Eidgenössische Technische Hochschule Zürich
Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Swiss Federal Institute of Technology Zurich

Departement Informationstechnologie und Elektrotechnik

Institut für Technische Informatik und Kommunikationsnetze

Security in Mobile Ad-Hoc Networks

Axel Burri

Diploma Thesis DA-2002.29
Summer Term 2002

Tutors: Vincent Lenders, Natalie Weiler
Supervisor: Prof. Dr. Bernhard Plattner

October 20th 2002

Zusammenfassung

In der jüngsten Vergangenheit hat das Bestreben nach Miniaturisierung und schnelleren Prozessoren zu einer neuen Generation vom mobilen Geräten geführt, welche oft zur drahtlosen Kommunikation fähig sind. Neue verteilte Anwendungen werden mit solchen Geräten möglich: manchmal besteht der Bedarf, diese Geräte zu einem *Ad Hoc Netzwerk* zu verbinden, besonders dann wenn keine Infrastruktur verfügbar ist. Zahlreiche Ad Hoc Routingprotokolle wurden von der Community [1] vorgeschlagen, Sicherheitsaspekte wurden aber meistens vernachlässigt.

Im Rahmen einer Diplomarbeit wurde ein Protokoll für mobile Ad Hoc Netzwerke geschaffen, um aus mehreren möglichen Routen diejenige auszusuchen, welche aufgrund von Vertrauens- und Quality of Service (QoS) Betrachtungen im Augenblick die geeignetste ist. Erreicht wird dies durch Erweiterung eines bestehenden Routingprotokolls, dem *Dynamic Source Routing* (DSR): jeder Host kann auf Anfrage seine momentane QoS signieren und verteilen. Anhand der übermittelten Signatur wird anschliessend ein Tupel [QoS, Trust] für den betreffenden Host ermittelt, welcher entscheidend für die Auswahl der optimalen Route ist.

So können nicht vertrauenswürdige und überlastete Knoten vom Routing ausgeschlossen werden, solange eine Verbindung über vertrauenswürdige Knoten möglich ist. Es kann davon ausgegangen werden, dass ein vertrauenswürdiger Knoten seine QoS korrekt berechnet und übermittelt, während es sich bei einem nicht vertrauenswürdigen Knoten um einen Angreifer handlen könnte, der seine QoS-Werte nach belieben modifiziert, und welchen man möglichst vom Routing ausschliessen möchte.

So ist es auch möglich, inmitten eines bestehenden DSR-Netzwerkes als kleinere Gruppe ein privates Netzwerk bestehend aus ausschliesslich vertrauenswürdigen Hosts zu bilden, ohne die anderen zu beeinträchtigen. Dies ist vor allem dann wichtig, wenn von einer Anwendung eine maximale QoS (Durchsatz, Stabilität) und hohes Vertrauen für die Verbindung verlangt wird.

Abstract

In recent years, the trend of miniaturization and growing processing power has led to a new generation of mobile devices, often with wireless communication capabilities. New distributed applications are possible with these devices: sometimes, when no infrastructure is available, there is still requirement to connect these devices together, forming an *ad hoc* network. Many ad hoc routing protocols have been proposed by the community [1]. However, security issues have mostly been neglected.

Within the scope of a diploma thesis a protocol designed for use in mobile ad hoc networks has been created, in order to choose the best route on the basis of trust and quality of service (QoS) considerations. This is being achieved by extending *Dynamic Source Routing* (DSR): each host can sign and distribute his current QoS on request. Based on the submitted signature, a [QoS, trust] tuple for this host is being calculated, which is deciding for the choice of the optimal route.

This way, untrustworthy and overloaded nodes can be excluded from routing, as long as a connection over trustworthy nodes is still possible. It is assumed that trusted nodes compute and transmit their QoS correctly, whereas a untrusted node could be an attacker who modifies his QoS at will, and therefore should be excluded from the routing.

This way it also becomes possible for a small group of trusted people to form a private network in the middle of an existing DSR network without bothering the others. This becomes especially important if an application requires the connection to meet with maximal quality of service (such as throughput or stability) and trust values.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einführung

1.1 Motivation

Obschon der Trend beim Wireless Networking eher in Richtung einer Sternförmigen Architektur (1 Basestation, N Hosts) verläuft, gibt es, solange es nicht möglich ist, überall auf der Welt eine Internetanbindung zu erhalten, dennoch zahlreiche Anwendungsgebiete für Ad Hoc Netzwerke.

Sicherheit spielt dabei eine immer wichtigere Rolle. Sei es die Sicherheit, dass Informationen für ein möglichst breites Publikum zugänglich sind, oder dass die angeforderten Informationen authentisch, unverfälscht oder vertraulich sind.

Ausgerechnet auf Sicherheitsaspekte wurde bei den zahlreichen Ad Hoc Networking Protokollen [2, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28] nur vereinzelt oder gar nicht eingegangen. So ist es bei diversen Protokollen ohne weiteres möglich, grosse Teile des Netzwerkes lahmzulegen. (siehe Kap. 3.2)

1.2 Anwendungsgebiete

In diesem Abschnitt werden zwei Fallbeispiele für Mobile Ad Hoc Netzwerke diskutiert, welche verschiedene Aspekte von Sicherheit beinhalten.

1.2.1 Polizeieinsatz

Ein Bankraub, die Bank wird von der Polizei umstellt. Einzelne Polizisten möchten über ein Ad Hoc Netzwerk miteinander Daten austauschen. Sie sind auf hohe Verfügbarkeit und Vertraulichkeit des Netzes angewiesen und werden ausschliesslich die Geräte ihrer Kollegen als Router benützen; keinesfalls etwa die eines Passanten, welcher sein Gerät mal kurz angeschaltet hat. Falls nämlich der Passant sein Gerät wieder abschaltet ist unter umständen das Netzerk unterbrochen und die Polizisten müssen ihren Standort verändern um es wiederherzustellen. Die Schlüssel jedes einzelnen Polizisten

werden von der obersten Dienststelle vorab signiert, damit Authentizität gewährleistet ist.

1.2.2 Kommunikation auf der Autobahn

Stellen wir uns vor, dass in jedem Fahrzeug ein Computer mit Funkkarte steckt. Ausserdem sind am Strassenrand vereinzelt Internet-Gateways mit derselben Funktechnik aber stärkeren Signalpegeln. Nun wollen zwei dieser Fahrzeuge miteinander kommunizieren, etwa um sich in der nächsten Raststätte zu verabreden, es besteht aber keine direkte Verbindung. Wie sollten sich nun die Geräte verhalten?

1. Die Gateways rauschen mit 120km/h an uns vorbei, bieten also nur äusserst kurzfristige Verbindungen und sind demnach schlecht für Routing geeignet. Entsprechend tief müssen dessen QoS Werte ausfallen.

2. Wir möchten eine Route entlang der fahrenden Autos etablieren, also müssen die dessen QoS Werte entsprechend hoch sein.

Falls nun also eine Verbindung zustande kommt, dann prioritär eine stabilere über die fahrenden Autos.

Kapitel 2

Zielsetzung

In diesem Kapitel wird eine Zielsetzung für Sicherheit in Ad Hoc Netzwerken definert. Im Rahmen einer Diplomarbeit ist es allerdings nicht möglich, auf alle Aspekte gleichermassen einzugehen, da jeder einzelne für sich ein äusserst komplexes Gebiet umspannt. Hauptaugenmerke hier sind Verfügbarkeit und Authentisierung.

2.1 Sicherheit

Sicherheit wird in der Literatur meist in fünf Teilaspekte unterteilt, welche im folgenden besonders in Hinblick auf Mobile Ad Hoc Netzwerke (MANET's) genauer untersucht werden:

1. availability (Verfügbarkeit)
2. confidentiality (Vertraulichkeit)
3. integrity (Integrität)
4. authentication (Authentisierung)
5. non-repudiation (Unleugbarkeit, Nachweisbarkeit)

2.1.1 Availability

Verfügbarkeit ist für die meisten Anwendungsgebiete wohl der wichtigste Aspekt eines Informationsnetzwerkes, da sie das Überleben dessen Dienste sichert. Dies gilt insbesondere auch für MANET's. Gerade hier ist es besonders schwierig, eine hohe Verfügbarkeit zu gewährleisten. Ein paar Gründe dafür seien hier aufgezählt:

- MANET's zeichnen sich durch eine geringe Sendeleistung der einzelnen Knoten aus.

- Drahtlose Übertragung: Störung durch lokale oder atmosphärische Phänomene vorhanden.
- Batteriebetriebene Hardware welche sparsam mit den Ressourcen umgehen muss
- Billige und daher fehleranfälligere Hardware
- Böswillige Aktionen: Denial of Service Attacken, Störsender

Um die Verfügbarkeit zu erhöhen ist es also notwendig den Empfangsradius zu vergrössern, indem der angebotene Dienst entweder repliziert oder weitergeleitet wird. Weiterleitung oder Replikation von Daten muss nicht zwingend vom Dienstanbieter selber gewährleistet werden. Vielmehr ist es wünschenswert, dass jeder Host, welcher zur Verbesserung der Verfügbarkeit eines beliebigen Dienstes beitragen kann, seine Ressourcen auch freiwillig für andere zur Verfügung stellt. Solche Plattformen werden von den zahlreichen MANET Routingprotokollen bereitgestellt.

2.1.2 Confidentiality

Vertraulichkeit stellt sicher, dass Informationen gegenüber nicht authorisierten Drittpersonen abgesichert werden. Vertraulichkeit kann durch diverse kryptographische Methoden erreicht werden, welche auf einem höheren Layer operieren und deshalb nicht speziell an MANET's angepasst werden müssen. Beispiele dafür sind: SSH, SSL, GPG u.v.a.

Vertraulichkeit auf IP-Ebene kann ebenfalls sehr wichtig sein, so entlocken schon blosse Routing-Informationen dem Netzwerk Vertrauliche Informationen. Statistiken über: "Wer spricht mit wem? Wann? Wie oft?" können für einen Angreifer sehr Wertvoll sein und sollten demnach ebenfalls vertraulich behandelt werden. Dies kann z.B. durch kryptographisch gesicherte Tunnel zwischen Router gewährleistet werden. Verfügbare Plattformen wie z.B. IP-Sec müssen jedoch an MANET's angepasst werden.

2.1.3 Integrity

Integrität garantiert dass Informationen unverändert übertragen werden. Dies ist speziell für Router problematisch: Ein Datenpaket, welches ohne zusätzliche Massnahmen (d.h. ohne kryptographische Signaturen) über mehrere Hops geschickt wird, kann von einem fehlerbehafteten oder gar bösartigen System kompromittiert werden. So ist es als Router immer möglich, eine Man-in-the-Middle Attacke durchzuführen, ohne dass sowohl Quell- als auch Zielhost etwas davon merken. Für MANET's ist dies besonders problematisch, da jeder Host automatisch auch ein Router ist.

2.1.4 Authentication

Authentisierung spielt sowohl auf höheren wie auch auf niedrigeren Layer eine zentrale Rolle für die Sicherheit eines Netzes. Ohne Authentisierung kann sich jeder für jemand anderes ausgeben und in diesem Rahmen zahlreiche Attacken durchführen.

Einerseits ist es natürlich wünschenswert, möglichst jeden einzelnen Mitspieler im Netzwerk zu authentisieren, andererseits bringt dies aber auch einen grossen zusätzlichen Aufwand mit sich: Schlüssel müssen auf sicheren Wegen vorab ausgetauscht und signiert werden, ein Aufwand welcher beileibe nicht jeder Benutzer bereit ist auf sich zu nehmen. Allerdings bestehen schon ausgeklügelte Verfahren dazu (SSL, Kerberos, GPG u.v.a.). Viele Sicherheitsrelevanten Probleme lassen sich ohne vorhergehende Authentisierung nicht lösen.

2.1.5 Non-repudiation

Nachweisbarkeit stellt sicher dass der Ursprung einer Meldung nicht verleugnet werden kann. Sie spielt vor allem dann in Netzwerken eine grosse Rolle, wenn es um die Erkennung und Isolation von fehlerhaften oder bösartigen Hosts geht, z.B. wenn ein Punktesystem für die Qualität oder das Vertrauen einzelner Hosts aufgebaut werden soll. So kann man dem Übeltäter seine Schuld nachweisbar anlasten und entsprechende Gegenmassnahmen einleiten.

2.2 Sicherheitsmodell

Zur Erhöhung der Verfügbarkeit ist es wünschenswert, jedem Knoten im Ad Hoc Netzwerk eine *Quality of Service* (QoS) zuordnen zu können, welche Informationen für möglichst alle Verfügbarkeits-Aspekte eines MANET's bereitstellt:

- Auslastung des Systems
- Stationarität
- Batteriestatus

Die Einführung von QoS-Werten für alle Hosts bietet die Möglichkeit, die Routen nicht nur nach der geringsten Anzahl Hops (*Minimum Hops*) zum Ziel zu ermitteln, sondern auch nach der *maximalen QoS*.

Allerdings ergeben sich in Hinblick auf bösartigen oder nicht korrekt funktionnierenden Hosts auch neue Schwachstellen: so kann ein Angreifer seine eigene QoS (oder auch die anderer Hosts) beliebig modifizieren, wenn er das zugrundeliegende Protokoll nicht einhält. Aus diesem Grund ist es

notwendig, denjenigen Host, welcher Informationen über QoS bereitstellt, zu authentisieren. Aus der Authentisierung können dann Werte für *Vertrauen* (Trust) dieses Hosts ermittelt werden. Vertrauen ist hier in einem weiteren Sinne zu verstehen: "Vertrauen auf dass der Host korrekte QoS-Werte übermittelt"

Ziel dieser Arbeit ist also die Bereitstellung einer Infrastruktur, welche möglichst unabhängig von den verschiedenen Protokollen eine Gesamtsicht von *Vertrauen* und *QoS* über das bekannte Netzwerk bietet (Abb. 2.1). Diese Daten sollen dann dem Routing-Protokoll zugänglich gemacht werden.

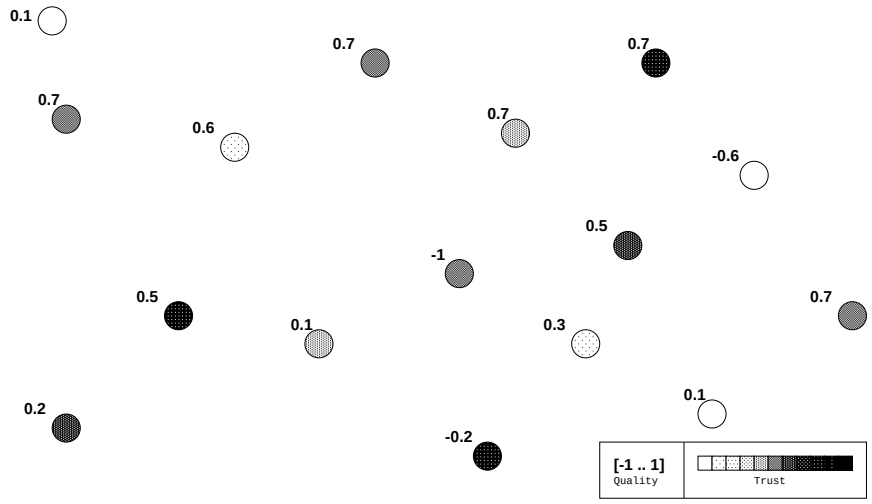


Abbildung 2.1: Sicherheitsmodell, basierend auf *Vertrauen* and *Qualität*

Wichtig hierbei ist, dass die QoS eines Knotens für alle Mitspieler gleich aussehen sollte. Nicht aber das Vertrauen. Jeder sollte ein nach seinen Bedürfnissen gestaltetes Vertrauensmodell aufbauen können, d.h. jeder sollte selber entscheiden können, wem er nun vertraut oder wem nicht.

Diese Infrastruktur sollte ein Kompromiss aus folgenden Aspekten sein:

- maximale Skalierbarkeit
- maximale Genauigkeit
- minimaler Overhead
- schnelle Reaktionszeit auf Veränderungen der bereitgestellten Werte
- Unabhängigkeit vom darunterliegenden Routing-Protokoll
- Unabhängigkeit von der Authentisierungsmethode

Kapitel 3

MANET Security: State of the Art

3.1 MANET Routingprotokolle

Mobile Ad Hoc Netzwerke (MANET's) sind ein offenes Forschungsthema [1]. Viele Vorschläge für Routing-Protokolle sind gemacht worden, welche je nach Szenario besser oder schlechter geeignet sind. Bis dato ist noch keines davon zum Standard auserkoren worden. Die wichtigsten werden hier kurz aufgelistet.

3.1.1 Proaktive Protokolle

Proaktive Routingprotokolle (auch Table-Driven genannt) zeichnen sich dadurch aus, dass fortlaufend Routen zu allen anderen Knoten unterhalten werden. Das führt einerseits zu einem höheren Signalisierungsaufwand, hat aber zum Vorteil, dass neue Verbindungen schnell etabliert werden können.

Analog zu stationären Netzen wird auch hier zwischen *Link State* und *Distance Vector* Protokollen unterschieden:

Distance Vector

- Highly Dynamic Destination-Sequenced Distance Vector Routing (DSDV) [2, 28]

Link State

- Optimized Link State Routing Protocol (OLSR) [21]
- Topology Broadcast based on Reverse-Path Forwarding (TBRPF) [22]
- Fisheye State Routing Protocol (FSR) [24]

3.1.2 Reaktive Protokolle

Reaktive Routingprotokolle (auch On-Demand-Driven genannt) unterhalten im Gegensatz zu proaktiven Systemen nicht fortlaufend Routen zu anderen Knoten, vielmehr werden diese erst bei Bedarf erkundet. Dies impliziert eine zusätzliche Zeitverzögerung beim Aufbau einer neuen Route. Der Signalisierungsaufwand ist allerdings geringer[1], jedoch stark abhängig vom Verkehrsaufkommen.

Bekannte reaktive Protokolle:

- Dynamic Source Routing Protocol (DSR) [2, 18]
- Ad Hoc On Demand Distance Vector (AODV) [2, 20]
- Temporally Ordered Routing Algorithm (TORA) [2]

3.1.3 Hybride Protokolle

Hybride Routingprotokolle sind Mischungen aus proaktiven und reaktiven Protokollen. Hier wird versucht, die Vorteile aus beiden Kategorien zu nutzen.

Bekannte hybride Protokolle:

- Zone Routing Protocol (ZRP) [2, 19]

3.1.4 Hierarchische Protokolle

Hierarchische Routingprotokolle sind vor allem für grössere Ad Hoc Netzwerke gedacht.

Bekannte hierarchische Protokolle:

- Landmark Routing Protocol (LANMAR) [23]

3.2 Attacken

Wie auch in Kabelgebundenen Netzen gibt es für Drahtlose Netze diverse mögliche Attacken, um die Sicherheit zu untergraben. Einige für diese Arbeit besonders relevante sind hier kurz beschrieben.

[1] Dies ist besonders für Batteriebetriebene Hosts wichtig

3.2.1 Impersonification

Prinzipiell hat jeder Router im Netz die Möglichkeit, die Identität eines anderen Teilnehmers vorzugaukeln, falls sich dieser nicht korrekt authentisieren kann. Zur Abwehr gibt es diverse Möglichkeiten der Kryptologie, welche auch für Ad Hoc Netzwerke Gültigkeit haben (SSH, SSL, GnuPG u.v.a.).

3.2.2 Multiple Identities

Jeder Host in einem Netzwerk hat prinzipiell die Möglichkeit, seine Identität (IP/MAC-Adresse, Private Key) zu wechseln, oder simultan mithilfe von *virtual interfaces* mehrere Identitäten vorzutäuschen. Dies hat weitgehende Konsequenzen:

- Ein als bösartig erkannter Host kann sofort reagieren und seine nächste Identität annehmen, welche ihm für seine Attacken wieder freie Bahn lässt.
- Jegliche Sicherheit im Netz, welche darauf basiert, dass die Anzahl bösartiger Hosts nur eine gewissen Prozentsatz ausmacht, kann damit unterminiert werden. Eine hinreichende Anzahl Virtual Hosts, welche diesen Prozentsatz übersteigt, kann trivialerweise diese Sicherheit zunichte machen. So kann unter anderem auch *Threshold Cryptography* [11] untergraben werden.

Das Erstellen von virtual interfaces kann nicht verhindert werden. Jeder im Netz mit einem dazu fähigen Netzwerkadapter kann dies ungehindert tun. Die Aufspürung von virtual interfaces stellt sich als sehr schwierig heraus: der einzige Ansatzpunkt für eine Aufspürung ist derselbe Standort des Senders, welcher vielleicht anhand von Übermittlungsdauer oder anderen physikalischen Eigenschaften ermittelt werden kann.

Die Verhinderung von Multiple Identities ist im Allgemeinen ein sehr komplexes Gebiet und wird in der Kryptologie heiss diskutiert. Für MANET's anwendbare Lösungen sind mir nicht bekannt.

3.2.3 Denial of Service

Denial of Service (DoS) Attacken sind immer sehr schwierig abzuwehren. Sie richten zwar keinen direkten Schaden an, können aber in Kombination mit anderen Attacken durchaus sehr wirkungsvoll sein. Am schlimmsten sind DoS-Attacken dann, wenn einzelne Hosts gezielt ausgeschaltet werden können.

MANET Routing Protokolle sind äusserst anfällig auf diese Art von Attacken. So können Routen von einem einzelnen Angreifer beinahe beliebig modifiziert werden, so dass einzelne Hosts nicht mehr erreicht werden können.

Kapitel 4

Modelle für sicheres Routing

4.1 Quality of Service

In diesem Abschnitt wird eine Möglichkeit aufgezeigt, wie QoS in MANET's unabhängig vom darunterliegenden Routingprotokoll beschrieben und ermittelt werden kann.

4.1.1 Zertifizierte QoS

Zur Sicherstellung einer gewissen QoS kann für einen Host ein Zertifikat ausgestellt werden, welches pauschal folgende Aspekte zertifiziert:

- Verfügbare Bandbreite
- Bereitschaft zum Routen
- Zuverlässigkeit beim Routen

Diese Zertifikate könnten beispielsweise von einer unabhängigen Prüfstelle ausgestellt werden. Solche Zertifikate müssen an die verwendete Hardware gebunden sein.

4.1.2 Signierte QoS

Anstatt ein Zertifikat für die QoS zu besitzen (Kap 4.1.1) kann ein Host eine ermittelte QoS auch selbst signieren. Geniesst er genügend Vertrauen, so werden die anderen Hosts diesen Wert akzeptieren.

4.1.3 Ermittlung von QoS

Voraussetzungen

- Jeder Knoten kann QoS seiner Nachbarn ermitteln (z.B. durch Beobachten seines Verhaltens über die Zeit gemittelt).

- Jeder Knoten besitzt (optional) ein Zertifikat für eigene QoS, welches im Voraus von einer CA ausgestellt wurde.

- Das Routingprotokoll kennt Mechanismen, um eine alternative Route zu finden.

Vorgehensweise

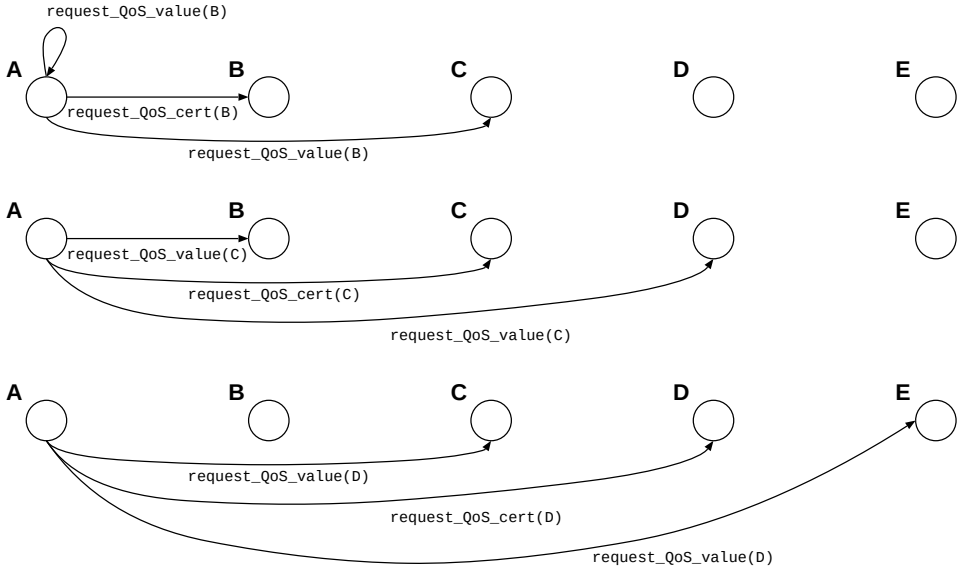


Abbildung 4.1: QoS propagation

Nachdem das Routing Protokoll eine Route ermittelt hat, wird die Qualität der Route wie folgt berechnet:

```
l = # hops to destination
host[l+1] = array of host_adr

for(i=1; i < l; i++) {
  v1 = request_QoS_value(host[i-1], host[i]);
  v2 = request_QoS_value(host[i+1], host[i]);
  c = request_QoS_cert(host[i]);
  if(v1 or v2 or c is not acceptable) {
    request_new_route();
    exit;
  }
}
```

Vorteile

- unabhängig vom Routing-Protokoll (ausser Voraussetzungen)
- gegenseitige Kontrolle
- Source-Host hat Kontrolle über QoS der gesamten Route

Nachteile

- vorhergehende Zertifizierung nötig (falls angewendet)
- aufwändig, vor allem bei sich schnell verändernder Topologie

4.2 Vertrauen (Trust)

In diesem Abschnitt werden Methoden gezeigt, wie einzelne Knoten sich ausweisen können, damit ein Wertesystem für *Vertrauen* aufgebaut werden kann.

4.2.1 Router Zertifikate

Um sicherzustellen dass nur diejenigen Hosts als Router benutzt werden, welche sich in irgendeiner Form kryptographisch ausweisen können, müssen die Routing-Algorithmen zwangläufig verändert werden. Für einen Host, der ein Paket von einem Router erhält, muss entscheidbar sein, ob dieses Paket authentisch ist, also vom betreffenden Router mit einem bekannten und gültigen Schlüssel signiert worden ist (siehe Abb. 4.2). Dies hat weitgehende Implikationen:

- Schlüssel müssen ausgetauscht werden, bevor sicheres Routing stattfinden kann
- Jedes Paket muss von jedem einzelnen Router signiert werden (Performanceeinbussen, Overhead)
- Die Signaturen müssen entweder von jedem einzelnen Router (Hop-by-Hop, was wiederum das Vertrauen in die Route vermindert) oder vom Zielhost gesamthaft überprüft werden.

X.509

Als weit verbreiteter Standard für Zertikikate bietet sich X.509 [9] als Basisplattform an, obschon dieser für solche Aufgaben etwas überdimensioniert ist, was wiederum einen Overhead mit sich bringt.

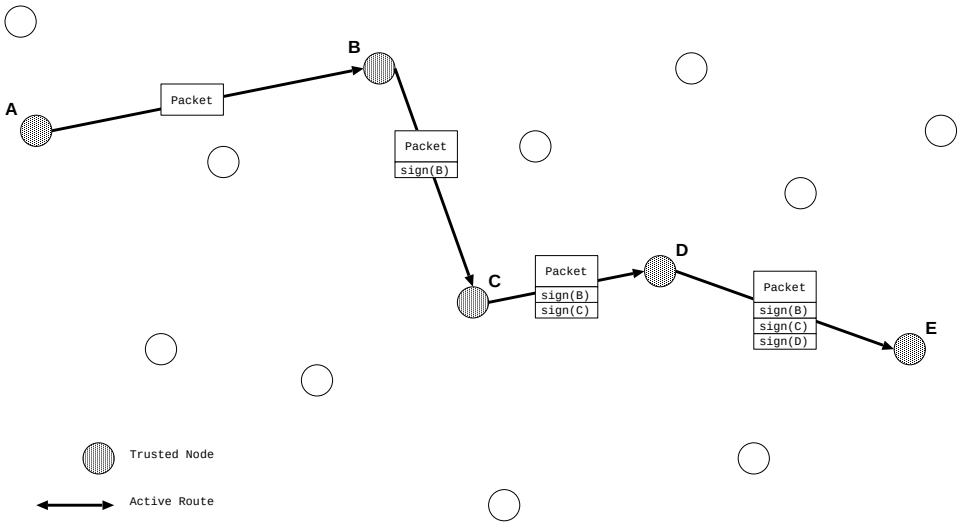


Abbildung 4.2: Routing Zertifikate

GNU Privacy Guard (GnuPG)

GnuPG [7] kann in diesem Rahmen durchaus auch als ernstzunehmender Kandidat für eine Zertifikatsplattform dienen. Es bietet die Vorteile eines *Web of Trust* welches mehrere Abstufungen für Vertrauenswerte kennt. GnuPG unterstützt alle wichtigen Verschlüsselungsverfahren und bietet zudem einfache Schnittstellen sowohl für das Key-Management als auch für die Ver- und Entschlüsselung.

4.3 Network Sniffer

Eine mögliche Massnahme zur Steigerung der Sicherheit ist die Einführung von *Network Sniffer*, welche die aktiven Router überwachen (s. Abb. 4.4). Die Idee ist, dass fehlerbehaftete oder gar böswillige Router sofort als solche erkannt werden können, sobald Pakete nicht so weitergegeben werden wie dies vom Protokoll vorgegeben wird. Eine solche Überwachung ist also stark vom darunterliegenden Routingprotokoll abhängig.

Voraussetzungen

- Der Sniffer muss die eingehenden sowie die verschickten Pakete des zu überwachenden Routers sehen können.
- Der Sniffer muss wissen, zu welchem Node die verschickten Pakete gemäss Protokoll gehen müssen (d.h. er muss die Routingtabellen kennen)

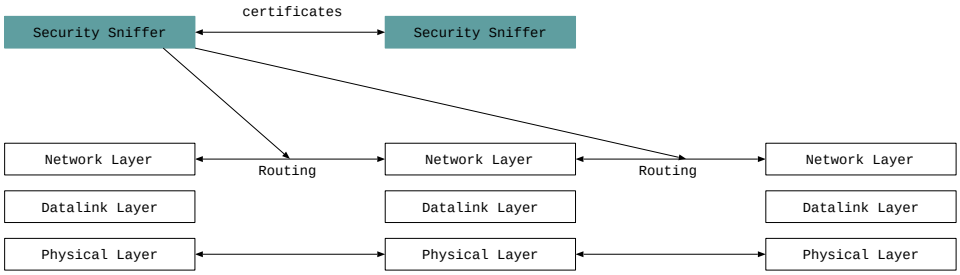


Abbildung 4.3: Network Sniffer: Layer-Modell

Der Network Sniffer erkennt

- Verzögerungen beim weiterleiten von Paketen (QoS)
- nicht weitergeleitete Pakete (Überlast, Fehlferhalten oder DoS-Attacken)
- an falsche Destination gesendete Pakete (Fehlverhalten oder DoS-Attacken)
- veränderte Pakete (Fehlverhalten, Man-in-the-Middle Attacken)

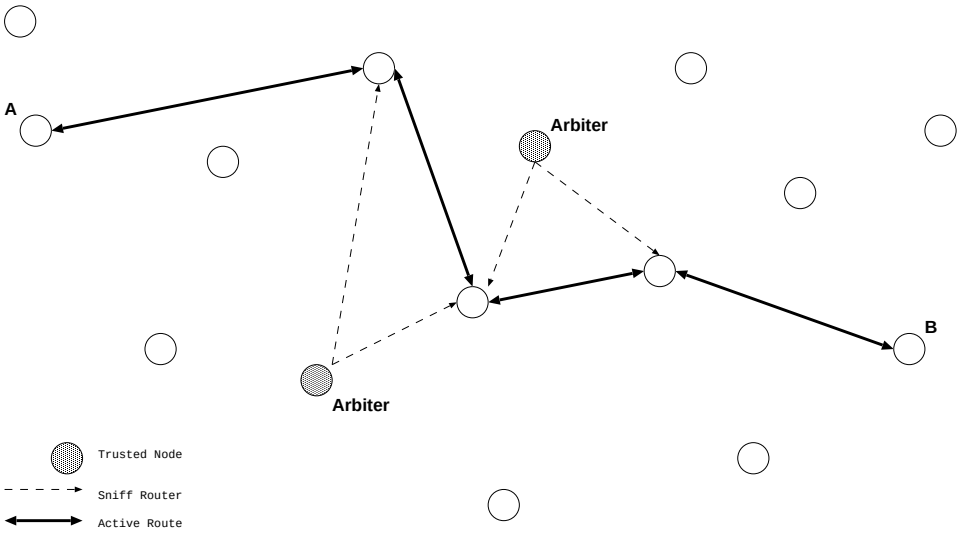


Abbildung 4.4: MANET Network Sniffer

4.3.1 Next-Hop Network Sniffer

Eine einfachere Möglichkeit einen Router zu überwachen geht vom vorangehenden Host aus. Dieser kann, vorausgesetzt es besteht eine Zweiwegkommunikation, den nächsten Hop überwachen. Die Sicherheit des Netzes erhöht

sich automatisch, da ein fehlerhaftes Verhalten des nächsten Hops sofort erkannt werden kann und dieser gegebenenfalls in Zukunft aus dem Routing ausgeschlossen werden kann.

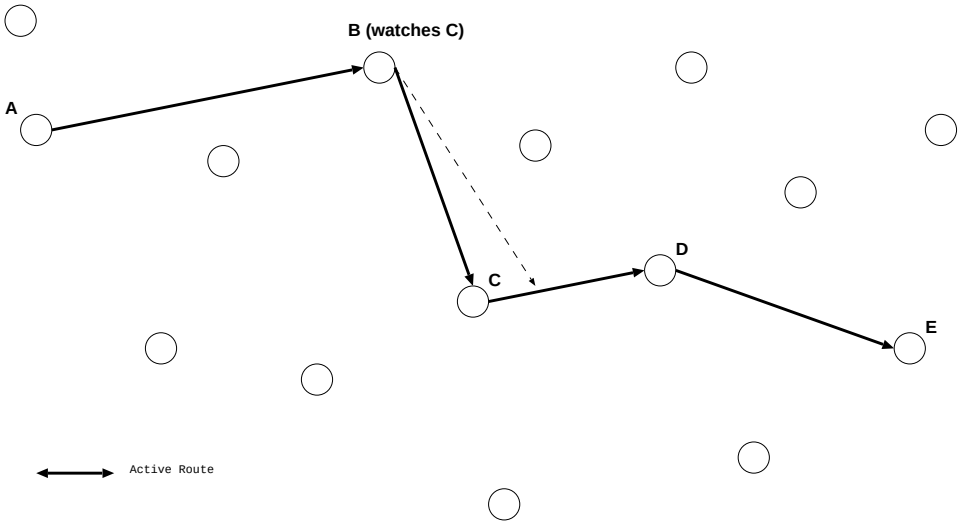


Abbildung 4.5: Next-Hop Network Sniffer B überprüft von C gesendete Pakete

Voraussetzungen

Es gelten die gleichen Voraussetzungen wie beim einfachen Network Sniffer (Kap. 4.3).

Verwendungszweck

Ein solches Modell kann sowohl bei proaktiven wie auch bei reaktiven Routingprotokollen (siehe Kap. 3.1) zur Etablierung einer QoS-Infrastruktur Verwendung finden.

4.3.2 Trusted Network Sniffer

Um die erkannten Tatsachen weitergeben zu können, muss der Sniffer im Netzwerk dafür genügend Vertrauen geniessen. Ein Dritter kann nicht wissen, ob der Sniffer korrekte Werte übermittelt oder ob es sich um einen bösartigen Host handelt.

Eine mögliche Massnahme, um Übeltäter auf dem Netz zu entlarven besteht nun darin, dass der Sniffer beim Erkennen eines Fehlers sofort Alarm

schlägt. Er sendet dann per Broadcast oder Multicast eine Warnmeldung, dass der Übeltäter sofort aus allen Routing-Tabellen zu entfernen ist[1].

Voraussetzungen

Es gelten die gleichen Voraussetzungen wie beim einfachen Network Sniffer (Kap. 4.3), und zusätzlich:

- der Trusted Network Sniffer verfügt über ein gültiges Zertifikat
- Broadcast (z.B. Flooding) oder Multicast möglich

Probleme

- Hat ein bösartiger Host ein Zertifikat eines Trusted Network Sniffers korrumpiert, so kann er gezielt einzelne gutartige Hosts als Übeltäter markieren und so Teile des Netzes lahmlegen.
- Der Übeltäter kann die Warnmeldung auch sehen und wird als Reaktion darauf seine Identität ändern (siehe dazu auch Kap. 3.2.2).

[1] Broadcast oder Multicast darf natürlich nicht scheitern, wenn der überführte Übeltäter nicht mitmacht!

Kapitel 5

Design

Während im Kapitel 4 (Modelle für sicheres Routing) einige generische Ansätze aufgezeigt wurden, wird hier ein davon abgeleitete Design für sicheres, QoS/Trust basiertes Routing (des weiteren *Secure-Routing* genannt) besprochen.

5.1 Anforderungen

- Jeder Host soll die Route zu einem Zielhost aus mehreren angebotenen selber auswählen können. Dies impliziert ein *Source-Routing* Verfahren.

- Die Wahl der Route soll nicht nur von der Anzahl Hops, sondern auch von QoS/Trust-Werten abhängig sein.

5.2 Kryptologie

5.2.1 Key Management

Jeder Host im Netzwerk muss einen oder mehrere private Schlüssel besitzen, mit welchen er seine ermittelten QoS-Werte signieren kann. Die zugehörigen öffentlichen Schlüssel muss er vorab mit den anderen Hosts auf sicherem Weg ausgetauscht haben, damit eine Authentisierung und Ermittlung eines Trust-Wertes überhaupt möglich wird. Ansonsten wird er bei allen anderen einen Trust-Wert von 0 (*untrusted*) erhalten.

Dazu ist eine *Public Key Infrastructure* notwendig, welche es einem Benutzer mit möglichst einfachen Mitteln ermöglicht, Schlüssel zu generieren, signieren und auszutauschen.

5.2.2 QoS Werte

QoS Werte sind möglichst genaue Informationen über Auslastung, Stationarität und gegebenfalls Batteriestatus. Jeder Host muss seine QoS selber ermitteln.

5.2.3 Trust Werte

Die zugrundeliegende *Public Key Infrastructure* sollte nicht nur zwischen *vollem Vertrauen* und *keinem Vertrauen* unterscheiden können. Vielmehr sollte sie eine möglichst feine Abstufung von Vertrauen erlauben.

Die von der *Public Key Infrastructure* gelieferten Werte werden dann auf eine Skala von 0 (kein Vertrauen) bis 255 (ultimatives Vertrauen) skaliert.

5.2.4 Nonce

Ein Angreifer, welcher alle von seinem Opfer signierten QoS-Werte sammelt, kann diese zu einem späteren Zeitpunkt wieder ins Netz einspeisen. So wird es ihm z.B. möglich, einen gesammelten schlechten QoS-Wert erneut ins Netz einzuspeisen und so die anderen davon überzeugen, dass dieser Wert vom Opfer kommt (vorausgesetzt die Signatur ist noch immer gültig).

Um diesen *Replay-Attacken* vorzubeugen ist der Einsatz einer *Nonce*, also eines Zufallswertes notwendig. Ein Host, welcher eine Anforderung für einen QoS-Wert abschickt, wählt eine Nonce und fügt sie dieser Anforderung an. Der Empfänger wird seinen QoS-Wert zusammen mit der erhaltenen Nonce signieren. Für den Angreifer werden so seine gesammelten Pakete unnütz, da es sehr unwahrscheinlich ist, dass in absehbarer Zeit wieder dieselbe Nonce verwendet wird.

5.3 QoS Intervall

Die QoS/Trust-Werte sollen nicht ewig gültig sein, vielmehr soll jeder Host diese in regelmässigen Abständen neu anfordern. Wie gross dieser Zeitraum sein sollte muss noch genauer evaluiert werden, sicherlich jedoch irgendwo zwischen einigen Sekunden bis einigen Minuten.

5.4 Secure Routing Modell

Jeder Host führt in seinen Routing-Tabellen für alle ihm bekannten Host die folgenden Attribute mit:

1. IP-Adresse
2. Key-ID

3. QoS Wert [0..255] (initial 0)

4. Trust Wert [0..255] (initial 0)

Ein Schlüssel (also die Key-ID) ist immer an eine IP-Adresse gebunden. Dies ist deshalb wichtig, da sonst ein Angreifer mit gefälschter IP-Adresse mit seinem Schlüssel (dem kein Vertrauen geschenkt wird) gezielt den Trust-Wert für diese Adresse auf Null herabsetzen könnte, indem er einfach einen QoS-Wert, die er mit seinem Schlüssel korrekt unterschreibt, zum Opfer sendet.

Eine neue Key-ID für die gleiche IP-Adresse wird aus demselben Grund nur dann geandert, falls sich mit dem neuen Schlüssel auch der Trust-Wert verbessert.

Kapitel 6

Implementation

In diesem Kapitel wird die Implementation des *Secure-DSR* genauer erläutert. Es erfüllt die im Kapitel 5 (Design) modellierten Spezifikationen für *Secure-Routing*.

Es wurde entschieden, ein Kernel Modul zu Erstellen, welchem das DSR-Protokoll zugrunde liegt und zusätzlich die Möglichkeit bietet, die Source-Route anhand von ermittelten QoS/Trust-Werten zu ermitteln.

6.1 Übersicht

Folgende Schritte wurden unternommen:

1. Festlegung auf Implementierung (nicht Simulation) eines Secure-Routing Modells

2. Festlegung auf das DSR-Protokoll: Source-Routing Protokoll, unterstützt multiple Routen, Kandidat für MANET-Standard

3. Erweiterung des DSR Internet-Draft [18]

4. Erweiterung des DSR Kernel Modules von Alex Song [12]

5. Erstellung eines Interfaces für die Kommunikation zwischen Kernel-Module und Userland Crypto-Engines

6. Erstellung eines GnuPG [7] Plugins für Krypto-Interface

Es wurde viel Wert darauf gelegt, das Design möglichst simpel zu halten, um zukünftige Änderungen einfacher zu gestalten.

6.2 Dynamic Source Routing (DSR)

MANET Routingprotokolle sind ständigen Veränderungen unterworfen, so auch das DSR Protokoll, auf welchem diese Arbeit basiert. Alles hier beschriebene bezieht sich auf das Internet-Draft vom 21. Februar 2002 [18].

Alle Erweiterungen zum Protokoll sind so ausgelegt, dass sie dieses Draft nicht verletzen, das bedeutet insbesondere dass ein Netzwerk basierend auf dem bestehenden DSR Protokoll *ohne* Secure-DSR reibungslos mit einzelnen Hosts *mit* Secure-DSR interagieren kann. Das DSR Protokoll ist so ausgelegt, dass es unbekannte DSR-Options (siehe Kap. 6.2.2) in Datenpaketen ignoriert, insbesondere auch die hier verwendeten Secure-DSR-Options.

6.2.1 Ermittlung von QoS und Trust

QoS Werte werden nur von Hosts angefordert, welche für die aktuell benötigte Route zu einem Ziel-Host auch in Betracht gezogen werden müssen.

Dies gilt für die folgenden Situationen:

- Empfang einer Route Reply Option
- Senden einer Source Route Option

Dabei wird zuerst geprüft, ob noch aktuelle[1] QoS/Trust-Werte für den betreffend Host gespeichert sind. Falls die Werte noch gültig sind, wird dieser Host nicht angefragt.

Asynchronität

Die Ermittlung von QoS/Trust geschieht Asynchron, d.h. es wird mit dem Routing nicht abgewartet, bis alle Hosts ihre Werte übermittelt haben. Dies wäre zwar wünschenswert, da aber die Rechenzeit sowohl für die Erstellung als auch für die Überprüfung von Signaturen erheblich ist, würde jeder Verbindungsaufbau massiv verzögert werden.

6.2.2 Secure-DSR-Options

Hier werden nur die zusätzlichen, für das Secure-DSR hinzugefügte Options aufgelistet. Eine Übersicht aller übrigen DSR-Options ist im Internet-Draft [18] aufgeführt.

DSR-Datenpaket

Ein DSR-Datenpaket besteht aus folgenden Teilen:

[1] QoS Werte sind nur für einen bestimmten Zeitraum gültig (siehe Kap. 5.3)

1. IP-Header
2. DSR-Header
3. eine oder mehrere DSR-Options
4. Payload oder Next-Header

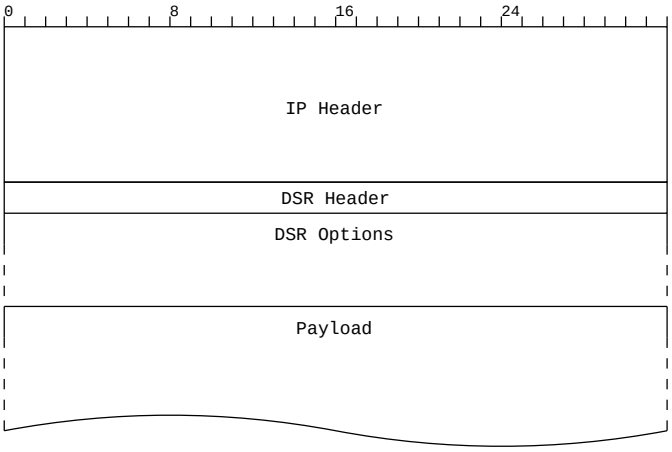


Abbildung 6.1: DSR Datenpaket mit Payload

QoS Request Option

QoS Request Options werden immer dann versendet, wenn die QoS/Trust Werte derjenigen Hosts, welche für das Source-Routing zu einem Zielhost in Frage kommen, nicht mehr aktuell sind (siehe Kap.5.3).

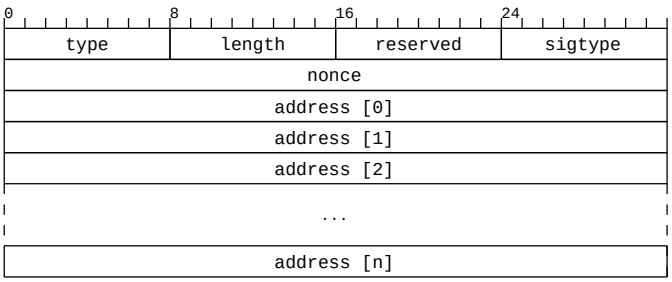


Abbildung 6.2: QoS Request Option

Eine QoS Request Option besteht aus folgenden Datenfeldern (siehe auch Abb. 6.2):

- `Option Type`: 17
- `Option Data Len`: 8-bit unsigned integer. Länge der Option, in Bytes, ausgenommen `Option Type` und `Option Data Len` Felder.
- `Reserved`: Reserviert für künftige Erweiterungen. Muss als 0 gesendet werden und beim Empfang ignoriert werden.
- `Signature Type`: 8-bit unsigned integer. Typ der angeforderten Signatur. Erlaubte Werte sind: `NONE=0, SSL=1, GPG=2`.
- `Nonce`: 32-bit Zufallswert, muss für QoS Reply benutzt werden, um Replay-Attacken zu verhindern.
- `Address[1..n]`: Liste der Zieladressen. Jeder hier aufgeführte Host muss mit einer QoS Relpy Option antworten.

QoS Reply Option

QoS Reply Options werden als Antwort auf Qos Request Options versendet.

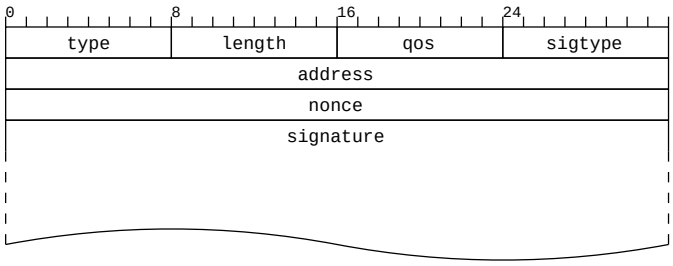


Abbildung 6.3: QoS Reply Option

Eine QoS Reply Option besteht aus folgenden Datenfeldern (siehe auch Abb. 6.3):

- `Option Type`: 16
- `Option Data Len`: 8-bit unsigned integer. Länge der Option, in Bytes, ausgenommen `Option Type` und `Option Data Len` Felder.
- `QoS Value`: 8-bit unsigned integer. Gibt den aktuellen QoS Wert dieses Host an.

- `Signature Type:` 8-bit unsigned integer. Typ der benutzten Signatur. `NONE=0, SSL=1, GPG=2`
- `Address:` IP-Adresse des Hosts.
- `Nonce:` Von der QoS Request Option übernommener Wert.
- `Signature:` Signatur (variabler Länge) von den vorangehenden Feldern (inklusive `Option Type` and `Option Data Len`). Die Länge der Signatur beträgt `Option Data Len - 12`.

Beim Empfang einer QoS Reply Option wird zuerst die Signatur geprüft und der zugehörige Trust-Wert ermittelt. Danach wird folgendermassen vorgegangen:

1. Hat die Key-ID geändert und der Trust-Wert sich nicht erhöht, wird die QoS Reply Option ignoriert.
2. Hat die Key-ID geändert und der Trust-Wert sich erhöht, wird die dem Host zugehörige Key-ID aktualisiert.
3. Ist die Key-ID korrekt, so wird der neue QoS-Wert und der ermittelte Trust-Wert übernommen.

6.3 Wahl der Basisplattform

Dieser Abschnitt erklärt die grundlegenden Design-Entscheide, welche zur Erstellung des Secure-DSR Programmpaketes gefällt wurden.

6.3.1 Betriebssystem

Für die Wahl des Betriebssytems sind zwei Punkte wichtig:

1. DSR Implementation im Quellcode mit akzeptablem Lizenzmodell verfügbar
2. Erweiterung möglichst einfach

Name	**OS**	**Quellcode**	**Lizenz**
Rice Monarch Project [13]	FreeBSD 3.3	ja	frei
Piconet II [12]	Linux 2.4	ja	GPL

Tabelle 6.1: DSR Implementationen

Die zwei zur Zeit verfügbaren DSR-Implementationen sind in der Tabelle 6.1 zusammengestellt. Die Entscheidung fiel auf Piconet II, einerseits weil ich mich bereits gut mit Linux auskenne und damit die Erweiterung des Codes bedeutend vereinfacht wird, andererseits weil die FreeBSD Implementation als pre-alpha release deklariert ist.

Windows

Microsoft Windows ist sowohl auf Desktop PC's wie auch auf PDA's (Microsoft Pocket PC) das meistverbreitetste Betriebssystem, daher wäre es durchaus wünschenswert, Secure-Routing auch darauf zu implementieren. Trotzdem wurde dieses Betriebssystem im Rahmen dieser Arbeit nicht berücksichtigt, da für eine Windows-Implementation der gesamte DSR-Programmcode neu geschrieben werden müsste und sich Linux aufgrund dessen freien Verfügbarkeit und offenem Quellcode für wissenschaftliche Arbeiten viel besser eignet.

6.3.2 Piconet II

Piconet II [12], eine Implementation des DSR-Protokolls für Linux 2.4, wurde als Basis für die vorgeschlagenen Secure-Routing Erweiterungen benutzt. Es handelt sich um ein Kernel-Modul, welches sowohl auf Intel x86 als auch auf ARM Architekturen lauffähig ist. Letzteres ermöglicht es, das Modul auch auf ARM basierten PDA's zu benutzen.

Piconet II erfüllt fast alle DSR Spezifikationen, Probleme wurden lediglich bei Verwendung von Übertragungsmedien mit unidirektionalen Verbindungen[2] gefunden.

Die Implementation ist relativ sauber, aber schlecht (beinahe gar nicht) dokumentiert.

6.4 Programmaufbau

Der Programmaufbau wird hier nur in groben Zügen erklärt, um den Einstieg in den Source Code zu erleichtern. Details können dem gut dokumentierten Code entnommen werden.

6.4.1 Routing Tabellen

Das DSR Internet-Draft [18] berücksichtigt mehrere Möglichkeiten, die gefundenen Routen zu speichern.

Route Cache

Die einfachste und auch effizienteste Variante dient ein Route Cache, d.h. Routen werden in einer Hash-Tabelle der Form: [Destination, Hops to Destination] gespeichert, und zwar werden die Routen aus Datenpaketen mit Routing Informationen, wie z.B. einer Route Reply Option, extrahiert.

[2] genauer: die Vermeidung von Loops bei Route Discovery entsprechen nicht dem DSR-Draft

Piconet II [12] verwendet einen solchen Route Cache (allerdings sehr ineffizient, es fehlt unter anderem ein Hash-Algorithmus zur schnelleren Suche nach Hosts).

Link Cache

Eine weitere Variante ist die Verwendung eines Link Caches, bei welchem ein Graph mit allen bekannten Links aufgebaut wird. Die beste Route wird hier mit dem *Shortest Path First* Algorithmus von Dijkstra ermittelt, was geringe Performance-Einbussen mit sich bringt, andererseits aber eine vollständigere Sicht der bekannten Verbindungen ermöglicht. Dies ist insbesondere bei Secure-DSR für die Berücksichtigung von QoS/Trust-Werten notwendig, da die beste Route nicht unbedingt diejenige mit den wenigsten Hops sein muss.

Feldname	**Bedeutung**
Address	IP-Adresse des Knotens
Timestamp	Zeitpunkt der letzten Aktualisierung
QoS	Aktueller QoS-Wert
Trust	Aktueller Trust-Wert
Sig-Type	Signaturtyp (0=NONE, 1=GPG, 2=SSL)
Key-ID	Identifikation des Schlüssels
State.expire	Verfallsdatum des QoS und Trust Wertes
State.req_sent	1, falls QoS Request unterwegs (gesendet)
	0, falls kein QoS Request unterwegs
State.nonce	Verwendete Nonce (Zufallswert)

Tabelle 6.2: Attribute eines Knotens im Link Cache Graph

Ein solcher Link-Cache wurde implementiert (`graph.c, linklist.c, spf.c, pqueue.c`). Jeder Knoten ist eindeutig definiert durch seine IP-Adresse. Die Attribute eines Knotens sind in Tabelle 6.2 aufgelistet. Links sind unidirektional, ihre Metrik (bzw. Distanz) ist definiert durch eine Funktion von QoS und Trust des Quell- und Zielknotens (siehe Kap. 6.4.2).

6.4.2 Distanzfunktionen

Zur Ermittlung der kürzesten Route wurden vier Funktionen implementiert. Die Distanz der gesamten Route ergibt sich aus der Summe der Distanzen der einzelnen Links. Die Distanz eines Links ist ihrerseits eine Funktion aus den QoS- und Trust-Werten des Quell- und Zielnodes.

	$D(a,b)$	die Distanz des Links von Node a nach Node b,
	$Q(x)$	der QoS-Wert des Nodes x,
sei:	$T(x)$	der Trust-Wert des Nodes x
	α, β	Streckungsfaktoren (const.)
	γ	Zusätzliche Kosten für einen Hop (const.)

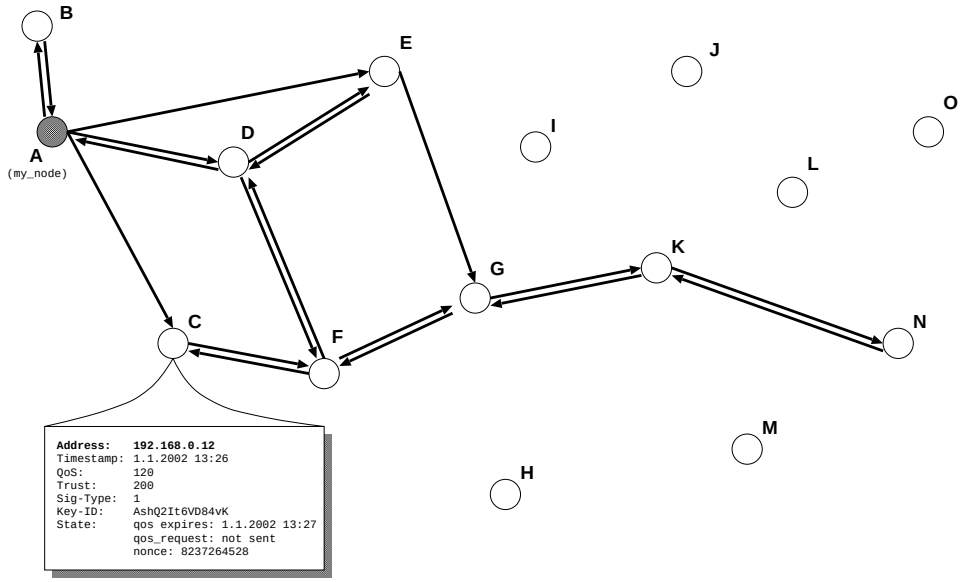


Abbildung 6.4: Beispiel eines Link Caches

Dann gelten folgende Distanzfunktionen:
Minimum Hops:

$$D(a,b) = 1$$

Maximum QoS:

$$D(a,b) = \frac{\alpha}{Q(b)}$$

Maximum Trust:

$$D(a,b) = \frac{\beta}{T(b)}$$

Kombiniert:

$$D(a,b) = \frac{\alpha}{Q(b)} + \frac{\beta}{T(b)} + \gamma$$

6.4.3 Krypto-Interface

Die vom Secure-DSR benötigten Kryptographischen Funktionen (signieren und Signaturen überprüfen) wurden als Userspace Programm Implementiert. Gründe Dafür sind:

- Public-Key-Cryptography Programmbibliotheken (`GPGME, libssl`) sind nicht als Kernel Patches erhältlich, eine Portierung wäre äusserst umständlich.
- Erweiterung um zusätzliche Public-Key Infrastrukturen durch Plugins möglich.

- Möglichkeit der Verwendung von Schlüsseln eines beliebigen Users im System.

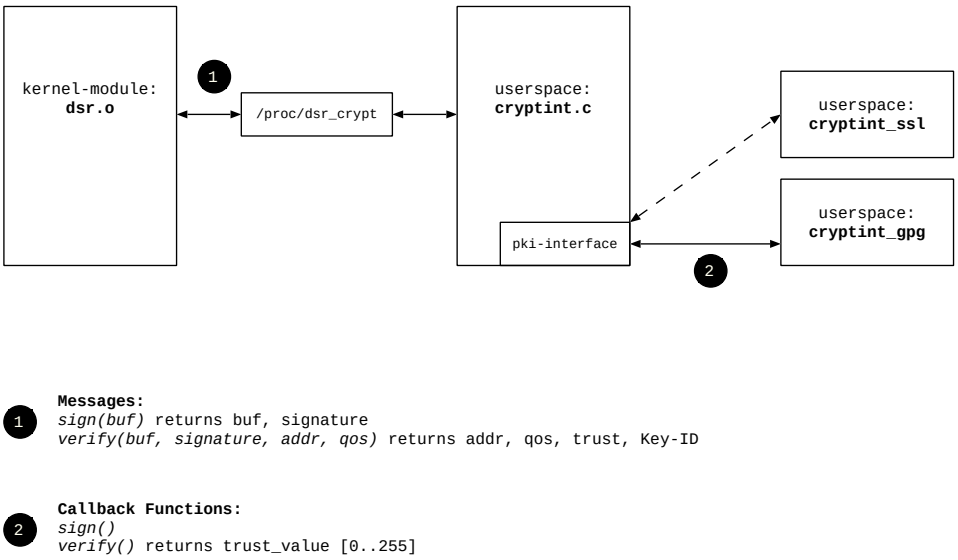


Abbildung 6.5: Verknüpfung der verschiedenen Codebausteinen

Aus diesem Grund wurde eine Library erstellt (`cryptint.c`, im folgenden Krypto-Interface genannt), welche für alle benötigten kryptographischen Funktionen verantwortlich ist. Diese Library macht nichts anderes, als auf Events vom Kernel-Modul zu warten und die zugehörigen Callback-Funktionen der Plugins aufzurufen. Diese Events werden vom Kernel-Modul über das proc-Filesystem ausgelöst (`/proc/dsr_crypt`, siehe Abb. 6.5). Das Krypto-Interface bearbeitet diese und schreibt die Resultate wieder zurück. Zwei Events sind definiert:

- Signieren von Daten
- Signatur überprüfen, daraus Trust-Wert ermitteln

Momentan ist nur ein GnuPG-Plugin implementiert (`cryptint_gpg.c`). Dieses Plugin ruft seinerseits Funktionen aus GPGME [8], einer Library für GnuPG Funktionen, auf. Anfragen für SSL-Signaturen werden zurzeit noch nicht unterstützt.

Kapitel 7

Testumgebung

7.1 Unterstützte Prozessor-Architekturen

7.1.1 Intel x86

Intel x86 und damit kompatible Systeme dominieren den PC-Markt. Diese Architektur ist allein schon aus diesem Grund das primäre Zielsystem für Secure-DSR.

7.1.2 Strong ARM

Die meisten aktuellen Handheld-Computer (PDA's) basieren auf einem Strong ARM Core. Diese stellen nebst Notebooks die wohl grösste Zielplattformen für Ad Hoc Netzwerke dar. Aus diesem Grund wurde grossen Wert darauf gelegt, dass Secure-DSR nebst Intel x86 Systemen auch auf diesen lauffähig ist.

Zur Kompilierung des Kernel-Modules auf diese Systeme wurde die Cross-Compiler Suite *toolchain* [15] verwendet. Das Krypto-Interface hingegen wurde native auf dem ARM-System kompiliert. Es ist bestimmt ebenfalls möglich die toolchain-Suite dafür zu verwenden, obschon bei der mir vorliegenden Version inkompatibilitäten mit den verwendeten Standard-Libraries ausgemacht wurden.

Compaq IPAQ

IPAQ's werden standardmässig mit Pocket PC Betriebssytem ausgeliefert, diese Geräte werden allerdings schon seit längerem von Linux unterstützt. Zur Entwicklung kam die Linux-Distribution *Familiar Linux* (v0.6-rc1) [14] zum Einsatz, mit welchem sehr gute Erfahrungen gemacht wurden. Das Kernel-Modul wie auch das Krypto-Interface sind auf dem IPAQ voll lauffähig.

7.2 Unterstützte Netzwerk-Architekturen

Da die Secure-DSR Software nicht direkt auf der Hardware aufsetzt, sondern lediglich einen bestehenden IP-basierten Network-Layer voraussetzt, sollte sie problemlos auf allen von Linux unterstützen Netzwerkarchitekturen lauffähig sein.

7.3 Testsystem

7.3.1 Wireless LAN

Um ausführliche Tests mit dem implementierten Secure-Routing durchzuführen, wäre es wünschenswert, Feldversuche mit richtigen Broadcast-Netzwerken wie z.B. Wireless LAN (IEEE 802.11) zu tätigen. Ein solcher Test wäre sehr aufwändig, mangels Zeit und verfügbarer Hardware musste ich mich jedoch mit minimalen Tests begnügen:

Aufgebaut wurde ein Ad Hoc Netzwerk bestehend aus zwei Systemen, einem Notebook und einem Compaq IPAQ, beide mit WLAN-Karte. Dies kann zwar nicht als Feldversuch angesehen werden, beweist jedoch, dass die Software prinzipiell auf diesen Systemen lauffähig ist.

7.3.2 VMware

Als einfachstes Testsystem für Kernel-Module hat sich der Einsatz von VMware [16] erwiesen. Mittles VMware ist es möglich, mehrere Betriebssysteme (in diesem Falle Linux) auf einem einzelnen Computer laufen zu lassen, und zwischen den einzelnen VMwares ein virtuelles Netzwerk zu errichten um so das Protokoll ausgiebigen Tests zu unterziehen. Dies ist insofern von grossem Vorteil, da bei der Implementierung von Kernel-Modulen ein kleiner Fehler genügt, um das gesamte Betriebssystem lahmzulegen.

Ein Problem ist jedoch der grosse Delay, der zwangläufig beim simultanen Einsatz mehrerer VMwares entsteht. Zum Beispiel werden dadurch die Resend-Timeouts regelmässig ausgelöst, was keine realistische Simulationsumgebung darstellt. Allerdings wird so der Code auf Extremsituationen[1] getestet.

7.3.3 Ethernet

Im Labor wurde ein Netzwerk auf Ethernet-Basis zu Testzwecken erstellt (siehe Abb. 7.1). Dieses Netzwerk besteht aus einer beliebigen Anzahl Clients (*alpha, beta, ...*) und einem Server (*Dealer*), von welchem die einzelnen Hosts mithilfe von Scripts gesteuert werden. So können Kommandos, um etwa die Secure-DSR Software zu verteilen und zu starten oder die Logfiles zu

[1] z.B. massiv überlastete Hosts oder sehr schlechte Übertragunsverhältnisse

aggregieren, von einem zentralen Ort aus gegeben werden, was die Administration erheblich vereinfacht. Jeder Client hat zwei Netzwerkkarten, die eine für die Verbindung zum Command-Net, die andere zum Simulation-Net, auf welchem Secure-DSR läuft.

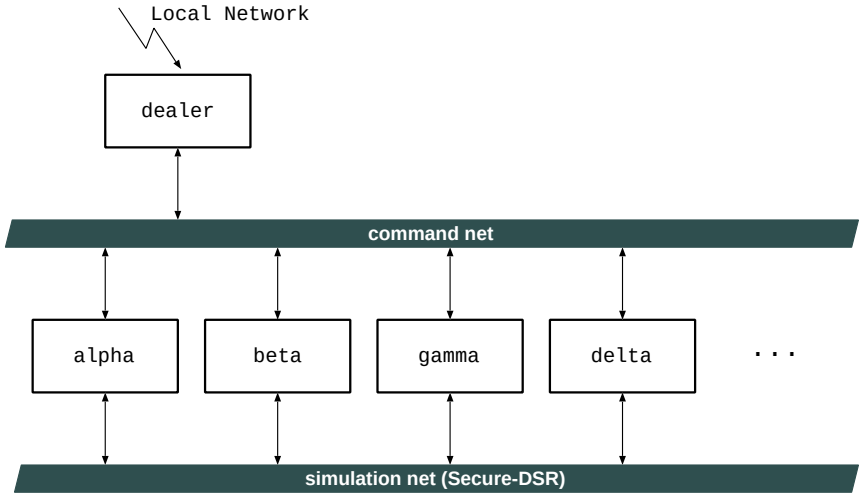


Abbildung 7.1: Anordnung der Hosts in Testumgebung

Simulation eines Ad Hoc Netzwerkes

In einem Ethernet Netzwerk kann jeder Host den anderen sehen, ohne zusätzliche Massnahmen ist also das Simulations-Netz äquivalent zu einem *fully interconnected* Ad Hoc Netzwerk. Um die einzelnen Clients voneinander abzuschirmen müssen also die Datenpakete noch gefiltert werden. Da Secure-DSR direkt auf den Netfilter-Tabellen [6] des Linux-Kernels operiert, kann das Filtern nicht mit `iptables` geschehen.

Aus diesem Grund wurde das Secure-DSR Kernel Modul um Filtering auf MAC-Adressen erweitert.[2] Dem Secure-DSR Modul kann beim Start eine *Droplist* übergeben werden (siehe auch Kap. B.2.2). Alle Pakete mit einer Absenderadresse, welche dort aufgeführt ist, werden dann ignoriert. So wird es möglich, beliebige Ad Hoc Netzwerktopologien, insbesondere auch solche mit unidirektionalen Links, zu simulieren.

[2] Diese Erweiterungen können beim Kompilieren mit dem Flag `DSR_DEBUG_DROPLIST` aktiviert werden.

Kapitel 8

Schlussfolgerungen und Ausblick

8.1 Resultate

8.1.1 Theoretischer Teil

Im Rahmen einer Diplomarbeit wurden diverse Modelle zur Etablierung von *Quality of Service* und *Vertrauen* in MANET's, einem noch sehr jungen Forschungsgebiet, evaluiert.

Sicherheit ist in diesem Zusammenhang ein sehr komplexes Thema. Sowohl die Einarbeitung in die verschiedenen MANET Routingprotokolle und diversen anderen für die Arbeit relevanten Papers, als auch die Suche nach einem möglichst universell einsetzbaren Modells für sicheres Routing in Ad Hoc Netzwerken nahm viel Zeit in Anspruch.

Schliesslich wurde die Idee eines universell einsetzbaren Modells auf eines speziell für Source-Routing Protokolle beschränkt. Daraus entstand das *QoS and Trust Based Secure Source Routing* (siehe Kap. 5).

8.1.2 Praktischer Teil

Auf der Basis des im theoretischen Teil entstandenen *Secure Source Routing* wurde das DSR-Protokoll [18] erweitert und das *Secure-DSR* Protokoll auf Linux implementiert. Dabei handelt es sich um ein voll funktions- und einsatzfähiges Softwarepaket bestehend aus einem Kernel-Modul und einem Userspace-Deamon. Diese Software ist auf allen von Linux unterstützen drahtlosen Übertragungsmedien, insbesondere Wireless LAN (IEEE 802.11), lauffähig. Nebst Intel x86 ist die Software auch kompatibel mit ARM Prozessoren, welches den Einsatz auf PDA's ermöglicht.

Dieses Softwarepaket wurde in einer eigens dazu aufgestellten Testumgebung getestet. (siehe Kap. 7.3.3)

8.2 Ausblick

8.2.1 Quality of Service

Ein zentraler Bestandteil des Secure-DSR Protokolls wurde noch nicht implementiert: die Ermittlung der eigenen QoS.

8.2.2 Evaluation

Aus Zeitmangel konnte das Secure-DSR Protokoll nicht evaluiert werden. In diesem Abschnitt sind die Aspekte des Protokolls aufgeführt, welche noch zu evaluieren wären:

- *Routenwahl*: Wie stark erhöht sich die Stabilität bei Verwendung von Secure-DSR? Wieviel Lastverteilung kann erreicht werden? Oder wird gar das gesamte Netz dadurch langsamer und instabiler?
- *QoS*: Nach welchen Gesichtspunkten soll die QoS ausgehend von verschiedenen Szenarien gemessen werden?
- *Distanzfunktionen* (Kap. 6.4.2): Sind diese Funktionen sinnvoll für eine optimale Routenwahl? Was sind geeignete Werte für α, β und γ?
- *Performance*: Wie sehr beanspruchen die verwendeten Krypto-Algorithmen die Ressourcen eines Hosts in einem grösseren Secure-DSR Netzwerk?
- *Overhead*: Wie sehr steigern die Secure-DSR Erweiterungen den Overhead im Netzwerk?
- *Timing* (Kap. 5.3): Ein vernünftiger Wert für die Gültigkeitsdauer von QoS/Trust-Werten muss ermittelt werden, in Hinblick auf Performance und Overhead im Secure-DSR Netzwerk.

Anhang A

Verwendete Software-Tools

Für diese Arbeit wurden folgende Software-Tools eingesetzt:

- Linux 2.4.18 [5]
- gcc 2.95.4 [17]
- GnuPG 1.0.7 [7]
- GPGME 0.3.9 [8]
- Familiar Linux v0.6-rc1 [14]
- Toolchain (14-Jul-2000) [15]
- VMware 3.1.1 [16]

Anhang B

Aufsetzen des Secure-DSR Systems

B.1 Verzeichnisstruktur

`cryptint/`	Krypto-Interface Quellcode
`doc/presentation/`	Präsentation (Open Office)
`doc/report/`	Bericht (LaTex)
`doc/tgif/`	Grafiken (Tgif)
`dsr/`	Secure-DSR Kernel Modul Quellcode
`testsuite/`	DSR Paket Generator

B.2 Kompilierung der Secure-DSR Software

B.2.1 Vorbereitungen

1. Für die Cross-Kompilation des Kernel-Modules auf ARM-Plattform muss die Toolchain Compiler Suite installiert werden [15].

2. Installieren Sie GnuPG [7] sowie GPGME [8]. Dies wird vom Krypto-Interface benötigt.

3. Erstellen sie ein Schlüsselpaar (Secret-Key und Public-Key) mit den von GnuPG bereitgestellten Mitteln.

4. Aktivieren Sie Routing:
   `echo 1 > /proc/sys/net/ipv4/ip_forward`

B.2.2 Kernel-Modul

Der Source-Code für das Kernel-Modul befindet sich im Verzeichnis `dsr`.

1. Wechseln Sie in das Verzeichnis `dsr`.

2. Modifizieren Sie die Datei `Makefile`: setzen sie die Pfade der Kernel-Header (`HPATHx86` und `HPATHarm`) korrekt.

3. nach Aufruf von `make` wird das Kernel-Module sowohl für x86 (`dsrx86.o`) als auch für ARM (`dsrarm.o`) kompiliert.

4. um das Modul in den Kernel einzubinden verwenden Sie den Befehl:
   `insmod dsrx86.o [param=value...]`

   Folgende Parameter sind verfügbar:

`ifname`	Name des Netzerkadapters (default=`eth0`)
`gw`	Gateway-Modus: 0=off, 1=on (default off)
`drop`	Liste aller MAC-Adressen, welche beim Empfang ignoriert werden sollen.

Beispiel:
```
insmod dsrx86.o ifname=eth1 droplist="12:34:56:78:90:ab 00:00:ff:ab:de:f0"
```

B.2.3 Krypto-Interface

Der Source-Code für das Krypto-Interface befindet sich im Verzeichnis `cryptint`. Es ist momentan keine Cross-Kompilierung für ARM möglich. Für den Einsatz auf diesen Systemen muss die Software native kompiliert werden.

1. Wechseln Sie in das Verzeichnis `cryptint`.

2. nach Aufruf von `make` wird das Krypto-Interface kompiliert

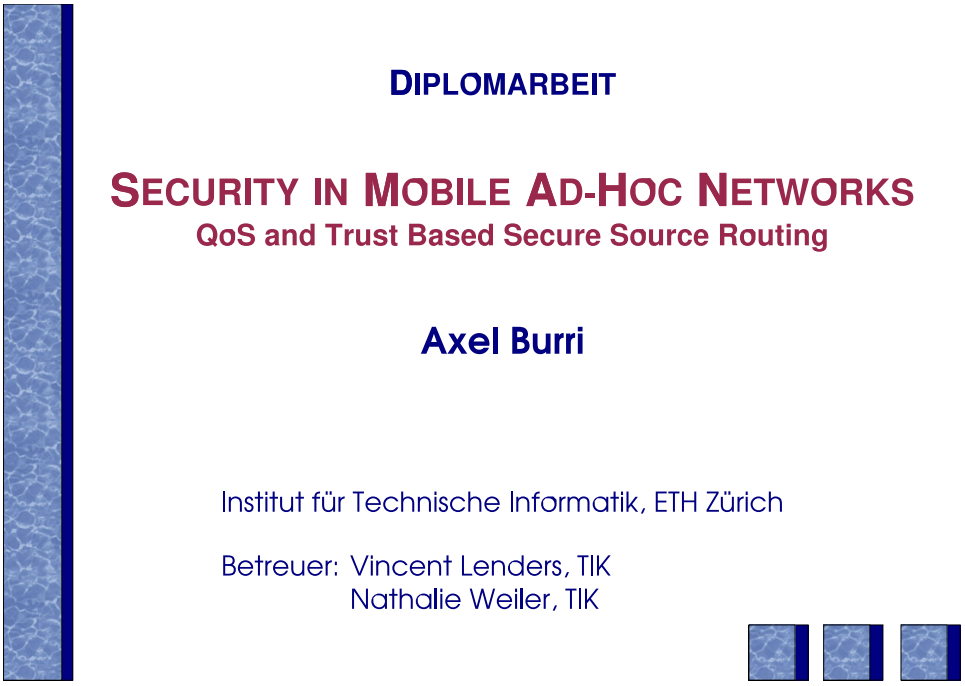
3. Starten Sie das Programm:
   `cryptint_gpg [-d] [-p password] [sign_key...]`.

   Folgende Optionen sind verfügbar:

`-d`	Start als Daemon (empfohlen)
`-p password`	Passwort für den ausgewählten Schlüssel
`sign_key`	Signatur-Schlüssel. Wird dieser nicht angegeben wird der Standard-Schlüssel von gpg benutzt

Anhang C

Präsentation



Inhalt

- Einführung in Mobile Ad Hoc Netze (MANET's)
- Einführung DSR
- Motivation
  - Quality of Service (QoS)
  - Web of Trust
- Erweiterung des DSR Protokolls
  - Secure DSR
  - Distanzfunktionen

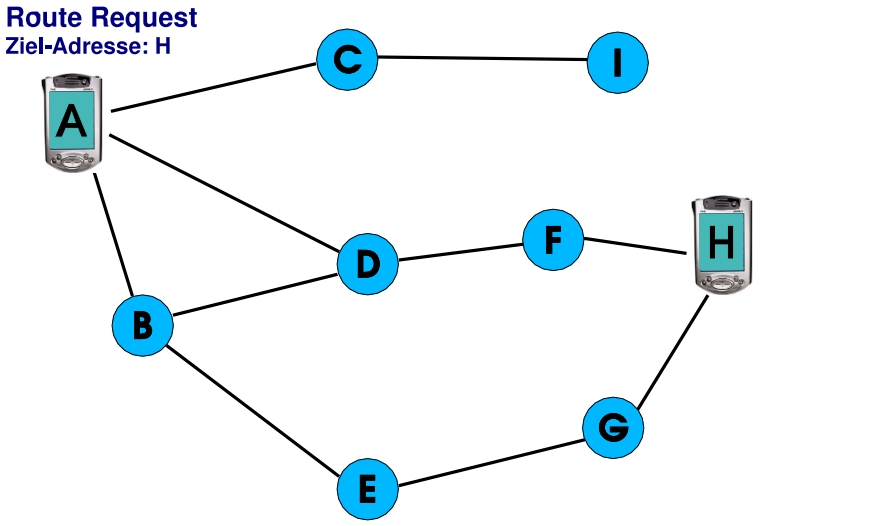
Mobile Ad-Hoc Networks

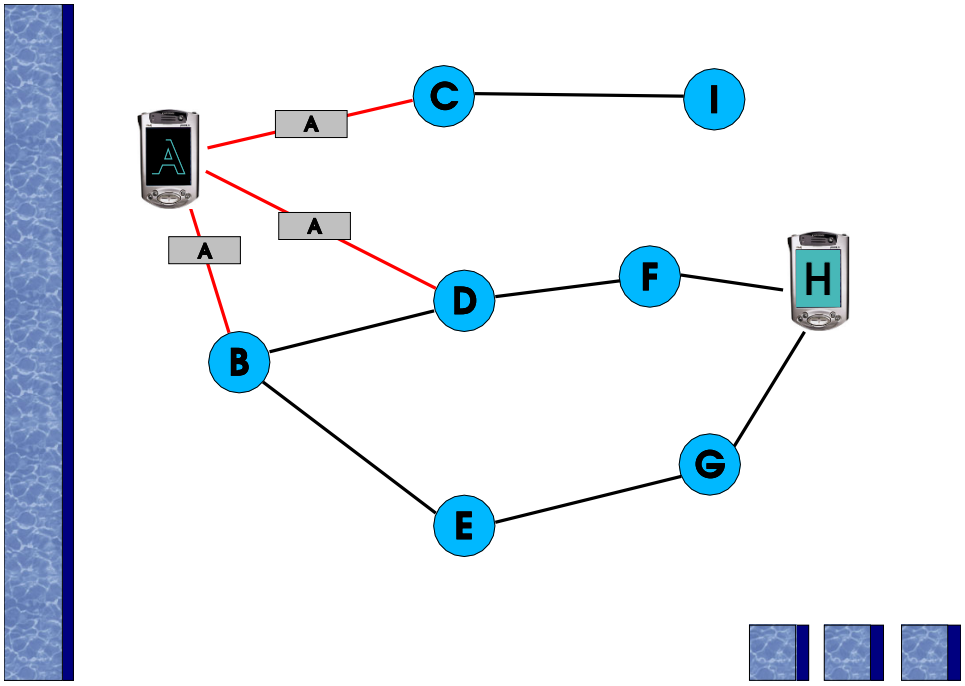
- Ausschliesslich von drahtlosen und mobilen Geräten gebildete Netze
- Müssen sich selbst organisieren
- Jedes Gerät ist auch ein Router
- Routing Protokolle:

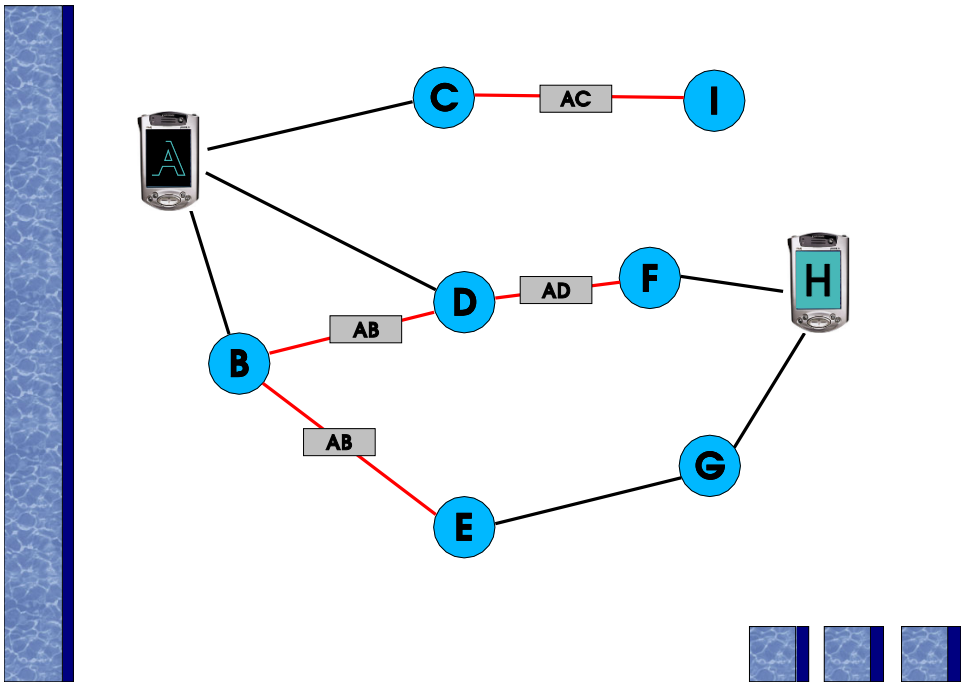
Proaktiv	Reaktiv	Hybrid	Hierarchisch
DSDV	DSR	ZRP	LANMAR
OLSR	AODV		
TBRPF	TORA		
FSR			

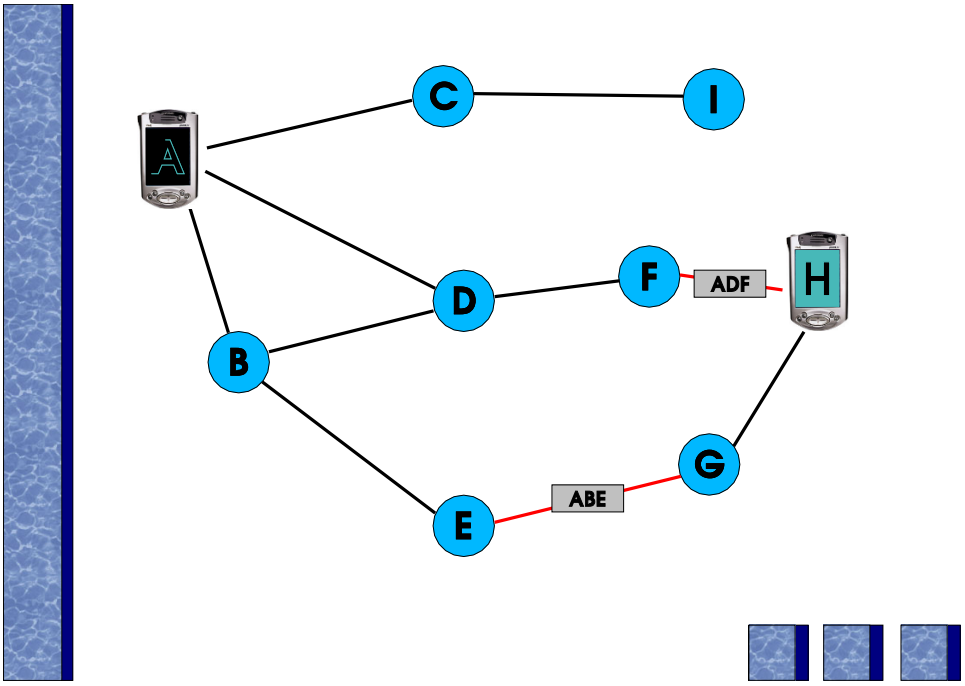
Dynamic Source Routing (DSR)

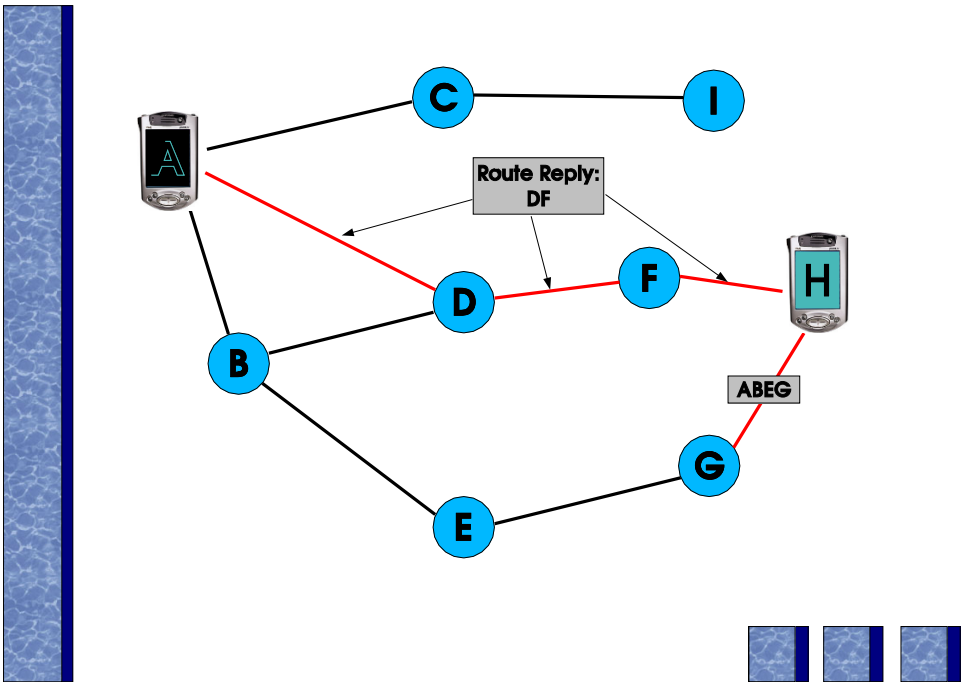
- Source Routing
  - Jedes Datenpaket beinhaltet eine DSR Source-Routing Option
- Route Discovery (*Reaktiv*)
  - Route Request: Suche nach Route zu Zielhost mittels *Flooding*
  - Route Reply: Antwort des Zielhosts auf *Reverse Path*

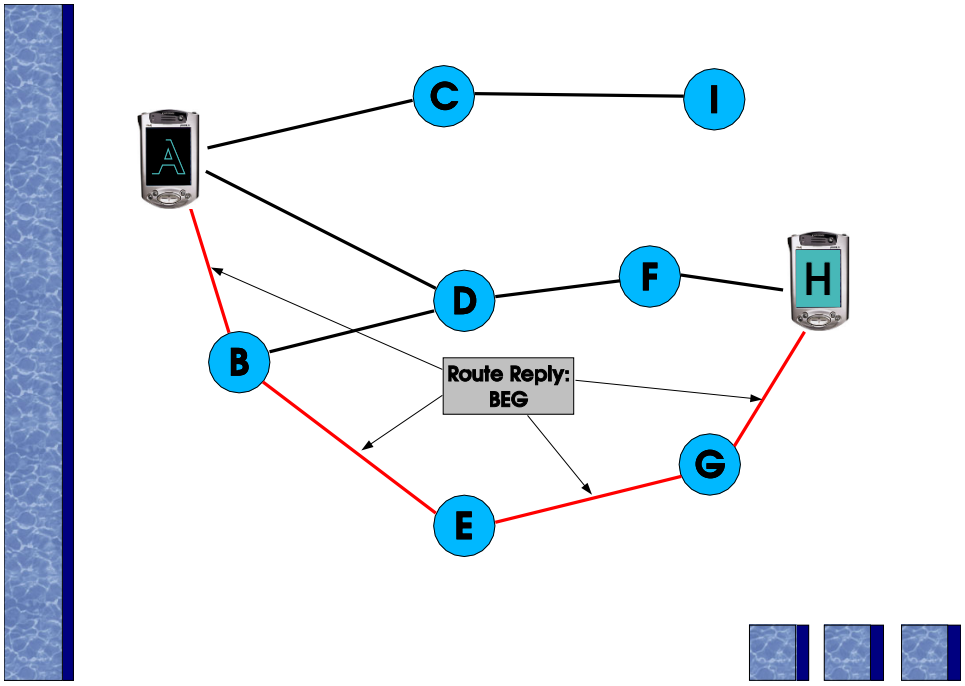


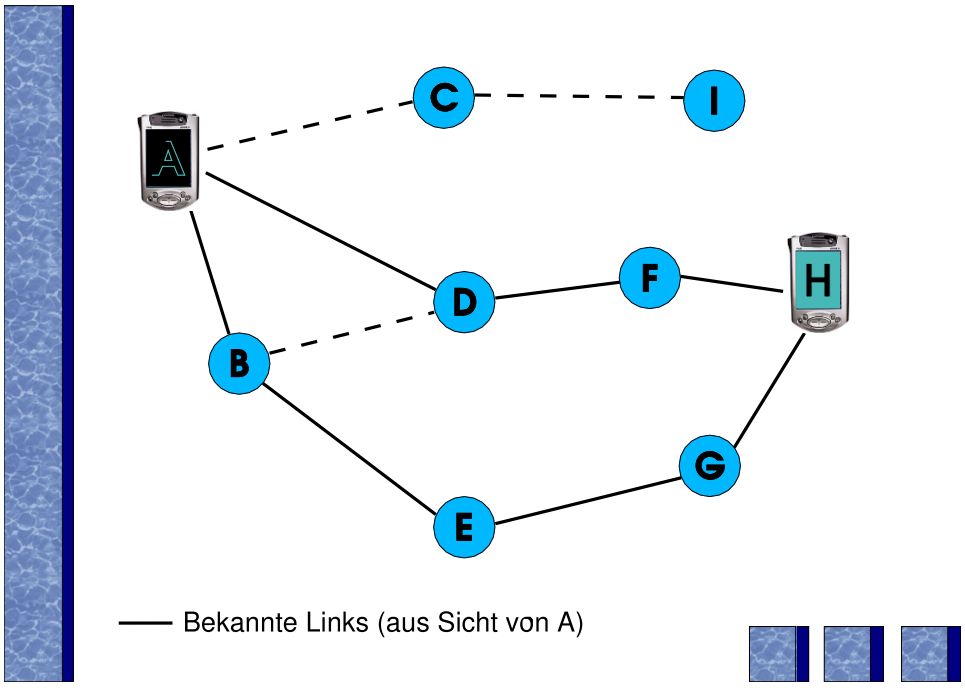

A
C
I
A
A
D
F
H
B
G
E


C
AC
I
D
AD
F
H
AB
B
AB
G
E


A
C
I
D
F
ADF
H
B
G
ABE
E


A
C
I
Route Reply:
DF
D
F
H
B
ABEG
G
E


A
C
I
D
F
H
B
Route Reply:
BEG
G
E


A
C
I
D
F
H
B
G
E
Bekannte Links (aus Sicht von A)

Motivation

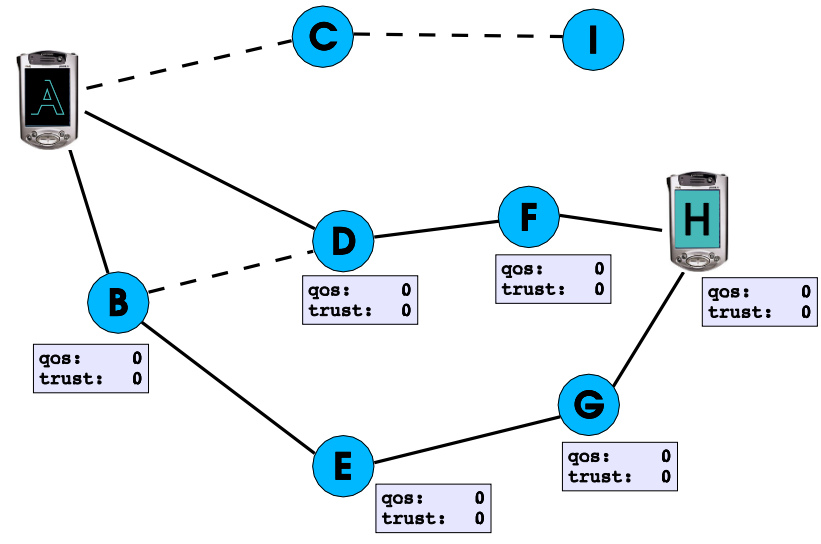
- DSR ist äusserst Anfällig auf *Denial-of-Service* Attacken
  - Notwendigkeit einer Authentisierung aller Hosts
- DSR wählt als Route immer diejenige mit *Minimum Hops*
  - Dies ist nicht unbedingt die beste Route!
  - Besser: Routenwahl nach *Quality of Service*

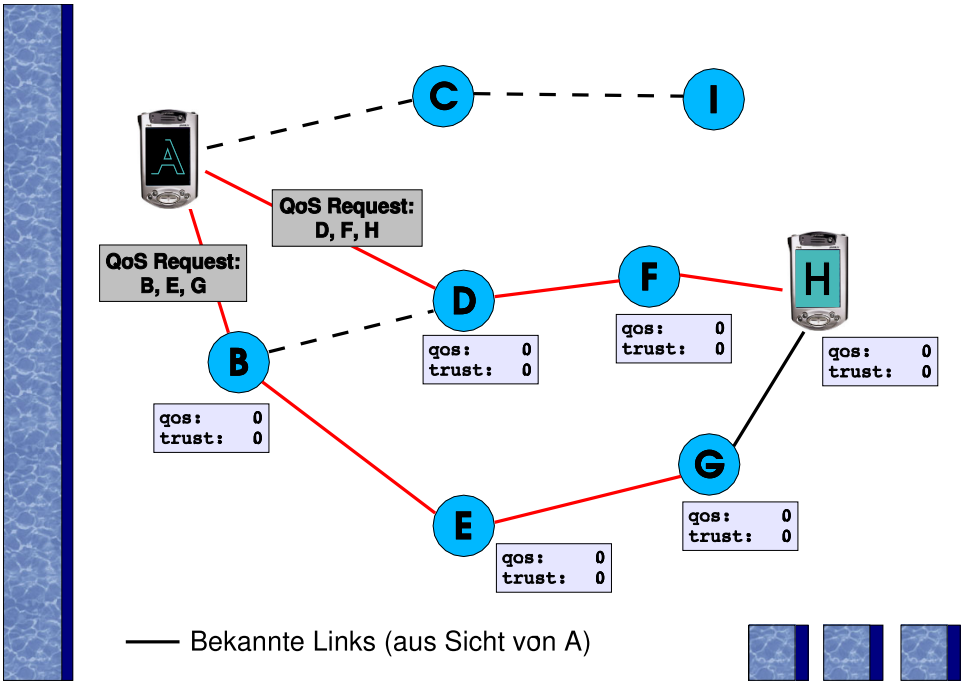
Quality of Service eines Hosts

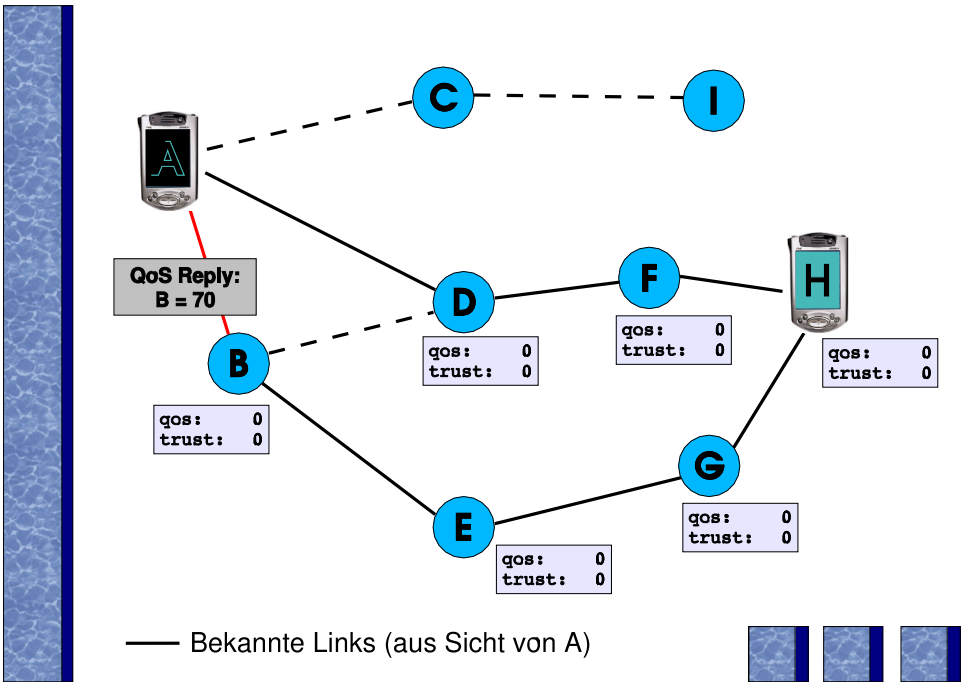
- Wozu?
  - Finden der besten Route
  - Lastverteilung
- Information für:
  - Auslastung
  - Stationarität
  - Batteriestatus
- Problem: Neue Attacken möglich
  (Angreifer kann QoS beliebig modifizieren!)
- Abhilfe: *Zertifizierte QoS*
  - Tupel QoS/Trust
  - *Web of Trust*

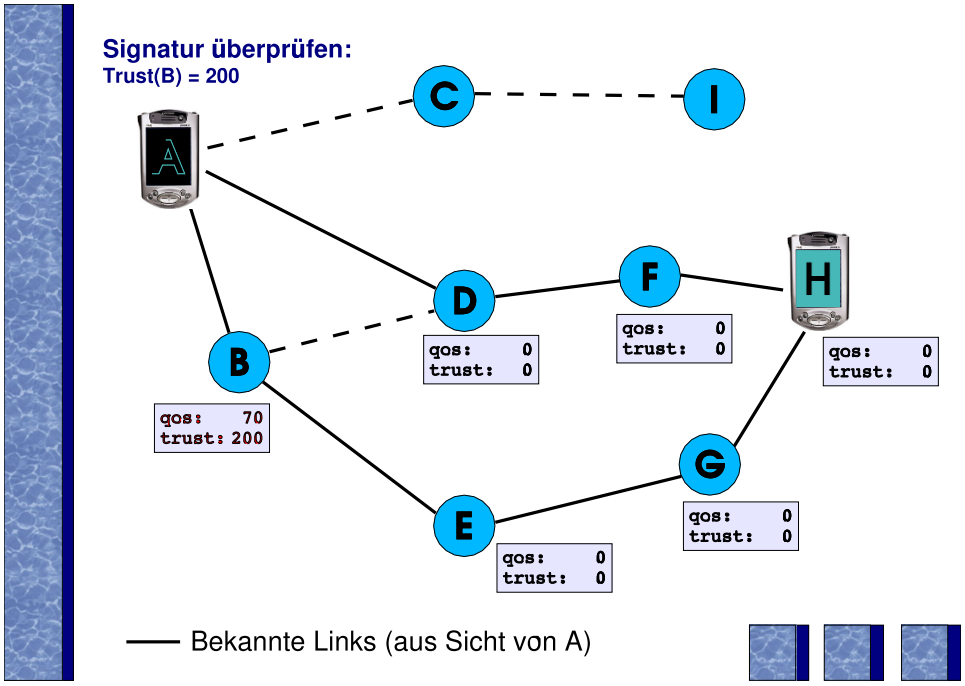
Erweiterung des DSR-Protokolls

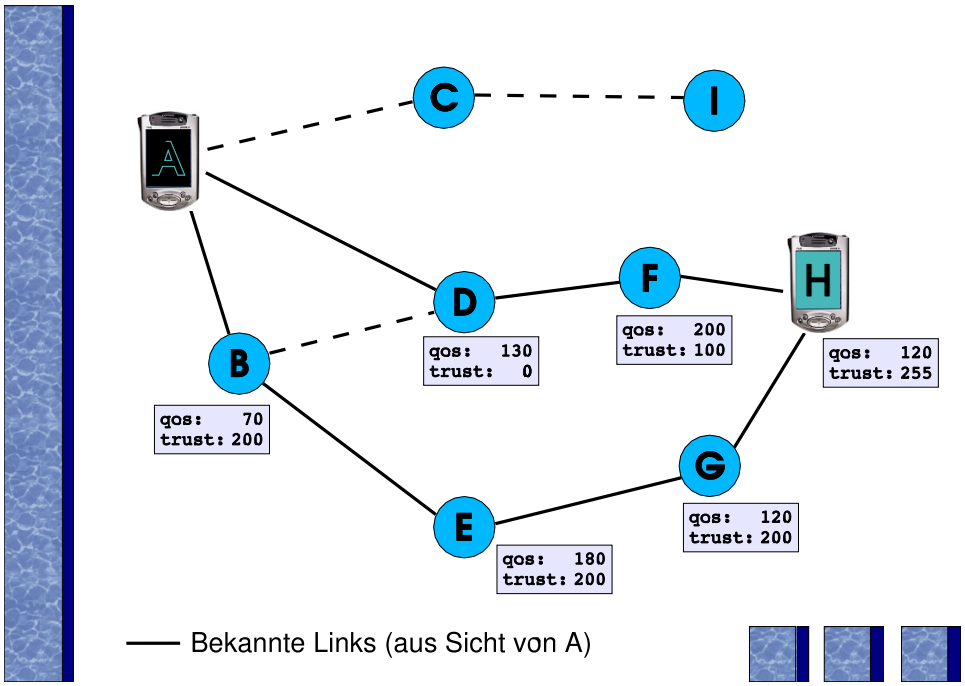
- QoS Request
  - Asynchron
    - Vermeidung von Performance Engpässen aufgrund langwieriger Krypto-Operationen
  - An alle unbekannten Hosts (z.B. nach Route-Reply)
- QoS Reply
  - Signierte QoS
  - *Web of Trust* zur Ermittlung von Vertrauen




A
C
I
QoS Request:
D, F, H
QoS Request:
B, E, G
D
F
H
qos: 0
trust: 0
B
qos: 0
trust: 0
qos: 0
trust: 0
qos: 0
trust: 0
G
E
qos: 0
trust: 0
qos: 0
trust: 0
Bekannte Links (aus Sicht von A)


A
C
I
QoS Reply:
B = 70
D
F
H
qos: 0
trust: 0
B
qos: 0
trust: 0
qos: 0
trust: 0
qos: 0
trust: 0
G
E
qos: 0
trust: 0
qos: 0
trust: 0
Bekannte Links (aus Sicht von A)

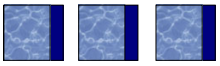

Signatur überprüfen:
Trust(B) = 200
A
C
I
D
F
H
B
qos: 0
trust: 0
qos: 0
trust: 0
qos: 0
trust: 0
qos: 70
trust: 200
G
E
qos: 0
trust: 0
qos: 0
trust: 0
Bekannte Links (aus Sicht von A)


A
C
I
D
F
H
B
qos: 130
trust: 0
qos: 200
trust: 100
qos: 120
trust: 255
qos: 70
trust: 200
G
E
qos: 120
trust: 200
qos: 180
trust: 200
Bekannte Links (aus Sicht von A)

Distanz-Funktionen

- Minimum Hops $D(a,b)=1$
- Maximum QoS $D(a,b)=\alpha/Q(b)$
- Maximum Trust $D(a,b)=\beta/T(b)$
- Kombiniert $D(a,b)=\alpha/Q(b)+\beta/T(b)+\gamma$

D(a,b): Distanz von Link [a,b]
Q(b): QoS Wert von b
T(b): Trust Wert von b



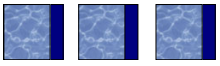
Paket-Typen

QoS Request

- Signatur-Typ
- Nonce
  (Zufallswert, verhindert Replay Attacken)
- Ziel-Adresse[0..N]

QoS Reply

- QoS-Wert
- Signatur-Typ
- Adresse
- Nonce
- Signatur



Erreichte Ziele

- Erarbeitung eines **Secure DSR** Modells
  - Kernel Modul Implementierung
  - GNU Privacy Guard (GPG) Schlüsselmanagement
- Aufbau einer Testumgebung auf Ethernet-Basis
  - Test der Implementierung

Weiterführende Arbeiten

- Evaluierung des **Secure DSR** Modells
  - Messung des Overheads (Traffic, Performance)
  - Revidieren der Distanz-Funktionen
  - Reaktion auf Attacken?
- Ermittlung der eigenen QoS
  - Auslastung
  - Batteristatus
  - Stationarität

Danke für Ihre Aufmerksamkeit

Literaturverzeichnis

[1] MANET (Mobile Ad Hoc Networking) Working Group
www.ietf.org/html.charters/manet-charter.html
(September 2002)

[2] Charles E. Perkins
Ad Hoc Networking
Addison-Wesley, ISBN 0-201-30976-9, 2001

[3] List of papers on security for ad hoc networks
www.ccs.neu.edu/home/zhufeng/security_manet.html
(September 2002)

[4] AirSnort: a wireless LAN (WLAN) tool which recovers encryption keys
airsnort.shmoo.com
(September 2002)

[5] Linux Kernel
www.kernel.org
(September 2002)

[6] The netfilter/iptables project
www.netfilter.org
(September 2002)

[7] The GNU Privacy Guard (GnuPG)
www.gnupg.org
(September 2002)

[8] GPGME - GnuPG Made Easy
www.gnupg.org/gpgme.html
(September 2002)

[9] Internet X.509 Public Key Infrastructure Certificate and CRL Profile
www.ietf.org/rfc/rfc2459.txt
(September 2002)

[10] GNU General Public License (GPL)
www.gnu.org/copyleft/gpl.html
(September 2002)

[11] Lidong Zhou and Zygmunt J. Haas
Securing Ad Hoc Networks
Department of Computer Science, Cornell University, Ithaca, NY USA
www.cs.cornell.edu/home/ldzhou/adhoc.pdf
(September 2002)

[12] Alex Song
DSR implementation: *Piconet II*
sourceforge.net/projects/piconet
(September 2002)

[13] Rice Monarch Project (Rice University, 1998)
DSR implementation
www.monarch.cs.rice.edu/dsr-impl.html
(September 2002)

[14] *Familiar Project*: Linux Distribution for Handheld Computers
familiar.handhelds.org
(September 2002)

[15] *Toolchain*: ARM Linux Cross Compiler Suite
handhelds.org/download/toolchain
(September 2002)

[16] VMware workstation
www.vmware.com
(September 2002)

[17] GNU C Compiler
gcc.gnu.org
(September 2002)

[18] *Dynamic Source Routing Protocol (DSR)*
www.ietf.org/internet-drafts/draft-ietf-manet-dsr-07.txt
(September 2002)

[19] *Zone Routing Protocol (ZRP)*
www.ietf.org/internet-drafts/draft-ietf-manet-zone-zrp-04.txt
(September 2002)

[20] *Ad Hoc On Demand Distance Vector (AODV) Routing*
www.ietf.org/internet-drafts/draft-ietf-manet-aodv-11.txt
(September 2002)

[21] *Optimized Link State Routing Protocol (LSR)*
www.ietf.org/internet-drafts/draft-ietf-manet-olsr-06.txt
(September 2002)

[22] *Topology Broadcast based on Reverse-Path Forwarding (TBRPF)*
www.ietf.org/internet-drafts/draft-ietf-manet-tbrpf-05.txt
(September 2002)

[23] *Landmark Routing Protocol (LANMAR) for Large Scale Ad Hoc Networks*
www.ietf.org/internet-drafts/draft-ietf-manet-lanmar-04.txt
(September 2002)

[24] *Fisheye State Routing Protocol (FSR)*
www.ietf.org/internet-drafts/draft-ietf-manet-fsr-03.txt
(September 2002)

[25] *The Interzone Routing Protocol (IERP)*
www.ietf.org/internet-drafts/draft-ietf-manet-zone-ierp-02.txt
(September 2002)

[26] *The Intrazone Routing Protocol (IARP)*
www.ietf.org/internet-drafts/draft-ietf-manet-zone-iarp-02.txt
(September 2002)

[27] *The Bordercast Resolution Protocol (BRP)*
www.ietf.org/internet-drafts/draft-ietf-manet-zone-brp-02.txt
(September 2002)

[28] *Highly Dynamic Destination-Sequenced Distance Vector Routing (DSDV)*
www.cs.odu.edu/ lee_m/papers/perkins-bhagwat-sigcommr94.pdf
(September 2002)