# Ad-hoc 라우팅 계층에서의 서비스 디스커버리 프로토콜 구현*

김보성[0] 고영배[1] 노용성[2]
[0,1]아주대학교 정보통신 전문대학원, [2]삼성 종합기술원
winnerxg@dmc.ajou.ac.kr, youngko@ajou.ac.kr, yongsung.roh@samsung.com

# Implementation of service discovery protocol on routing layer in Ad-hoc network environment

Bo-Seong Kim[0] Young-Bae Ko[1] Yong-Sung Roh[2]
[0,1]Graduate School of Information and Communication, Ajou University [2]Samsung Advanced Institute of Technology

**요 약**

향후 세상의 모습을 그려볼 때 흔히들 유비쿼터스 시대를 언급한다. 이를 위해서는 다양한 서비스 공급자들과 사용자들이 서로 네트워크로 묶여 있어야 하며, 이때의 네트워크는 선이 없는 무선 환경이고, 국소 지역을 커버하는 Ad-hoc망이 기본 네트워크 망이 될 것이다. 이러한 네트워크로 이루어진 다양한 장소, 임의의 시간에 존재하는 다수의 서비스들 중에서 원하는 서비스를 찾는 일을 service discovery라고 하며, 이를 위해서 기존의 wired 망에서는 네트워크 계층 위에서 이를 수행하는 작업이 이루어져 왔다. 본 논문에서는 service discovery 실행을 별도의 에이전트나 응용 레벨에 두지 않고, 이를 Ad-hoc 라우팅 계층에서 수행 함으로써 오버헤드 트래픽이 줄어들고, 응답 시간이 짧아지는 성능 향상이 있음을 제시하고 있다. 또한 동적/부분 caching 기법에 대해서도 제시하고 있다. 아울러 다양한 사용자의 기호(User Preference)를 표현하고, 이를 바탕으로 서비스를 검색하기 위한 새로운 서비스 표현/검색 모델을 제시하였다. 실제로 이러한 service discovery 기능을 Ad-hoc 라우팅 계층에서 처리하도록 하는 Ad-hoc 라우팅 데몬을 구현하고, 테스트 베드를 구축하여 동작 시킴으로써 실제 상황에서도 얼마나 효율적인가를 구체적으로 제시하고 있다.

## 1. 서 론

Ad-hoc 환경은 유선환경과 비교했을 때, 특수한 환경이라고 말할 수 있다. 노드의 mobility, 불규칙한 RF채널의 속성으로 인한 링크의 불안정성, Unicast와 Broadcast간에 각각 서로 다른 data rate을 사용함으로써 발생하는 transmission range의 차이로 인한, heterogeneous 문제점 등은 기존의 wired 망에서와는 다른 라우팅 방식을 요구한다. 이렇듯 불안정한 무선링크를 매체로 하는 Ad-hoc 노드들이 multi-hop 토폴로지 상에서 데이터를 주고 받는 것 이면에는, 제안된 라우팅 마다 고유의 scheme에 따라서 라우팅 컨트롤 패킷들을 주고 받게 된다. 또한 Ad-hoc 노드들은 각자가 라우터의 역할을 하기도 하며, 서비스 공급자(Service Provider: SP)의 역할을 하기도 하며, 서비스 사용자(Service Demander: SD)이기도 하다. 원하는 서비스를 찾기 위해서는 서비스를 제공하는 SP의 주소를 찾아야 하며, 또한 해당 SP의 포트 번호와 같은 부가적인 정보도 알아야 한다. 이 논문에서는 " Service discovery" 를 라우팅 계층에서 수행함으로써 얻을 수 있는 이점에 대해서 설명하고, 사용자의 기호(User Preference)를 반영하는 Service Discovery 모델을 제시할 것이다. 또한, 이러한 내용을 실제로 구현해서 다른 Service Discovery 방식들과 비교 분석한다.

## 2. 관련연구

서비스 공급자(SP)를 찾는 서비스 사용자(SD)는 SP의 IP주소, 서비스 포트 번호와 같은 정보를 알아내기 위해서 어떠한 에이전트나 미들웨어 시스템을 이용하게 될 것이다. 그리고 대부분의 이러한 에이전트 시스템은 라우팅 계층 위에서 수행된다. Service Discovery를 위한 에이전트나 미들웨어 시스템들은 일반적으로 두 가지 수행 절차를 거쳐야만 원하는 서비스를 찾고, 해당 서비스 공급자(SP)에게 접근하여 서비스를 받을 수 있다. 이는 다음과 같다. 첫 번째로 SD는 SREQ(Service Request), SREP(Service Reply) 패킷과 같은 컨트롤 패킷들을 주고 받음으로써 " Service Discovery" 과정을 수행한다. 이로써 앞서 언급한 것처럼 SP의 IP주소, 포트 번호와 같은 정보를 얻게 된다. 두 번째는 앞서 얻어온 IP 주소로 향하는 route를 찾는, " Route Discovery" 과정을 수행 해야 한다. 이처럼 두 단계의 *① Service Discovery, ② Route Discovery* 과정을 거쳐야만 한다는 의미는, 최소한 두 번의 Flooding이 네트워크 전체에 걸쳐서 일어 나야 한다는 뜻이 된다. (이는 SREQ, RREQ와 같은 request 패킷들의 특성에 기인한다.) 이 논문에서는 이러한 두 단계의 절차를 한 번에 수행하는 방법을 사용해서 성능 향상을 보여준다.

또한, Service Discovery의 모델에 대해서는 아직 정확한 표준이 없다. 기존에는 URL 기반으로 서비스를 명세하고, 이러한 URL을 처리하도록 하는 방법이 제안되기도 하였다. 가령, 프린터 서비스를 명시하고자 한다면 사용자는 " /printer/laser/A4/600dpi" 와 같은 URL정보를 SREQ에 실어 보내고, 서비스 공급자는 자신이 서비스 가능한 요구에 해당하는 URL을 포함하고 있는지를 체크 하도록 하는 방식이다. 하지만, 앞서 제시된 URL기반의 서비스 명세는 계층적이라는 특징을 보이고 있다. 이는 어떠한 사용자가 특정 속성에 가중치를 두어서, 선호도를 보이는 요구를 표현하는 데 있어서 단점이 된다. 이 논문에서는 이러한 사용자 기호(User Preference)를 반영하기 위해 서비스 속성(Service Property) 기반의 서비스 명세를 채택함으로써 다양한 Query Model을 구현해 볼 수 있었다.

## 3. Service Discovery의 성능 향상 방안

### 3.1 *" Service Discovery" 와 " Route Discovery" 의 통합*

Service Discovery 기능을 라우팅 계층에서 함께 수행한다면, " *서비스를 찾기 위해서 사용자가 SREQ패킷을 네트워크에 flooding 했다. 그러므로 곧 SREP가 나에게로 돌아올 것이다.*" 에서 그치지 않고, 실제로 SREQ/SREP가 거쳐간 경로가 각각의 중간 노드의 라우팅 테이블에 기록이 된다. 이 결과, SREP를 받은 사용자(SD)는 SREP로부터 SP의 주소를 얻을 수 있을 뿐만 아니라, 이미 SP로 가는 경로(route)가 Ad-hoc 네트워크상에 설정되어 있으므로, 다시 " RREQ flooding" , " RREP reception" 과 같은 Route Discovery

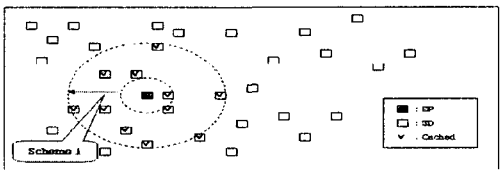
* 본 논문은 삼성종합기술원과의 공동연구에 의해 수행 되었음

과정을 거치지 않아도 된다. 즉, Service Discovery를 위해서 SREQ/SREP 시퀀스가 한번 수행되고 나면, 추가적인 Route Discovery(RREQ/RREP 시퀀스)가 불필요 하다는 의미이다. Service Discovery를 라우팅 계층에서 수행하도록 하는 접근 방법은 Ad-hoc 네트워크의 링크가 불안정하고, 성능이 낮다는 점을 고려할 때, 응답 시간이나, 컨트롤 패킷에 의한 오버헤드 측면에서 상당한 이점이 된다.

### 3.2 *Service request frequency를 고려한 동적/부분 Caching*

동적/부분적으로 캐싱을 하도록 하는 아이디어를 추가함으로써 더욱 효율적인 Service Discovery를 가능케 할 수 있다. 이는 극단적인 Caching과 No-Caching의 중간 형태로서, SP가 사용되는 빈도에 따라서 동적으로 캐싱의 범위를 조절할 수 있도록 함으로서 네트워크 측면에서는 오버헤드를 줄이고, 사용자(SD) 측면에서는 응답 시간을 줄이는데 기여를 한다. 우선, 이러한 동적/부분적 캐싱을 일반적인 형태의 Caching/No-Caching과 비교, 분석을 해보자.

Service Discovery를 위한 시스템들은 구조적으로 볼 때, 크게 두 가지로 분류된다. 즉, push based 방식과 pull based 방식이 있다. Push based방식은 SP가 자신이 외부로 알리고자 하는 정보들(e.g., 서비스 타입/스펙/설명/, IP주소, 포트 번호)을 전체 네트워크에 advertisement하는 형태의 시스템이다. 이러한 형태는 네트워크상에 주기적인 혹은 스킴에 따라 간헐적인 advertisement 패킷이 broadcast혹은 multicast 되야 하므로, 네트워크상에 파급되는 컨트롤 패킷들에 의한 오버헤드가 크다. 하지만 SP를 찾는데 걸리는 응답 시간이 짧다. 반면에, pull based 방식은 SP가 advertisement를 하지 않고, SP를 찾아야 할 필요가 있을 때마다 사용자의 SREQ 패킷이 직접 SP까지 이르러서야 해당 SP의 정보를 알 수 있는 방식이다. 이 방식은 네트워크상에 파급되는 컨트롤 패킷들(SADV, SREQ, SREP)에 의한 오버헤드가 적다는 장점이 있는 반면에, 캐싱에 의한 효과를 보지 못하므로 사용자 입장에서는 그만큼 응답 시간이 길다는 단점이 있다. 그러므로 이러한 극단적인 Push(Caching), Pull(No-Caching) 기반의 방식을 피하고, 이 두 방식의 장점을 채택한 hybrid형태의 caching을 하도록 한다.




<그림 1> 동적/부분 Caching을 위한 Scheme 1의 동작

실제로 hybrid caching이라는 의미는 단순히 Caching/No-Caching의 중간인 Partial-Caching만을 의미하지 않는다. 이 Partial-Caching의 범위는 동적으로 조절이 되어야만 더욱 효율적이다. 가령, SP는 자신의 서비스가 자주 사용되는 경우, 그 caching의 범위를 넓히고, 사용 빈도가 낮아 졌을 때는 이렇게 넓혀진 caching 범위를 좁혀야 효율적인 동적/부분적 caching이 수행될 것이다. 이 때, caching의 범위는 단순히 Advertisement Hop Range로 조절되는 것에만 국한되지 않고, Data-Flow기반의 Caching 방식으로 더욱 확장될 수 있다. 앞의 <그림1>에서는 이와 같은 동적/부분적 caching의 두 가지 방식 중, 홉 수 조절에 기반한 Caching-Scheme 1의 동작을 제시하고 있다.

<그림1>에서 보다시피 Scheme 1은 Advertisement hop range를 증/감 시키면서 SADV 패킷을 broadcasting하는 방식이다. 이때, range 증/감은 다음 식에 의해서 수행하였다.

- range: Advertisement 범위
- N(x): 시간 [x-1]에서 [x]사이에 SD가 SP에 접속하거나 세션을 열어서 서비스를 제공받은 횟수.
  (구현에서는 편의를 위해서 TCP접속만을 고려하였다.)
- range >= 1 : scheme 1에서의 최소 range는 1이다.

***range 하나 증가: if N(t) > N(t-1) * 1.5***
***range 하나 감소: if N(t) < N(t-1) * 1.0***

### 3.3 *사용자 기호를 반영하는 Service Discovery 모델*

서비스를 요구하는 사용자는 " 프린트 서비스를 찾아와라." 혹은 " 스트리밍 서버를 찾아와라" 와 같은 매우 단순한 query를 원하지는 않을 것이 분명하다. 특히, 까다로운 사용자는 " A4출력이 가능하고, 컬러 출력이 가능하며, 600dpi 해상도를 모두 만족하는 프린트 서비스를 찾아와라" 라고 요구할 수 있다. 또한 앞서의 조건을 75%(proximity threshold) 이상 만족시키는 서비스만을 찾아와라" 라고 매우 까다롭게 자신의 기호를 표현할 수도 있다. 즉, 사용자의 요구 중심으로 Service Discovery가 수행되어야 함을 의미한다.

이러한 기능을 위해서는 앞서 언급한 URL기반의 서비스 표현은 적당하지 않다. 이를 해결하기 위해서 우리는 고정적 계층 구조가 아닌, 유연하며 확장이 편리한 속성(Property)기반의 표현을 고안했다. 즉, 하나의 서비스를 표현하기 위해서는 해당 서비스가 가지고 있는 자신만의 속성을 하나의 벡터(Service Property Vector)에 포함하도록 한다. 이러한 속성 기반의 벡터 표현은 표현이 간결하며, 어떠한 서비스든지 표현이 가능하다. 서비스 공급자(SP)는 자신이 등록하고자 하는 서비스의 특징을 하나씩, 하나씩 Service Property Vector에 기록한다. 이때, 어떠한 계층 구조나 순서 같은 규약은 없다. 단지 [Category : Value] 의 한 쌍이 벡터의 한 엔트리를 형성하게 된다는 점이 중요하다. 이때, Category란 해당 서비스 벤더들 간에 통용되는 표준 속성일 수도 있고, 벤더 고유적으로 정의해 놓은 스페셜 속성(기능)일 수도 있다. Value는 해당 속성의 값이다.

SP가 이렇게 자신의 서비스를 명세하여 등록한 이후에는, 각 노드의 라우팅 데몬 내에 존재하는 Service Discovery 모듈이, 다양한 요구를 가진 서비스 사용자들(SD)에 의해 요청된 query를 처리 하게 된다. 예를 들어, SD측에서는 ***" Service_Type=S1 and (0.9 > prob(A=p and D=q and B=r))"*** 라는 query를 SREQ에 실어서 flooding하게 할 수도 있다. 이러한 query 모드를 Proximity Matching Mode라고 정의 하였다. 이외에 3가지 Matching Mode가 더 있다. ***" Service_Type=S1"*** 는 Type Matching Mode이다. ***" Service_Port=p0"*** 는 Port Matching Mode이다. ***" Service_Type=S1 and (1.0 == prob(A=p and F=q))"*** 는 Exact Matching Mode이다. 이는 proximity의 특별한 경우로서 proximity threshold가 1.0인 경우이다.

## 4. 구 현

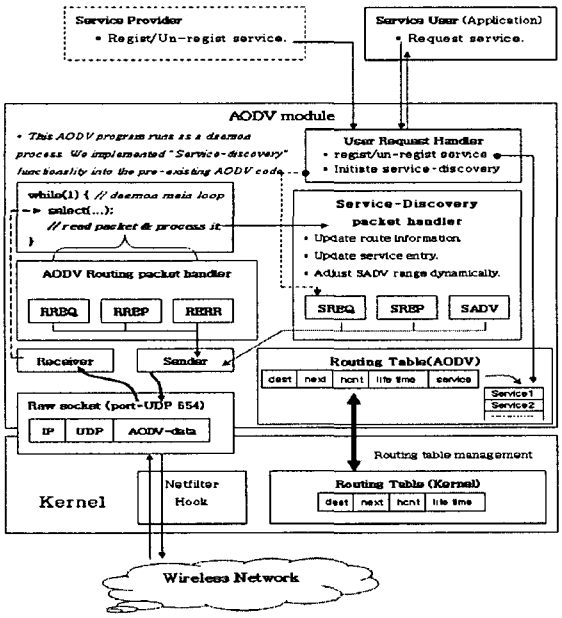
### 4.1 Service Discovery 기능을 추가한 AODV 데몬의 S/W구조

구현에 사용한 라우팅 데몬은 IPv4용 AODV-UU 0.8버전이다. Service Discovery기능을 수행하면서 동시에 route 정보들을 다루어야 하므로, 기존의 AODV를 확장하는 형태라고 이해하면 된다. AODV 데몬은 S/W적으로 볼 때, 비동기적으로 들어오는 외부 AODV패킷들을 처리해야 하므로 select()를 이용한 I/O Multiplexing 구조이다. 또한 route 정보들마다 설정된 expire time을 적용하기 위해서 하나의 타이머를 가지고 있다. 이는 큐 형태로 구성되어져 있어서 다수의 route 엔트리들에 대한 타임아웃 정보를 관리할 수 있게 된다. 또한, AODV 컨트롤 패킷을 주고 받을 때 raw socket을 사용함으로써 TTL과 같은 정보를 직접 액세스할 수 있고, 인터넷 게이트웨이를 구현하기 위한 tunneling의 목적으로 간단한 Encapsulation-프로토콜을 직접 운용할 수 있다.

이러한 기본적인 AODV를 확장하여 SREQ, SREP, SADV 패킷 핸들링 모듈을 작성하였다. 하지만 이러한 확장은 단순히 서비스관련 컨트롤 패킷을 운용하는 것만을 포함하지는 않는다. 즉, hybrid한 특성 때문에 서비스 정보가 라우팅 정보와 밀접한 관계를 가지게 되므로 AODV라우팅 테이블의 확장 및 이를 관리하는 메소드가 다수 추가 되었다. 특히 SREQ/SREP 패킷의 경우 서비스 관련 처리를 하기 이전에, 항상 RREQ/RREP 패킷을 받았을 때 수행하는 route 관리와 관련된 작업을 수행하도록 하였다. 실제로는 이러한 선행 작업에 의해서 route discovery 관련 flooding이 줄어든다.

또한 내부적으로는 서비스 자체를 다루기 위한 모듈로서 서비스 표현/파싱/매칭 모듈들이 포함 되었다. 또한, SD와 SP에게 API를 제공하기 위한 모듈도 포함한다. 이 모듈의 서

비스관련 기본 primitive로서 DEFINE, REGIST, UNREGIST, SHOW, QUERY(4가지 모드)등이 있다. 이중에서 서비스의 정의를 위한 DEFINE 기능은 보조적으로 XML파일을 작성하고, 이 파일을 배포하는 프로토콜을 위한 추가모듈이 필요하다. 하지만, 이 추가모듈은 현재 구현에서 제외하였고 포로토타입 형태 까지만 구현하였다. 이를 보완하기 위한 대체 모듈로서 테스트 및 검증을 수행하였다.




<그림 2> Service Discovery 기능이 추가된 AODV 데몬의 구조도

위의 구조도에서 좌측 및 아래쪽 kernel부분을 기존의 AODV의 모듈 및 자원이용 모습이라고 보면, 우측은 service discovery 기능을 위한 제반 모듈 및 기존 AODV수행 코드에 확장되어 붙어서 수행되는 모습을 나타낸 것이라고 보면 이해가 쉽다.

### 4.2 테스트 환경

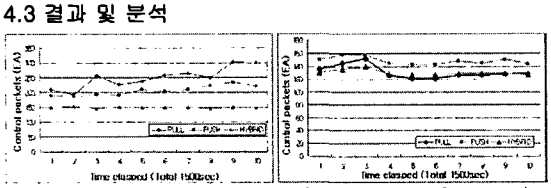
테스트는 다음과 같은 환경 하에 수행 하였다.

- 3-hop linear topology (총 4개의 노트북 노드)
- Linux kernel 2.4.24 / AODV-UU 0.8 (IPv4)
- 수행 비교 모델: ① **Push 모델**, ② **Pull 모델**, ③ **Hybrid 모델**
- 각 모델 별로 수행된 테스트 시간: 1500초

*(각 노드들은 MAC filtering을 이용해서 논리적으로 격리시켰다. 그러므로 propagation delay를 보상하기 위해 매 패킷의 전송마다 200ms의 휴면 시간을 두었다.)*

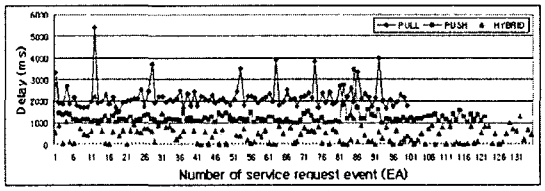
- 각각의 노드들이 분산/독립적으로 만들어내는 실시간 정보들(전송되어 나가는 control packet개수, SD측에서의 Service Discovery 수행 시 delay등)을 수집하기 위해서 하나의 collector 서버를 만들었다. 이 서버는 wired 망에 연결된 필자의 PC에서 수행 하였으며, Ad-hoc 노드들이 wired망에 있는 서버에 접근하기 위해서 Node A를 인터넷 게이트웨이로 구성하였다.

### 4.3 결과 및 분석




<그림 3> 컨트롤 패킷 오버헤드 비교 (좌측:λ =0.1, 우측:λ =0.01)

테스트를 진행하면서 외부로 전송되는 컨트롤 패킷의 개수를 150초마다 기록하여 결과를 도출하였다. λ 값은 사용자의 서비스 요청이 발생하는 시기를 결정하는 poisson분포 변수로서 사용된다. Hybrid 방식(" Service Discovery + Route Discovery" 는 물론, 동적/부분 Caching 기능까지 포함한 방식)은 λ =0.1일 때, PULL이나 PUSH 방식보다 더 적은 컨트롤 패킷이 발생됨을 알 수 있다. λ 가 매우 작으면 PULL이 항상 우위에 있음은 자명하다. 여기서 특이한 점은 PULL방식의 경우, 잦은 Service Discovery 요청이 있을 때, 오히려 PUSH 방식보다 더 많은 컨트롤 패킷을 만들어 내었다는 것이다. 이는 PULL방식에 의한 비용이 PUSH방식의 SADV컨트롤 패킷의 주기적 flooding 비용보다 더 커질 수 있음을 의미한다.




<그림 4> 지연 시간의 비교 (λ =0.1일 때)

앞의 <그림4>는 서비스 요청 이벤트 발생시, Service Discovery 과정에 소모되는 시간을 측정한 것이다. 사용자 입장에서 보면 자신이 어떠한 A라는 서비스를 찾아달라고 요청하고 나서부터 A에 대한 정보를 찾아오고, A라는 서비스가 존재하는 호스트로 가기 위한 경로까지 모두 찾을 때까지의 시간을 측정한 것이다. 즉, 사용자(SD)가 SP에 접속하거나 세션을 열기 바로 전까지의 지연 시간을 측정한 결과 그래프이다. 보다시피 지연 시간에서도 Hybrid 방식은 PUSH나 PULL방식에 비해서 훨씬 짧은 지연 시간을 보여 주고 있다. 이 결과, <그림 4>에서와 같이 동일한 시간(1500초)에 Hybrid방식이 PULL이나 PUSH보다 더 많은 서비스 요청을 처리하는 모습을 볼 수 있었다.

## 5. 결 론

결과에서 보듯이 논문에서 제시한 hybrid 모델이 가장 좋은 성능을 보여주었다. 비록 홉 수가 몇 개 되지 않았지만, 실제 구현 환경에서도 시뮬레이션에서와 같은 비교 우위의 결과를 보여 줌으로써 hybrid모델의 성능을 직접 입증할 수 있었다는데 의의가 있다고 볼 수 있다.

마지막으로 보다 동적인 Service Discovery 모듈 운용에 대해서 언급하고자 한다. 가령, Service Discovery에 참여하지 않고, 단지 라우팅에만 참여하고 싶은 노드의 경우도 있을 수 있다. 이 경우는 명시적으로 Service Discovery관련 기능을 수행하지 않도록 policy를 설정하면 된다. 이는 단순히 SREQ,SREP,SADV 패킷들을 처리하지 않으면 된다. 이 보다 좀더 나은 policy는 상황에 맞도록 스스로 Service Discovery 기능을 수행/정지 시키는 것이다. 이 경우, Network/System 자원의 모니터링을 통해서 선택적으로 Service Discovery 기능을 On/Off 시키도록 하는 방법이 있다. 자원이 충분할 때는 [라우팅+서비스] 둘 다 수행하고, 자원 상황이 일정 수준 이하로 떨어졌을 때는 [라우팅 only 모드]로 수행하도록 할 수 있다. 향후 연구 과제로는 Internet Gateway를 이용한 Wired/Ad-hoc망 간의 Service Discovery가 있다.

## 참고 문헌

[1] R. Koodli and C. E. Perkins, " Service Discovery in On-Demand Ad Hoc Networks," Internet draft, MANET Working Group,draft-koodli-manet-servicediscovery-00.txt,Sep.2002.

[2] C. Perkins and E. Belding-Royer, " Ad hoc On-Demand Distance Vector (AODV) Routing" , in RFC 3561, July 2003.

[3] Vikas Kawadia, et. al " System Service for Ad-Hoc Routing: Architecture, Implementation and Experiences" , MOBISYS, 2003.

[4] AODV-UU Resource, http:// user.it.uu.se/~henrikl/aodv/.