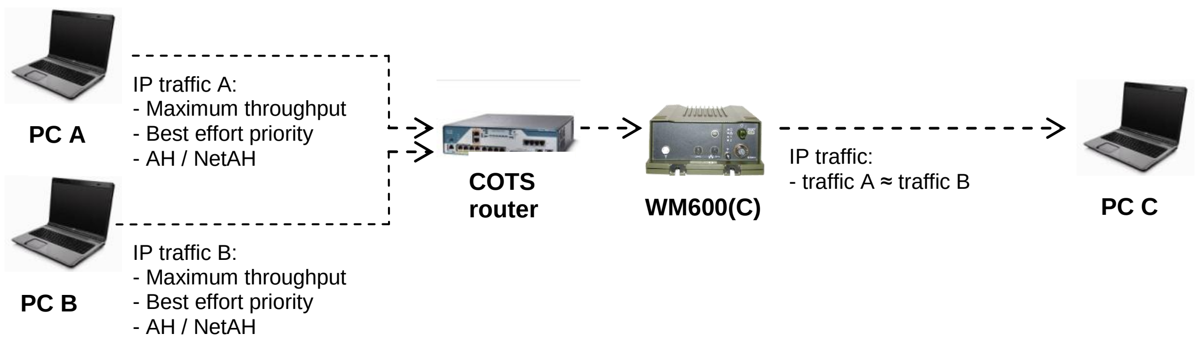

Figure 22 Test scenario 2: Verify use of NetAH via COTS router, no priority differences

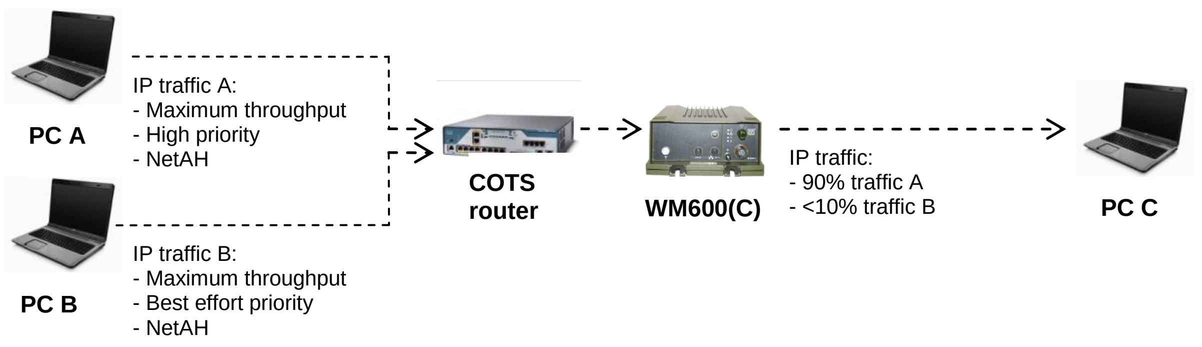

Figure 23 Test scenario 3: Verify prioritizing of high priority packets when congestion occurs and using NetAH

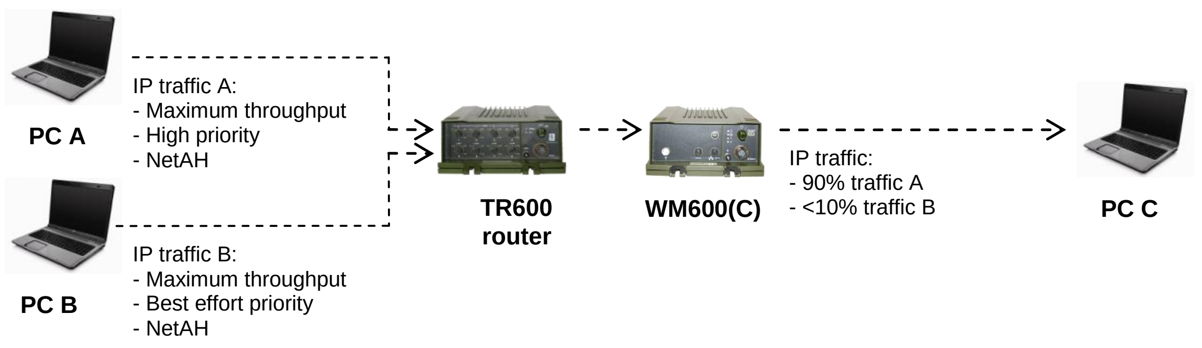

Figure 24 Test scenario 4: Verify use of NetAH via TR600 tactical router

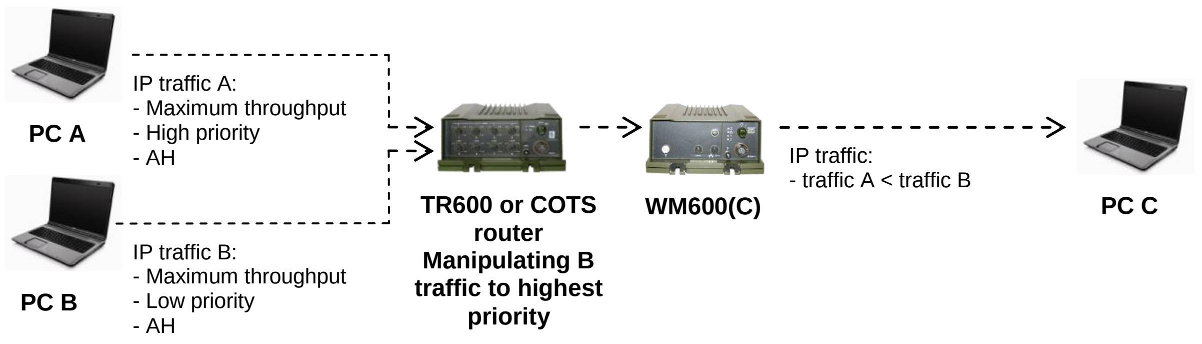

Figure 25 Test scenario 5: Demonstrate consequences of manipulated AH

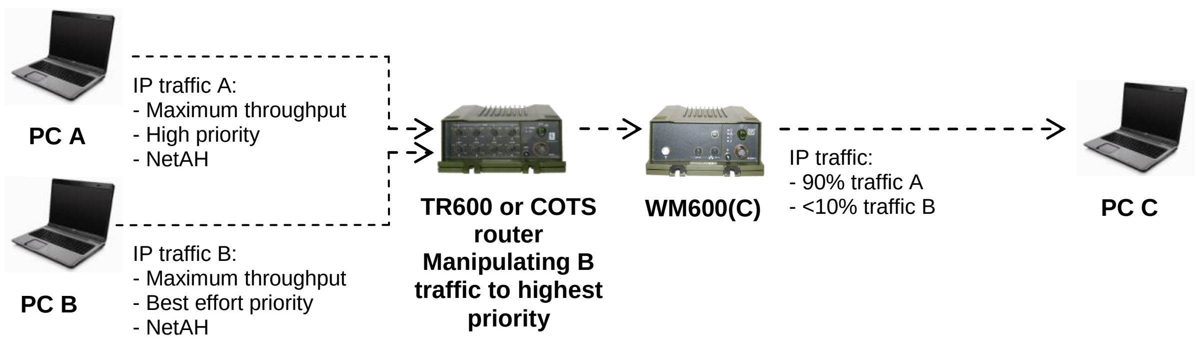

Figure 26 Test scenario 6: Verify handling of packets with manipulated NetAH

6.1 PC and SR/WM setup

6.1.1 Manipulation of DSCP field

The mapping between the DSCP values and radio priority queues were described in section 4.2.3.

The DSCP is set in the originating IP hosts (PC A and PC B) by using the ip6table application:

PC A: `ip6tables -A OUTPUT -t mangle -p !icmpv6 -j TOSP --set-tos 0xc0`

PC B: `ip6tables -A OUTPUT -t mangle -p !icmpv6 -j TOSP --set-tos 0xc0`

The DSCP field is manipulated in the router by:

```
ip6tables -A FORWARD -t mangle -p !icmpv6 -j TOSP --set-tos 0xc0
```

6.1.2 IPERF

The iperf application is used for generation of IP traffic through the network. Iperf must run in both peers that generate the traffic flow, as a client at the source and as a server at the destination. The traffic is measured and reported by the server at the destination.

Server setup:

```
iperf -s -V                         // V=ipv6
```

Client setup

```
iperf -c fc10:f510:3:11::2 -m -V    // m=display maximum segment size
```

For information: the similar setup for UDP traffic could be:

```
iperf -s  -u -i 1 -V                       // V=ipv6 u=UDP i=interval 1=1s
iperf -c fc10:f510:3:11::2 -u -b 100k -V   // u=UDP b=bandwidth 100k=100kbps
```

6.2 Test procedures and results

Table 4 Test procedures

Proc	Description	Expected result	Result
1a	Purpose: Find maximum throughput without AH or NetAH. Test network topology: Test scenario 1. Please note that the Test scenario 1 data flow indications are valid for Proc 1b. 1. Generate TCP traffic from PC A to PC C (path AC) by using iperf. 2. Generate TCP traffic from PC B to PC C (path BC) by using iperf. 3. Find the maximum throughput with equal and concurrent TCP traffic over the two paths AC and BC. (MSS=1428 / MTU=1500)		1. AC_{Max} = 389 kbps 2. BC_{Max} = 386 kbps 3. AC = 205 kbps BC = 185 kbps
1b	Purpose: Verify that SR600 discards low priority packets when congestion occurs. Test network topology: Test scenario 1. 1. Use iptables to manipulate the DS field settings at PC A and PC B. Set PC A to high priority traffic and PC B to low priority traffic. 2. Generate TCP traffic via path AC using iperf with maximum throughput. 3. Generate TCP traffic via path BC using iperf with maximum throughput. (MSS=1428 / MTU=1500)	3. AC traffic throughput as in 1a1. A small fraction of BC traffic is received in PC C.	3. AC = 380 kbps BC = 34 kbps
2a	Purpose: Find maximum throughput with AH. Test network topology: Test scenario 2. Repeat 1a using AH. (MSS=1342 / MTU=1382)	Maximum end to end throughput is expected to be a bit lower than in 1a.	1. AC_{Max} = 362 kbps 2. BC_{Max} = 325 kbps 3. AC = 213 kbps BC = 137 kbps
2b	Purpose: Verify that a COTS router handles the NetAH. Find maximum throughput with NetAH. Test network topology: Test scenario 2. Repeat 1a using NetAH. SPD and SAD shall be set in all hosts. (MSS=1342 / MTU=1382)	Maximum end to end throughput is expected to be the same as for 2a.	1. AC_{Max} = 314 kbps 2. BC_{Max} = 338 kbps 3. AC = 211 kbps BC = 134 kbps
3a	Purpose: Verify that low priority traffic is dropped when congestion occurs when using NetAH. Test network topology: Test scenario 3. Repeat 1b using NetAH. (MSS=1342 / MTU=1382)	Similar results as for 1b.	3. AC = 410 kbps BC = 34 kbps
4a	Purpose: Verify use of NetAH via TR600 router. Test network topology: Test scenario 4. Repeat 3a using TR600 router (MSS=1342 / MTU=1382)	Similar results as for 3a.	
5a	Purpose: Demonstrate consequences of manipulated DS when using AH. Test network topology: Test scenario 5. 1. Use iptables to manipulate the DS field settings at PC A and PC B. Set PC A to high priority traffic and PC B to low priority traffic. (Traffic Class = 0x60)		

Proc	Description	Expected result	Result
	2. Use iptables to manipulate the DS field settings at the router for the traffic generated in PC B. The DS field priority shall be set higher than for PC A. (Traffic Class = 0xA0) 3. Set up the PCs to run AH. 4. Generate TCP traffic via path AC using iperf with maximum throughput. 5. Generate TCP traffic via path BC using iperf with maximum throughput. (MSS=1342 / MTU=1382)	5. BC traffic throughput higher than AC traffic throughput.	5. AC = 167 kbps BC = 244 kbps
6a	Purpose: Verify handling of packets with manipulated DS field using NetAH. Test network topology: Test scenario 5. Procedure is similar to 5a, except that the hosts run NetAH instead of AH. (MSS=1342 / MTU=1382)	5. AC traffic throughput as in 4a. A small fraction of BC traffic is received in PC C.	5. AC = 391 kbps BC = 42 kbps

7. Test network – code updates

7.1 Updated Linux Kernel code

Table 5 provides a summary of the modifications done to the Linux Kernel code.

Table 5 Overview of updated kernel code

File	Change	New/modified	Processing

include/linux/in.h	IPPROTO_QAH = 254
	proto_ports_offset()
		new
		mod	

include/linux/in6.h	#define IPV6_TLV_QAH 30	new	
include/linux/ip.h	struct ip_qauth_hdr	new	
include/linux/pfkeyv2.h	#define SADB_SATYPE_QAH 10	new	
include/net/ah.h	*ip_qauth_hdr()	new	
include/net/ipv6.h	#define NEXTHDR_QAUTH 254	new	
include/net/xfrm.h	#define XFRM_PROTO_QAH 254 xfrm_id_proto_match()	new mod	

net/ipv6/ah6.c	qah6_output()
	qah6_output_done()
	qah6_input()
	qah6_input_done()
	qah6_err()
	qah6_type
	qah6_protocol
		new	outbound
			outbound
			inbound
			inbound
			out/inbound
			out/inbound
			out/inbound

net/ipv6/exthdrs.c	ipv6_hop_qah()
	ipv6_parse_hopopts()
		new
		mod
			hop-by-hop
			hop-by-hop

net/ipv6/exthdrs_core.c	ipv6_ext_hdr()	mod	hop-by-hop
net/ipv6/datagram.c	datagram_recv_ctl()	mod	hop-by-hop
net/ipv6/ip6_input.c	ip6_input_finish()	mod	inbound
net/ipv6/xfrm6_input.c	xfrm6_rcv_spi()	mod	inbound/hop-by-hop
net/ipv6/xfrm6_state.c	__xfrm6_state_sort_cmp()	mod	inbound/hop-by-hop
net/ipv6/xftm6_policy.c	_decode_session6()	mod	inbound/hop-by-hop

net/key/af_key.c	pfkey_satype2proto()
	pfkey_proto2satype()
	pfkey_msg2xfrm_state()
	pfkey_send_aquire()
		mod
		mod
		mod
		mod
			communication with setkey

net/xfrm/xfrm_input.c	xfrm_parse_spi()	mod	out/inbound

net/xfrm/xfrm_user.c	verify_newsa_info()
	verify_userspi_info()
		mod
		mod
			creation of SA

7.2 Function call chains

Generate IP packet with AH / NetAH
Generation of IP packet is initiated by one of the following function calls:

```
dccp_v6_send_responce()                 // net/dccp/ipv6.c
inet6_sk_rebuild_header()               // net/ipv6/af_inet6.c
ip6_datagram_connect()                  // net/ipv6/datagram.c
icmpv6_send()                           // net/ipv6/icmp.c
inet6_connection_sock()                 // net/ipv6/inet_connection_sock.c
udpv6_sendmsg()                         // net/ipv6/udp.c
tcp_v6_connect()                        // net/ipv6/tcp_ipv6.c
tcp_v6_err()                            // net/ipv6/tcp_ipv6.c
tcp_v6_send_synack()                    // net/ipv6/tcp_ipv6.c
tcp_v6_send_reset()                     // net/ipv6/tcp_ipv6.c
tcp_v6_send_ack()                       // net/ipv6/tcp_ipv6.c
tcp_v6_send_syn_recv_sock()             // net/ipv6/tcp_ipv6.c
rawv6_sendmsg()                         // net/ipv6/raw.c
```

..and the further function calls are:

```
 // Find xfrm_policy as IPsec policy
 xfrm_lookup(dst_entry, flowi, sock, int)      // net/xfrm/xfrm_policy.c
   flow_cache_lookup(flowi,…)                  // net/core/flow.c

     // Find policy to apply to this flow.
     xfrm_policy_lookup()                      // net/xfrm/xfrm_policy.c

   // Try to find matching bundle               // net/xfrm/xfrm_policy.c
   xfrm_find_bundle()

     // Check that bundle accepts flow and its components are still valid
     x = afinfo->find_bundle()
       __xfrm6_find_bundle()
         if (ipv6_addr_equal() & xfrm_ bundle_ok()) dst_clone(dst)
     xfrm_policy_put_afinfo(afinfo)

   // Reslove list of templates for the flow, given policy (lookup SA).
   xfrm_tmpl_resolve()                         // net/xfrm/xfrm_policy.ce
     xfrm_state_find()                         // net/xfrm/xfrm_policy.c

   // Allocate chain of dst_entrys, attach known xfrms, calc metrics…(create output chain)
   xfrm_bundle_create()                        // net/xfrm/xfrm_policy.c
     err = afinfo->bundle_create()
       __xfrm6_bundle_create()                 // net/ipv6/xfrm6_policy.c/
         xfrm6_output() / xfrm6_output_finish()  // net/ipv6/xfrm6_output.c/
           xfrm6_output_one()
             struct xfrm_state *x = dst->xfrm;
```

```
              xfrm6_encap()
              err = x->type->output(x,skb)            // = ah6_output()
                ah6_output(struct xfrm_state *x, struct sk_buff *skb)
```

Hop by hop processing

```
ipv6_rcv()                                  // net/ipv6/ip6_input.c
  ipv6_parse_hopopts()                      // net/ipv6/exthdrs.c
    ipv6_parse_tlv()
      if (curr->func(skb, off) == 0)        //   tlvprochopopt_lst[]
        ipv6_hop_qah()
  ip6_rcv_finish()                          // net/ipv6/ip6_input.c
    (ip6_route_input())                     // net/ipv6/ip6_input.c
    dst_input()                             // include/net/dst.h
      err = skb->dst->input(skb)            // = ip6_forward()
        ip6_forward()                       // net/ipv6/ip6_input.c
          xfrm6_policy_check()              // include/net/xfrm.h
            xfrm_policy_check()             // include/net/xfrm.h
              __xfrm_policy_check()
                xfrm_decode_session()       // net/xfrm/xfrm_policy.c
                  _decode_session6()        // net/ipv6/xfrm6_policy.c
          xfrm6_route_forward()             // net/ipv6/ip6_output.c
            xfrm_route_forward()            // net/xfrm/xfrm_policy.c
              __xfrm_route_forward()        // net/xfrm/xfrm_policy.c
                xfrm_decode_session()       // net/xfrm/xfrm_policy.c
                  _decode_session6()        // net/ipv6/xfrm6_policy.c
```

Inbound processing

```
ipv6_rcv()                                  // net/ipv6/ip6_input.c
  ipv6_parse_hopopts()                      // net/ipv6/exthdrs.c
    ipv6_parse_tlv()
      if (curr->func(skb, off) == 0)        //   tlvprochopopt_lst[]
        ipv6_hop_qah()
  ip6_rcv_finish()                          // net/ipv6/ip6_input.c
    ip6_route_input()                       // net/ipv6/ip6_input.c

ip6_input_finish()                          // net/ipv6/ip6_input.c
 /* parse extension headers */
 if ((nexthdr == ...) && ((skb->nh.raw[nhoff+36] == IPV6_TLV_QAH))
   hash = IPPROTO_QAH
 :
 xfrm6_policy_check()
   xfrm_policy_check()

ip6_push_pending_frames()

xfrm6_rcv()                                 // net/ipv6/xfrm6_input.c
  xfrm6_rcv_spi()                           // do {...} while
    if ((nexthdr == ...) && ((skb->nh.raw[nhoff+36] == IPV6_TLV_QAH))
        nexthdr = IPPROTO_QAH
    xfrm_parse_spi(nexthdr)
    do {
        xfrm_state_lookup(nexthdr)                // net/xfrm_state.c
           afinfo = xfrm_state_get_afinfo(famliy)
               afinfo = xfrm_state_afinfo[famlily]
           x = afinfo->state_lookup()
                 = __xfrm6_state_lookup()
                    proto == x->id.proto
           xfrm6_state_lookup(nexthdr)
        :
        nexthdr = x->type->input(x, skb)         // = ah6_input()

        xfrm_parse_spi(nexthdr)
    } while
```