Christiano Farina Haesbaert

OpenMdns - Trabalho de Conclusão I

Porto Alegre - Rio Grande Do Sul, Brasil

22 de novembro de 2010

Christiano Farina Haesbaert

OpenMdns - Trabalho de Conclusão I

Orientador:
Ana Cristina Benso da Silva

Pontifícia Universidade Católica do Rio Grande Do Sul

Porto Alegre - Rio Grande Do Sul, Brasil

22 de novembro de 2010

Sumário

Lista de abreviaturas e siglas

Lista de Tabelas

Lista de Figuras

1 Introdução p. 7

2 Os Protocolos p. 11

2.1 Problema . . . . . p. 11

2.1.1 Endereçamento . . . . . p. 11

2.1.2 Network Browsing ou Network Discovery . . . . . p. 12

2.1.3 Cenários . . . . . p. 12

2.1.3.1 IP Dinâmico . . . . . p. 12

2.1.3.2 Músicas na Rede Local . . . . . p. 13

2.1.3.3 Storage Dinâmico . . . . . p. 14

2.2 MDNS . . . . . p. 14

2.2.1 Origem . . . . . p. 14

2.2.2 Introdução . . . . . p. 15

2.2.3 Domínio Multicast . . . . . p. 15

2.2.4 Resource Records . . . . . p. 16

2.2.5 Queries . . . . . p. 17

2.2.5.1 One-Shot . . . . . p. 18

2.2.5.2 One-Shot, Accumulating Multiple Responses . . . . . p. 19
2.2.5.3 Continuous Querying . . . . . p. 19
2.2.6 Cache . . . . . p. 20
2.2.7 Known Answer Suppression . . . . . p. 21
2.2.8 Probing & Announcing . . . . . p. 21
2.2.9 Resolução de Conflitos . . . . . p. 22
2.3 DNS-SD . . . . . p. 22

3 Projeto OpenMDNS p. 25
3.1 Introdução e Conceitos . . . . . p. 25
3.1.1 ZeroConf Networking . . . . . p. 25
3.1.2 OpenBSD . . . . . p. 27
3.1.3 Licenças . . . . . p. 28
3.1.4 Projeto . . . . . p. 30
3.1.4.1 Requisitos . . . . . p. 31
3.1.4.2 Arquitetura . . . . . p. 32
3.1.4.3 Status . . . . . p. 34
3.1.4.4 Feedback . . . . . p. 34

4 Conclusão p. 37

Referências p. 40

Lista de abreviaturas e siglas

IPv4	Internet Protocol version 4,	p. 6
GNU	GNU is Not UNIX,	p. 7
ISC	Internet Systems Consortium,	p. 8
DHCP	Dynamic Host Configuration Protocol,	p. 10
IPC	Inter Process Communication,	p. 30
IP	Internet Protocol,	p. 6
DNS	Domain Name System,	p. 6
TCP/IP	Internet Protocol Suite,	p. 6
MDNS	Multicast DNS,	p. 6
DNS-SD	DNS Service Discovery,	p. 6
IETF	Internet Engineering Task Force,	p. 7
FSF	Free Software Foundation,	p. 7
LGPL	GNU Lesser General Public License,	p. 7

Lista de Tabelas

1 Estado da Implementação das Seções do Draft MDNS(1) . . . . . p. 38

2 Estado da Implementação das Seções do Draft DNS-SD(2) . . . . . p. 38

Lista de Figuras

1 Arquitetura do OpenMDNS . . . . . p. 39

1 Introdução

Os dispositivos de redes estão cada vez menores e mais portáveis, e as redes de computadores estão presentes em praticamente todo lugar, de residências à grandes corporações. Gradualmente a idéia de que cada dispositivo é um elemento estático em uma rede préviamente configurada vem se tornando cada vez mais remota. Sendo assim, deseja-se comunicação entre dispositivos com a menor infra estrutura possível, em especial, em ambientes nos quais não há um contingente técnico, como por exemplo uma residência familiar.

Por exemplo, uma funcionalidade desejável em uma rede IPv4 (*Internet Protocol version 4*) é a de mapear *hostnames* em endereços IP (*Internet Protocol*) sem a necessidade de um servidor de DNS (*Domain Name System*) (3) local ter sido préviamente configurado nos *hosts* participantes da rede. Uma segunda funcionalidade que revela-se interessante é a de realizar enumeração de serviços (*Network Browsing*), para que um *host* possa pesquisar quais serviços são oferecidos na rede local. Estas são funcionalidades desejáveis em qualquer rede IPv4, seja em uma residência, escola ou corporação, portanto é importante tê-las com o mínimo esforço possível.

Pode-se enumerar então duas funcionalidades desejáveis na rede local:

1. Resolver nomes de DNS sem configuração prévia e/ou servidor central.
2. Realizar enumeração de serviços.

Ambas funcionalidades não são fornecidas pela suite TCP/IP (*Internet Protocol Suite*), e portanto foram propostos dois protocolos que geralmente operam em conjunto, MDNS (*Multicast DNS*)(1) e DNS-SD (*DNS Service Discovery*)(2).

O **MDNS** fornece a primeira funcionalidade, este é basicamente uma forma de realizar requisições de DNS via *multicast*, obtendo uma resposta também via *multicast*, neste cenário cada *host* é um agente de MDNS autonômo não existindo um servidor central como no DNS convencional.

O **DNS-SD** fornece a segunda funcionalidade e permite que através de requisições de DNS seja possível realizar a enumeração de serviços. Ele é compatível com DNS convencional (*unicast*) e MDNS, sua popularização no entanto, deu-se pelo uso em conjunto com o MDNS.

Os protocolos MDNS/DNS-SD geralmente operam como protocolos complementares, com eles é possível por exemplo perguntar quais impressoras estão *online* ou qual o endereço IP de um *host* qualquer, ou até mesmo perguntar quais computadores possuem um serviço de compartilhamento de músicas.

Apesar de ambos protocolos ainda possuirem estado de *Internet Drafts* junto a IETF (*Internet Engineering Task Force*), estes já possuem duas implementações bastante maduras e utilizadas atualmente:

- **Bonjour**(4), implementação original dos protocolos, criado pela empresa Apple, seu principal desenvolvedor, foi o próprio criador do protocolo Stuart Cheshire, é amplamente utilizado na maioria dos produtos da empresa e distribuído juntamente com o MACOS-X. É liberado sob a licença Apache-2(5) e funciona em diversos sistemas operacionais (inclusive no Windows), possui cerca de 60000 linhas de código.
- **Avahi**(6), alternativa fornecida pela FSF (*Free Software Foundation*), fornece compatibilidade binária com o Bonjour. É reconhecido como uma excelente implementação por Stuart Cheshire(criador do protocolo), funciona em diversos sistemas operacionais e possui cerca de 45000 linhas de código.

Apesar do estado maduro das implementações atuais, há três problemas significativos que atingem ambas:

1. *Tamanho*: ambas são implementações grandes, apesar da complexidade do problema, acreditamos que é possível desenvolver uma implementação menor, mais compacta e mais simples. Em outras palavras, acreditamos que ambos projetos sofrem de *bloat*[1].
2. *Complexidade*: acreditamos que ambas são projetos desnecessariamente complicados, embora isto só poderá ser confirmado ou não no término deste trabalho.
3. *Licença*: as licenças Apache-2(5) e LPGL (*GNU Lesser General Public License*), que licenciam o Bonjour e o Avahi respectivamente, possuem certas implicações que não são aceitas em alguns sistemas operacionais, como é o caso do OpenBSD(7).

[1] excesso de código devido a má codificação ou excesso de funcionalidades

Ao analisar as alternativas existentes conclui-se que nenhuma das duas era compatível com as características e filosofias do sistema operacional OpenBSD.

O OpenBSD é um sistema operacional UNIX multi-plataforma aberto e livre. podemos resumir e enumerar suas principais características e filosofias(7) como:

1. *Segurança*: mais que em outros sistemas o OpenBSD possue um forte foco em segurança, muitas vezes outros atributos, como *performance* são sacrificados em prol de segurança. É reconhecido hoje como possivelmente o sistema mais seguro do mundo.

2. *Licença*: é aceito apenas código com licenças BSD ou mais livre(8), o que não é o caso de nenhuma das implementações existentes.

3. *Código Aberto*: não são aceitos *blobs* ou qualquer outro tipo de código fechado, esta não é uma questão política e sim técnica, código fechado é igual a código não depurável e é visto como uma fonte de *bugs*.

4. *Simplicidade e Design*: muita atenção é dada para a simplicidade e o *design* do sistema, muitas coisas são re-escritas pelo simples fato de não serem "simples e bem desenhadas" o suficiente, é de crença do projeto que simplicidade, segurança, portabilidade e robustez andam em conjunto.

5. *Portabilidade*: o sistema roda em diversas plataformas, portanto grande atenção é dada à portabilidade do código, fato que colabora também para a *correteza* dos programas, visto que nem sempre, *bugs* se manifestam igualmente em diferentes arquiteturas.

6. *Documentação*: o sistema é declaradamente *developer-friendly*. O alvo são usuários avançados e desenvolvedores, isto também colabora para que o sistema seja extensivamente bem documentado, a falta de documentação ou documentações imprecisas são considerados como *bugs*.

Sendo assim, acredita-se que ambas as alternativas de MDNS/DNS-SD existentes não atendem as expectativas do sistema OpenBSD.

O objetivo deste trabalho é fornecer livremente à comunidade do OpenBSD uma implementação de MDNS/DNS-SD sob a licença ISC (*Internet Systems Consortium*)(9). Espera-se que futuramente, o fruto deste trabalho possa ser incorporado ao sistema base do OpenBSD, sendo assim instalado juntamente com o sistema.

Esta implementação deverá fornecer meios para que:

1. Um programa qualquer possa realizar consultas de MDNS.
2. O *host* possa responder as consultas de MDNS feitas na rede local.
3. Um programa qualquer possa anunciar seus serviços na rede local via MDNS/DNS-SD.

Os capítulos deste texto são organizados da seguinte forma:

1. *Introdução*, esboço do problema, cenário atual e motivação do trabalho.
2. *Os Protocolos*, apresentação e detalhamento dos protocolos envolvidos, assim como uma apresentação mais detalhada de quais problemas o protocolo tenta endereçar.
3. *Projeto OpenMDNS*, apresentação do projeto a ser realizado neste trabalho.
4. *Conclusão*, considerações finais.

2 Os Protocolos

Neste capítulo serão abordados os problemas que os protocolos MDNS e DNS-SD procuram endereçar, assim como o funcionamento dos protocolos em acordo com os *drafts* MDNS(1) e DNS-SD(2) respectivamente.

2.1 Problema

Na atual versão do IP (IPv4), a comunicação entre *hosts* em uma rede local necessita de intervenção manual significativa, esta que nem sempre é viável pois necessita ou de uma infra-estrutura préviamente configurada ou de usuários com nível técnico elevado. O protocolo IPv4 possui nenhuma ou poucas funcionalidades/facilidades de autoconfiguração, e é o desejo do MDNS/DNS-SD permitir que a comunicação em uma rede local IPv4 seja alcançada com menos configuração e maior automatização(1). A seguir são apresentados os pontos que estes dois protocolos tentam endereçar.

2.1.1 Endereçamento

O *Endereçamento* na rede IPv4 é numérico e não adequado para humanos, um IP do tipo: 192.168.8.1 não é algo interessante de se decorar.

O problema aumenta quando este endereço é dinâmico (obtido via DHCP), pois o que se sabia antes agora pode não ser verdade. A inclusão de DHCP também introduz um outro problema: é necessário que alguém tenha configurado préviamente o serviço de DHCP.

O problema do endereçamento numérico é mitigado com o DNS, que oferece uma maneira de mapear nomes em endereços, assim como outros tipos de registro, porém não se pode esperar que toda rede possua um servidor central de DNS préviamente configurado e preparado para aceitar requisições por *hostnames* locais.

Em um cenário ideal, cada *host* ao ser ligado teria um nome único sem nenhum tipo de configuração prévia ou servidor central, por exemplo:

- O *host frodo.local* foi ligado.
- Os outros *hosts* da rede podem por exemplo agora efetuar o comando:

  ping frodo.local

  E este responde, independentemente de seu endereçamento numérico. Ou seja, o usuário não mais depende do conhecimento do IP numérico, todo *host* passa a ser endereçado por um nome único.

2.1.2 Network Browsing ou Network Discovery

Network Browsing ou *Network Discovery* é a funcionalidade que permite que serviços sejam oferecidos dinâmicamente na rede, um exemplo clássico de serviço é o de uma impressora ou servidor de arquivos. Na rede IPv4 atual não há um mecanismo nativo para a publicação e resolução de serviços da rede local, o usuário ou depende do conhecimento de como acessar o serviço ou a funcionalidade de *Network Discovery* está embutida no serviço em questão, como por exemplo o NetBIOS(10).

Em um cenário ideal, o usuário ao ligar seu computador poderia acessar uma lista de serviços oferecidos na rede local, como por exemplo: impressoras, compartilhamento de músicas, compartilhamento de arquivos entre outros. Caso este usuário tenha também serviços em seu computador, existiria um mecanismo simples que permitiria que este publicasse o serviço na rede local, seja qual for a natureza do mesmo.

2.1.3 Cenários

A seguir são apresentados três cenários hipotéticos que se beneficiariam caso as funcionalidades descritas anteriormente estivessem presentes na rede local:

2.1.3.1 IP Dinâmico

O objetivo do usuário é acessar via SSH um computador que estava desligado, o usuário sabe que este computador obtém seu IP na rede local via DHCP e que seu nome é *frodo.local*, este computador não possue monitor e todo seu uso é feito remotamente. A

rede local não possui um servidor de DNS interno para responder pelos nomes dos *hosts* locais, o cenário seria come descrito a seguir:

1. Usuário liga o computador, espera alguns segundos até que este inicie.
2. Tenta acessar o computador pelo IP que ele *geralmente* assume, no caso 192.168.8.25.
3. O computador não responde, usuário assume que ele adquiriu outro IP, mas qual ? O fato de ele não saber, fará com que ele tenha que conectar um cabo serial no computador ou que acesse os *logs* do servidor de DHCP, nada garante que ele possua acesso ao mesmo.

No cenário ideal, o usuário não se importaria com o endereço, simplesmente faria o acesso pelo nome *frodo.local*, este que não teria que ter sido préviamente configurado em nenhum lugar.

2.1.3.2 Músicas na Rede Local

Usuário está no trabalho e gostaria de compartilhar suas músicas com seus colegas, portanto quer publicar um serviço do tipo *músicas* na rede local, a empresa é grande e não quer ter que avisar colega-a-colega que ele agora está disponibilizando suas músicas, o usuário quer estes possam (ao procurar por músicas na rede) visualizarem e usufruirem de suas músicas.

Para que isto seja possível no cenário atual, a única esperança do usuário é que o protocolo no qual ele está disponibilizando suas músicas possua algum mecanismo de *Network Browsing* embutido, isto é, que todo a inteligência da publicação e enumeração dos serviços de músicas seja um artefato do protocolo em si.

É fácil imaginar como isto rapidamente se torna inadequado, a medida que cada tipo de serviço diferente possuiria um protocolo *especifico* para prover *Network Browsing*. Por exemplo, o programa que compartilha músicas implementaria o seu protocolo, as impressoras implementariam outro, os servidores de arquivos outro e etc.

O importante é notar que o que se deseja é o *Network Browsing*, ou seja, a habilidade de se enumerar serviços de diferentes tipos, esta é uma funcionalidade que pode ser genérica e não dependente de cada serviço sendo oferecido. n

2.1.3.3 Storage Dinâmico

Uma empresa possue um grande *array* de discos em seu *storage* principal, estes discos estão localizados em diferentes dispositivos conectados por uma rede IP, a adição e remoção de discos do *array* é uma tarefa frequente, seja pela remoção de discos em falha ou pelo aumento do *array*.

Seria interessante uma forma de se monitorar o estado e a presença dos discos, ou seja, seria interessante que cada disco assim como seu estado atual (ok, em falha e etc) fosse publicado como um serviço, no caso, um serviço que expressa o estado de um disco do *array*.

Este monitoramento dos recursos instáveis (no caso discos, que são inseridos e removidos) poderia ser feito via MDNS/DNS-SD com *Network Browsing*. Poderiam também ser tomado atitudes sempre que um evento destes ocorre.

A idéia deste cenário foi motivado pelo *feedback* fornecido por Marco Peereboom descrito no final deste texto.

2.2 MDNS

2.2.1 Origem

O desenvolvimento do protocolo teve origem quando a empresa Apple realizou sua transição(29) de *layer 3* de AppleTalk(30) para IPv4. Uma funcionalidade presente no AppleTalk e ausente no IPv4 era a de *Network Browsing* ou *Network Discovery*, com isto era possível por exemplo, enumerar todos os *hosts* da rede assim como seus serviços, cada *host* era identificado por um nome, assim como seus serviços, esta *auto-configuração* era automática e de fácil utilização.

Na tentativa de não desenvolver um protocolo totalmente novo para o IPv4, os engenheiros Chesnire e Krochmal optaram por adaptar um protocolo conhecido, no caso o DNS. É importante perceber que o DNS não é apenas um protocolo de mapeamento entre *hostnames* e IPs, ele também pode ser visto como um banco de dados distribuído armazenando virtualmente qualquer tipo de informação.

O protocolo MDNS, como proposto por Stuart Cheshire encontra-se ainda em estado de *Internet Draft* não sendo portanto um *RFC*, porém já pode ser considerado como funcional tendo em vista que as implementações existentes são relativamente maduras.

2.2.2 Introdução

O *Multicast DNS* descrito no *draft* (1) é uma forma de se realizar *queries* de DNS via *multicast* no enlace local de forma que não exista um servidor central préviamente configurado. Os *hosts* da rede local cooperam entre si sem configuração prévia através de mensagens de multicast com o intuito de manter uma base de dados de DNS.

Esta base de dados pode possuir diversos tipos de registro (*Resource Records*)(3), cada uma com seu propósito específico, sendo o mais comum o *Resource Record* tipo *A*, que mapeia um *hostname* em um endereço IPv4. Assim como no DNS convencional, os mesmos *Resource Records* são utilizados e possuem o mesmo significado, isto reforça novamente a idéia de que o MDNS é apenas uma forma de se realizar *queries* de DNS via multicast.

A maioria dos usuários não possui um domínio em seu nome na Internet, tampouco possui a maioria dos *hosts* um IP roteável na Internet, geralmente os *hosts* estão em uma rede privada, tendo acesso a Internet através de um único roteador fornecendo acesso através de NAT, portanto mesmo que estes possuíssem um domínio para registrar seus endereços, o mesmo seria inválido pois teria sentido apenas na rede privada local. É evidente que é possível ter um servidor de DNS configurado na rede local e configurar todos os *hosts* para adicionarem seus registros de DNS neste mesmo servidor, porém esta infraestrutura é cara para a maioria das redes e exige configuração não trivial, sendo inviável para a grande maioria dos usuários. O MDNS fornece esta funcionalidade de forma automática e não exige nenhum tipo de configuração, os *hosts* são livres para adquirirem um nome único válido apenas para o enlace local. Os usuário podem acessar os *hosts* da rede pelos seus nomes sem nenhum tipo de configuração prévia.

No capítulo seguinte será visto que esta automatização pode ser extendida para os serviços oferecidos de cada *host*, e este mecanismo será discutido mais detalhadamente quando for abordado o protocolo DNS-SD.

2.2.3 Domínio Multicast

O MDNS é um protocolo atuante apenas no enlace local, seu grupo *multicast* 224.0.0.251 está na faixa denominada pela IANA de *Link-Local* não sendo, portanto, roteado entre redes IP. O protocolo atua sob a porta UDP 5353, diferentemente do DNS unicast, não existem diversos domínios, portanto também não há hierarquia entre os mesmos, há apenas um domínio, o *.local*. Sendo este o domínio *multicast* e possuidor de semântica es-

pecial. Portanto nenhum *host* é detentor de domínio algum, no entanto, os *hosts* podem ser detentores de nomes únicos, estes que serão abordados ao longo do texto. Quando um *host* que suporta MDNS realiza uma *query* para o domínio *.local*, esta *query* deve ser feita via MDNS, pois este é o domínio *multicast*. Um *host* implementando apenas DNS convencional enviaria tal *query* para seu servidor de DNS préviamente configurado, provavelmente não retornando alguma resposta.

É importante ressaltar que o MDNS pode ser utilizado para resolver nomes globais (como no DNS *unicast*), para isto, todas as requisições de DNS, independente do domínio precisam ser enviadas via *multicast*. Neste caso, pelo menos um *host* no enlace local precisa fazer a ponte entre o DNS *unicast* (consultando os *Root Servers*) e o MDNS. Este não é o funcionamento natural e seu tratamento é melhor descrito no *draft*(1).

2.2.4 Resource Records

Outra diferença bastante significativa para o DNS convencional é a classificação dos *Resource Records* em dois tipos:

- *Unique*: *Resource Record* onde a tripla DNAME/TYPE/CLASS(3) é única e apenas um *host* responde por este *Resource Record*.
- *Shared*: *Resource Record* onde uma *query* pode elicitar múltiplas respostas, cada *host* pode responder por um ou mais *Resource Record* do tipo *shared*.

Um *Resource Record unique* é desejável por exemplo no mapeamento de *nome IP*, deseja-se que uma *query* pelo nome *foobar.local* do tipo *A* (IPv4 Address) retorne apenas uma resposta, seria estranho se um nome mapeasse para mais de um endereço, outro exemplo bastante comum é o *Resource Record* de tipo *HINFO* (Host Information), usado para mapear um nome para um par de *strings* CPU/Arquitetura.

Para que um *host* possua um *Resource Record unique*, ele precisa primeiro verificar a unicidade do nome em questão, para isto ele precisa realizar uma etapa chamada de *Probing*, em que um *host* certifica-se que o nome em questão não está sendo utilizado por nenhum outro *host*. Depois deste passo, o *host* pode anunciar quaisquer *Resource Record* sob este nome, podendo por exemplo anunciar ambos os *Resource Record* de tipo *A* e *HINFO*. A etapa de *Probing* será abordada em mais detalhe ao longo do texto.

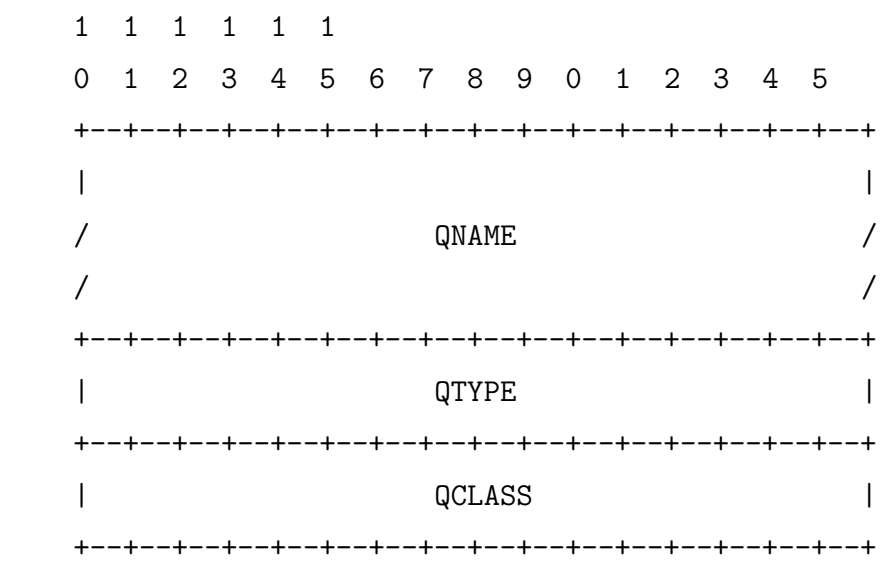
No caso do *Resource Record* do tipo *shared*, não é verdade que um *host* é o único possuidor do mesmo, isto é, cada *host* pode possuir zero ou mais instâncias do mesmo,

cada um com seus respectivos RDATA(3). Este tem sido utilizado apenas para o DNS-SD, onde por exemplo pode-se requisitar uma lista de todos os servidores de *HTTP*.

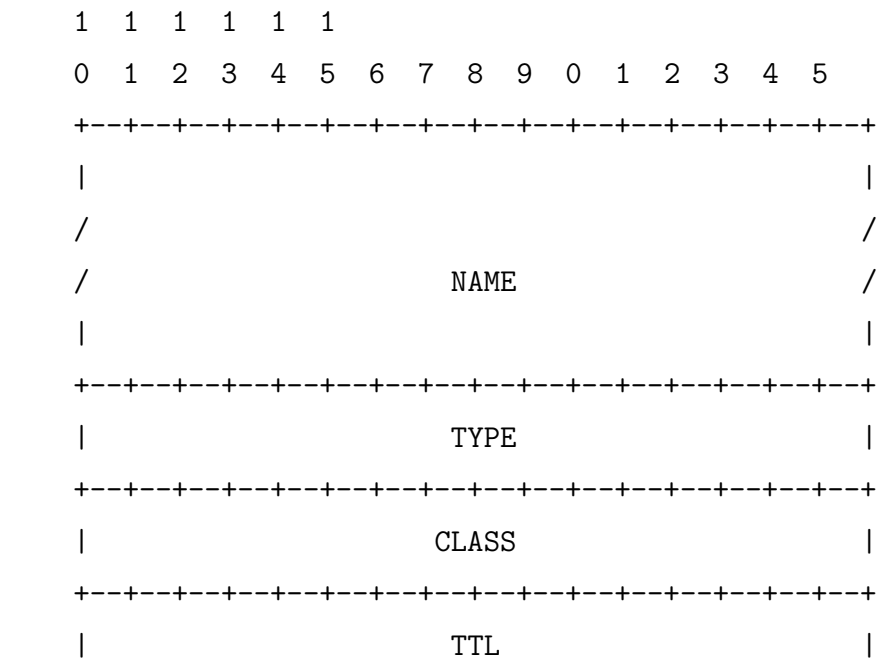
2.2.5 Queries

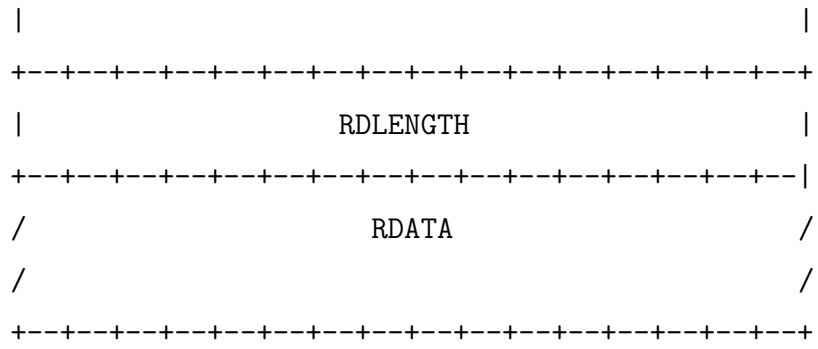
No MDNS existem três tipos de *query*, estas, assim como suas respectivas respostas possuem o mesmo formato do DNS *unicast* convencional como descrito no RFC 1035(3):

Question contida em uma Query



Resource Record contido em uma resposta (Answer).





Apesar do mesmo formato, um *Querier* de MDNS deve sempre enviar suas requisições com a porta origem UDP 5353, ao contrário dos cliente *unicast* onde a porta origem é efemera.

2.2.5.1 One-Shot

Neste tipo é enviado apenas uma uma *query* e esperado apenas uma resposta, caso sejam elicitadas múltiplas respostas, o cliente leva em conta apenas a primeira, desconsiderando as outras.

Este é o único tipo de *query* que pode ser utilizado por um cliente DNS *unicast* convencional, esta *query* é chamada de *Legacy Query* pois pode ser feita por clientes legados (cliente DNS convencional). Todas as *queries* no MDNS devem ser feitas com a porta origem UDP 5353, como os clientes legados utilizam uma porta origem UDP efêmera, um MDNS *Responder* é capaz de identificar tal condição e enviar uma resposta *unicast* ao invés da resposta *multicast* padrão, desta forma, o cliente consegue processar a resposta, coisa que não conseguiria caso esta tivesse sido enviada para o endereço *multicast*.

O suporte a *Legacy Queries* é interessante a medida que permite uma certa interoperabilidade com clientes legado com o mínimo de esforço.

A seguir é apresentado um exemplo de *One-Shot query* convencional (não legada), supõe-se que o *host bilbo.local* deseja descobrir o endereço IPv4 de *frodo.local*:

1. O *host bilbo.local* envia uma *query* com uma *question* com os seguintes atributos {QNAME=*frodo.local*, QTYPE=A, QCLASS=IN}, ele decide esperar por uma resposta em até 3 segundos, caso contrário *bilbo.local* assume que não existe o *Resource Record* desejado na rede.

2. O *host frodo.local* recebe a *query*, analiza a porta origem e determina que esta irá

elicitar uma resposta *multicast* visto que a porta origem é a porta MDNS (5353), *frodo.local* verifica que possui o *Resource Record* em questão e envia uma resposta com os seguintes atributos {NAME=*frodo.local*, TYPE=A, CLASS=IN, TTL=120, RDLENGTH=4, RDATA=192.168.8.21}.

3. A resposta chega no *host bilbo.local* antes de seu timeout de 3 segundos, este processa a resposta e dá como encerrado a requisição.

O atributo TTL escolhido como 120 segundos na resposta da *query* é o recomendado no *draft* para *Resource Record* de tipo A, ele especifica que tal asserção é válida por 120 segundos, todos os *hosts* que receberam a resposta devem armazenar em seu *cache* o *Resource Record* por 120 segundos. Sempre que um *host* considera realizar uma *query* ele primeiro consulta seu *cache* a fim de evitar um tráfego desnecessário na rede. No caso de uma *One-Shot query* se a informação já existe no *cache*, é suficiente para que nenhuma *query* seja enviada.

O RDLENGTH corresponde ao tamanho de um A record (4 bytes) e o RDATA é endereço IP em questão(3).

2.2.5.2 One-Shot, Accumulating Multiple Responses

Este tipo de *query* é semelhante ao anterior, porém, o cliente aguarda alguns segundos e acumula possíveis múltiplas respostas. É importante notar que só haverão múltiplas respostas caso o *Resource Record* seja *shared*, pois do contrário só pode existir necessariamente um *Resource Record*.

Este tipo de *query* é raramente utilizada e só faz sentido no uso de DNS-SD, sua funcionalidade é suprida pelo tipo *Continuous Querying* descrito a seguir, sendo assim não será abordado em detalhes.

2.2.5.3 Continuous Querying

São enviadas *queries* com intervalos crescentes, no *draft* é recomendado que se dobre o interavalo a cada *query*. Sendo o primeiro intervalo em 1 segundo. Todas as respostas recebidas são acumuladas assim como no tipo anterior, este tipo de *query* é relevante quando se deseja (no DNS-SD) enumerar dos os serviços disponibilizados na rede de um determinado tipo.

A seguir um exemplo onde o *host bilbo.local* deseja enumerar todos os servidores de NTP da rede local:

1. O *host bilbo.local* envia uma *query* com os seguintes atributos {QNAME=_ntp._udp.local, QTYPE=PTR, QCLASS=IN}. Ele irá reenviar esta *query* dobrando o intervalo de tempo a cada envio, sendo o primeiro intervalo de 1 segundo e considerando T0 como o tempo inicial, será enviada uma *query* em T0, T0 + 1, T0 + 2, T0 + 4, T0 + 8 até o limite de intervalo, no caso, uma hora.

2. Na rede local existem 2 serviços de NTP, um no *host gandalf.local* e outro no *host frodo.local*, os nomes do serviço NTP são *The Valar Clock NTP Server* e *The Shire Clock NTP Server* respectivamente. Ao receber a *query* enviada por *bilbo.local* *gandalf.local* responde com {NAME=_ntp._udp.local, TYPE=PTR, CLASS=IN, RDATA=*The Valar Clock NTP Server*, ...} e o *host frodo.local* com {NAME=_ntp._udp.local, TYPE=PTR, CLASS=IN, RDATA=*The Shire Clock NTP Server*, ...}. O importante é notar que são elicitadas duas respostas do tipo PTR, e eles apontam para o nome de seus respectivos serviços, o funcionamento da enumeração e resolução de serviços será melhor detalhado no capítulo que aborda o protocolo DNS-SD.

3. *bilbo.local* recebe as duas respostas e agora sabe que existem dois serviços do tipo NTP na rede. A razão para o envio de múltiplas *queries* deve-se ao fato de que o pacote pode não ter atingido todos os *hosts* da rede, um *switch* pode ter descartado o pacote devido à sobrecarga, ou pode ter ocorrido uma colisão no caso de uma rede *half-duplex*, os envios subsequentes diminuem a chance de falha.

2.2.6 Cache

Outro aspecto importante do MDNS é a necessidade de se manter o estado de *queries* passadas, ao contrário do cliente *unicast* padrão (toma-se de exemplo a implementação da LIBC do OpenBSD), onde é possível a cada chamada de *getaddrinfo(3)*(11) ou *gethostbyname(3)*(12) criar um novo *socket* enviar a requisição e aguardar por um *timeout*.

No MDNS o estado das *queries* passadas deve ser armazenados em um *cache*, este que deve ser consultado por futuras respostas. A simples presença de um *cache* complica bastante o protocolo como um todo, ele adiciona diversos *timers* e um protocolo de coêrencia de *cache*.

Cada *Resource Record* possui um TTL (*Time To Live*) que indica por quantos segundos aquela entrada deve ser considerada válida, no caso de *shared records* o *host*

realizando a *query* primeiro consulta seu *cache* por possíveis respostas e então envia sua *query*, junto com a *query* são incluidas todas as respostas previamente conhecidas, como descrito adiante na seção *Known Answer Supression*.

Este *cache* deve conter apenas os *Resource Records* propagados pela rede, ou seja, os registros vistos pelo *host* ao longo do tempo, e devem ser deletados do mesmo a medida que seu TTL expira.

O *draft* recomenda que um *host* tente renovar seus registros quando estes estão perto de expirar (ao término do TTL), para isto, é recomendado que se realize uma *query* pelo *Resource Record* nos 80%, 90% e 95% do tempo de vida, esta renovação deve ser evitada caso não exista interesse no *Resource Record*.

Os requisitos completos para coerência de *cache* e mecanismos para evitar um tráfego desnecessário são bastante complexos e serão abordados ao longo do texto quando este for conveniente.

2.2.7 Known Answer Suppression

O *Known Answer Suppression* (KNA) é um mecanismo para evitar a elicitação de respostas a *queries* nas quais já se sabe algumas das respostas, aplicando-se então apenas à apenas a *queries* feitas por *shared Resource Records*.

Um *host* ao realizar uma *query* das quais já possui algumas respostas, deve incluir estas na *Answer Section*(3) do pacote, um outro *host* ao receber a *query* não responderá a mesma se as respostas que seriam feitas já estão listadas no *Answer Section* da *query*.

Esta funcionalidade é bastante útil quando se realiza *queries* do tipo *Continuous Query*, onde uma *query* é enviada de tempos em tempos, seria um tráfego desnecessário e inconveniente se a cada re-envio fossem enviadas as mesmas respostas elicitadas a pouco.

2.2.8 Probing & Announcing

Quando um *host* deseja publicar um de seus *Resource Record* na rede, caso este seja do tipo *unique* ele precisa primeiro realizar uma etapa denominada de *Probing*, caso o *Resource Record* seja *shared* esta etapa pode ser ignorada e seguir para outra etapa denominada de *Announcing*.

No *Probing* o *host* adquire um nome único para um conjunto de *Resource Records* nos quais ele deseja públicar. Para que isto ocorra, o *host* primeiro verifica que nenhum

outro *host* da rede é detentor do nome desejado, podendo então partir para a etapa de *Announcing*.

A seguir é apresentado um exemplo onde o *host frodo.local* deseja públicar um conjunto de *Resource Records* sob o nome de *frodo.local*, mais especificamente ele deseja públicar: Um registro do tipo *HINFO*, outro do tipo *A* e por fim um do tipo *PTR* que será utilizado para o mapeamento reverso entre *hostname* -} IP.

2.2.9 Resolução de Conflitos

Um conflito surge quando dois *hosts* desejam publicar algum *Resource Record* único sob o mesmo nome. Quando um *host* recebe uma resposta com um *Resource Record* conflitante, este deve reiniciar o processo de *Probing*, o algoritimo de resolução de conflitos é então acionado na fase de *Probing* como descrito no *draft*(1).

Ao fim da resolução de conflito, o *host* perdedor deve escolher um outro nome para o seu *Resource Record* e reiniciar o processo de *Probing*, caso exista um novo conflito os passos são repetidos.

2.3 DNS-SD

O *DNS Service Discovery* descrito no *draft*(2)é um protocolo baseado no DNS que fornece a funcionalidade de *Network Discovery* ou *Network Browsing*. Apesar de o DNS-SD ser compatível com o DNS convencional (*unicast*), sua popularidade e uso deu-se em conjunto com o MDNS, na verdade o DNS-SD foi projetado em paralelo ao MDNS e pelos mesmos criadores.

O protocolo consiste em uma convenção de nomes e registros de DNS que devem ser utilizados de forma que seja possível realizar duas operações básicas:

1. *Enumeração de Serviços*
2. *Resolução de Serviços*

Na **Enumeração de Serviços**[1] utiliza-se *Resource Records* do tipo *PTR*(3) de forma análoga aos diretórios de um disco rígido. Para cada serviço da rede local é publicado um registro do tipo *PTR* que aponta para a instância do serviço em questão, o nome do

[1] também chamado de *browsing*

registro *PTR* é o protocolo do serviço. supõe-se que existam dois serviços do protocolo HTTP, portanto haveria dois registros do tipo *PTR* na seguinte disposição:

```
_http._tcp.local --> paginaslegais._http._tcp.local
_http._tcp.local --> paginaschatas._http._tcp.local
```

Pode ser feita uma analogia com a organização dos arquivos em um diretório, onde o diretório seria o protocolo de serviço (*_http._tcp.local*) e os arquivos as instâncias do serviço.

É importante perceber que uma *query* pelo nome do protocolo de serviço elicita múltiplas respostas, este é o caso clássico de *shared Resource Records* utilizados no MDNS.

Na enumeração de serviços há também um nome especial, no caso o *_services._dns-sd._udp.local*, este funciona como um diretório superior ao dos protocolos de serviço e fornece uma lista dos protocolos de serviço em questão, portanto, é possível fazer uma enumeração recursiva da seguinte forma:

1. Faz-se uma *query* do tipo *PTR* pelo nome *_services._dns-sd._udp.local*. O resultado serão diversos *Resource Records* que apontam para os nomes de protocolo de serviço.

2. Faz-se então uma nova *query* do tipo *PTR* para cada protocolo de serviço recebido na *query* anterior.

A partir da *Enumeração de Serviços* é possível se obter portanto, uma lista dos serviços ativos, com isso, é possível que se realize a **Resolução de Serviços**. Para que isto ocorra, cada serviço publica, em adição ao registro *PTR*, mais dois registros adicionais: *TXT*(3) e *SRV*(3).

Estes registros são publicados sob o nome no qual o registro *PTR* aponta (o nome do serviço em questão), no caso do exemplo anterior o nome de um dos serviços seria *paginaslegais._http._tcp.local*. Portanto, para realizar a resolução do serviço *paginaslegais._http._tcp.local* seria feita uma requisição pelos registros de tipo *SRV* e *TXT*.

O registro de tipo *SRV* especifica os seguintes atributos:

- *Priority e Weight*, a prioridade e peso do serviço, isto possibilita que seja possível publicar diversos serviços semelhantes e escolher apenas o com maior prioridade (menor valor). É pouco ou quase não utilizado no MDNS.

- *Port*, a porta do serviço em questão, no caso do HTTP, provavelmente seria 80. O protocolo de transporte está implícito no nome, no caso TCP para *paginaslegais._http._tcp.local.*

- *Target*, o nome do *host* que detém o serviço, isto possibilita que um *host* publique serviços que estão situados em outros *hosts*.

O registro de tipo *TXT* especifica atributos adicionais dependentes do protocolo do serviço em questão, no caso do HTTP poderia ser por exemplo um usuário de *login*, a versão do protocolo HTTP sendo utilizada, ou qualquer outra informação pertinente ao protocolo HTTP. Esta informação é dada sob uma lista de tuplas que mapeiam *chave:valor*.

3 Projeto OpenMDNS

O nome escolhido para o projeto foi OpenMDNS seguindo a linha de outros projetos parceiros do OpenBSD, como OpenBGP(13), OpenOSPF(14) e OpenSSH(15).

Neste capítulo será abordado o projeto do OpenMDNS assim como explicado e apresentado alguns conceitos importantes para a compreensão do projeto, tais como *ZeroConf*, Licenças, OpenBSD e um breve resumo das implementações existentes. Logo em seguida será discutido os requisitos do projeto assim como a arquitetura proposta.

3.1 Introdução e Conceitos

Neste trabalho é proposto o desenvolvimento de uma implementação dos protocolos MDNS/DNS-SD para o sistema operacional OpenBSD. Atualmente o sistema possui um *port*[1] do Avahi, e com algum esforço seria possível portar o Bonjour para que este funcionasse no sistema, porém diversos são os fatores que motivam a criação de uma nova implementação.

3.1.1 ZeroConf Networking

ZeroConf é o nome dado a uma série de técnicas e funcionalidades que fornecem um maior nível de auto-configuração para uma rede IPv4, estas técnicas permitem que os usuários finais possam inter-conectar seus computadores e seus dispositivos de rede, tais como impressoras e afins. Sem ZeroConf, uma rede IPv4 necessita de considerável intervenção manual, para que uma rede IPv4 possa operar razoavelmente bem, sendo necessários diversos serviços, os quais exigem uma configuração não não trivial, como por exemplo:

- DHCP: sem um servidor de DHCP na rede, os usuários devem configurar seus IPs

[1] Distribuicao de um programa UNIX

manualmente, ou o fardo fica pela configuração do servidor ou pela configuração de cada dispositivo.

- DNS: para que um computador possa endereçar o outro em uma rede local por um nome simbólico (*hostname*), é necessário que seja configurado um servidor de DNS local, o que também não é nada trivial. Sem isto, cada usuário precisa comunicar seu endereço IP aos seus próximos que desejam utilizar algum tipo de serviço de seu computador.

- Impressoras e afins: a configuração de recursos como impressoras em uma rede IPv4 é bastante trabalhoso, existem muitas variáveis envolvidas, tais como: o endereço da impressora, o driver, o protocolo de comunicação utilizado e etc. Fato que resulta na maioria dos usuários usarem apenas as impressoras de suas casa, estas que foram provavelmente préviamente configuradas por um usuário mais técnico.

O ZeroConf tenta abordar as questões acima com três mecanismos distintos:

1. *Link-Local Address Autoconfiguration*: protocolo bastante simples que permite um *host*, assumir um IP reservado qualquer na rede sem que seja necessário nenhum tipo de configuração. A seguir é apresentado o funcionamento básico deste protocolo:

   (a) Seleciona um IP randômico em um range reservado pela IANA para ser usado em redes locais.

   (b) Verifica através de *ARP Requests* que nenhum outro *host* responde por este endereço, caso sim, volta para *a* e seleciona outro endereço.

   É importante ressaltar que no RFC 3927 é um requisito que este protocolo só entre em funcionamento após uma falha confirmada da obtenção de um IP via DHCP, portanto este é um protocolo que funciona sem nenhuma configuração prévia (sem servidor de DHCP).

2. Resolução de *hostnames* automático na rede local, ou seja, fornecer as funcionalidades tradicionais do DNS, porém sem nenhuma configuração prévia. Esta funcionalidade é fornecida pelo MDNS.

3. *Network Discovery/Network Browsing*: possibilitar a realização da enumeração de serviços e recursos na rede, sem que seja necessário uma prévia configuração do mesmo. O exemplo clássico é de uma impressora. No caso ideal, o usuário ligaria seu computador e automaticamente teria acesso a todas as impressoras da rede, assim

como todos os outros serviços fornecidos pelos computadores da rede, tais como: servidores de arquivo (NFS/CIFS/FTP...) ou sincronizadores de relógio (NTP) e etc. Esta funcionalidade é fornecida pelo DNS-SD.

O ZeroConf é, portanto, a união destes três mecanismos. Neste trabalho propõem-se tratar dos últimos dois, o MDNS e o DNS-SD. O suporte ao primeiro foi rejeitado pois este seria desconexo do restante do trabalho, apesar de eles fornecerem uma suite de serviço, estes não tem muito em comum. No futuro deseja-se implementar a primeira funcionalidade no cliente *dhclient*(16) do OpenBSD.

3.1.2 OpenBSD

O OpenBSD é um sistema operacional UNIX multi-plataforma oriundo de um *fork(2)*(17) do NetBSD(18) em 1995 feito pelo seu fundador Theo De Raadt. O sistema é um direto descendente do BSD 4.4 desenvolvido em Berkeley na Califórnia, uma das variantes mais populares do UNIX de 1968.

O sistema é inteiramente grátis e o código é aberto, sendo licenciado em maior parte sob as licenças BSD e ISC e não possui fins lucrativos, todo o desenvolvimento é feito por voluntários e entusiastas. O projeto se mantém através da venda de CDs (estes que também são disponibilizados gratuitamente), o comprador paga na verdade pelos diversos adesivos e pelo encarte do CD, este dinheiro é então utilizado para financiar eventos e o custo dos desenvolvedores oficiais do projeto, tais como as passagens para os eventos e etc.

Como advertido no site(7) do projeto os ideais do projeto são:

> *Our efforts emphasize portability, standardization, correctness, proactive security and integrated cryptography.*

Todo o código do sistema é extensivamente auditado por falhas mesmo que mínimas ou até mesmo de documentação, todo código deve ser portável e correto, mesmo que isto ofereça um impacto na performance do mesmo. O OpenBSD é reconhecido hoje como possivelmente o sistema operacional mais seguro no mundo, e seu fundador, Theo De Raadt é tido como um dos maiores *experts* de segurança no mundo. Outro ponto bastante relevante do projeto é sua seriedade quanto a documentação, mais que em outros sistemas, é obrigação do sistema possuir sempre uma documentação acessível e atualizada.

Os desenvolvedores do OpenBSD presam por implementações compactas e seguras, rejeitando aplicações desnecessáriamente grandes e poluídas com funcionalidades não expressivas. É da crença do autor que tanto Avahi e Bonjour se enquadram nestas definições, tendo o primeiro cerca de 60 mil linhas de código e o segundo algo perto de 50 mil linhas.

O OpenBSD trouxe algumas implementações utilizadas no mundo inteiro, tais como o OpenSSH, PF, OpenBGPD, OpenOSPFD, OpenRIPD. Todas estas partilham dos princípios descritos acima.

3.1.3 Licenças

O OpenBSD é bastante restrito quanto as licenças do código presente no sistema base, para que isto seja compreendido é precisa-se entender como é feita a divisão entre sistema base e *third-party*(aplicativos disponibilizados via *ports* ou pacotes).

O sistema base(ou apenas base) compreende as aplicações base do sistema, desde os programas e utilitários em espaço de usuário ao *kernel* e *drivers*. É o sistema base que define na verdade o que é o OpenBSD, toda a base é auditada e mantida em grande parte pelos desenvolvedores do OpenBSD, é apenas na base que atuam as restrições de qualidade e política (licenças). É política do OpenBSD aceitar apenas código na base que possuam licença equivalente ou mais-livre-que a licença BSD(salvo raras exceções(8)), sendo a licença ISC a preferida em código novo.

Os *ports (third-party)*(19) são pacotes disponibilizados de diversas fontes diferentes, ao contrário do sistema base, eles não possuem nenhum tipo de restrição, pois estes não fazem parte do OpenBSD, são mantidos por diversas pessoas ao redor do mundo e não possuem vínculo algum com as filosofias e políticas do OpenBSD. O ponto importante a ser compreendido é que o *ports* é apenas um mecanismo com o fim de facilitar a instalação de programas de terceiros.

O Avahi possui licença LGPL e o Bonjour possui licença Apache-2, ambas não são aceitas no OpenBSD, sendo assim, o fruto deste trabalho deverá ser livremente distribuído sob a licença ISC, pois espera-se que este possa, ao término de sua implementação, ser aceito no sistema base do OpenBSD.

A seguir é apresentada uma cópia da licença BSD e ISC, em 22 de Julho de 1999 a Universidade de Berkeley recindiu a terceira cláusula da licença BSD.

Licença BSD original (4 cláusulas)

```
* Copyright (c) 1982, 1986, 1990, 1991, 1993
* The Regents of the University of California.  All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:
* 1. Redistributions of source code must retain the above copyright
*    notice, this list of conditions and the following disclaimer.
* 2. Redistributions in binary form must reproduce the above copyright
*    notice, this list of conditions and the following disclaimer in the
*    documentation and/or other materials provided with the distribution.
* 3. All advertising materials mentioning features or use of this software
*    must display the following acknowledgement:
* This product includes software developed by the University of
* California, Berkeley and its contributors.
* 4. Neither the name of the University nor the names of its contributors
*    may be used to endorse or promote products derived from this software
*    without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ‘‘AS IS’’ AND
* ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE
* FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
* DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
* LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
* OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
```

```
* SUCH DAMAGE.
*
```

Licença ISC

```
/*
* Copyright (c) CCYY YOUR NAME HERE <user@your.dom.ain>
*
* Permission to use, copy, modify, and distribute this software for any
* purpose with or without fee is hereby granted, provided that the above
* copyright notice and this permission notice appear in all copies.
*
* THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
* WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
* MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
* ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
* WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
* ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
* OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
*/
```

3.1.4 Projeto

O objetivo deste trabalho é fornecer uma implementação de MDNS/DNS-SD para o OpenBSD, para que isto ocorra, deverá ser implementada as funcionalidades pendentes do OpenMDNS, este que teve início o seu desenvolvimento em Fevereiro de 2010.

Esta implementação será distribuida livremente para a comunidade e usuários do sistema OpenBSD, sendo do desejo do autor que qualquer pessoa possa se beneficiar deste trabalho como bem entender e por isso foi escolhida a licença ISC que impoem nenhuma ou quase nenhuma restrição ao uso do *software*.

Pode-se enumerar as funcionalidades básicas da implementação como:

1. Fornecer meios para que um programa possa realizar consultas de MDNS, ou seja, deve existir alguma biblioteca ou integração com o sistema de modo que os aplicativos existentes se beneficiem do mesmo.

2. Fornecer meios para que o *host* responda as requisições de MDNS na rede, ou seja, a implementação deve responder a requisições MDNS de outros *hosts*.

3. Fornecer meios para que um programa possa publicar seus serviços via MDNS/DNS-SD, ou seja, cada programa, caso seja de seu agrado, deve ter a opção de públicar algum serviço.

3.1.4.1 Requisitos

A seguir são apresentados os requisitos que foram elaborados para atender a implementação. Estes requisitos surgiram a partir do estudo das implementações atuais, das características e expectativas do sistema OpenBSD, assim como da experiência do autor.

- *Tamanho*: a implementação deve ser menor e mais compacta que as existentes, logo espera-se atingir este requisito com um *design* adequado e com a não inclusão de funcionalidades supérfluas.

- *Simplicidade*: o código deve ser o mais simples possível, pois a simplicidade deve ser favorecida sempre que possível, otimizações de performance só deverão ser consideradas quando provado que a solução mais simples não é suficiente.

- *Portabilidade*: a implementação deve funcionar em todas as arquiteturas suportadas pelo sistema OpenBSD. O autor não dispoem de acesso a todas as arquiteturas, portanto, testes serão feitos apenas nas arquiteturas que estão disponíveis, são elas: *i386, amd64, alpha, sparc64*. Espera-se que a comunidade realize testes nas arquiteturas restantes. O suporte a múltiplas arquiteturas colabora para a localização de *bugs*, assim como garante que o código está correto quanto a questoes como alinhamento e *endianidade*.

- *Múltiplas Interfaces*: a implementação deve levar em conta o caso de um *host multi-homed*[2], para isto, deve haver suporte existir suporte para múltiplas interfaces de rede, deve existir uma opção para habilitar e desabilitar estas interfaces.

- *Interoperabilidade*: deve existir interoperabilidade com Avahi e Bonjour, para que isto ocorra deve-se atentar para a correta interpretação dos *drafts*.

- *Ferramenta de debug*: deve existir uma ferramenta básica de *debug* que permite operações simples como realizar consultas, publicar serviços e observar o estado interno do *software*.

[2]Um host com duas ou mais interfaces de rede

- *Integração com LIBC*: deve existir uma opção para habilitar a resolução de nomes do domínio *.local* via MDNS, para isto é preciso integração com as chamadas da biblioteca *gethostbyname(3)(12)*, e *getaddrinfo(3)(11)*. Isto fará com que qualquer programa possa se beneficiar automaticamente da resolução de nomes via *multicast*.

- *Segurança*: o código deve condizer com as políticas de segurança do sistema OpenBSD, tais como separação de privilégio(20) e *chroot*(20).

- *Dependências*: a implementação não deve depender de nenhuma biblioteca ou ferramenta que não esteja presente na base do sistema OpenBSD, fazendo com que caso o OpenMDNS seja aceito um dia como parte integrante do sistema, esta integração seja possível.

3.1.4.2 Arquitetura

Na *Figura 1* é apresentado um esboço da arquitetura do OpenMDNS, esta que é composta por três componentes base:

1. **mdnsd**
2. **libmdns**
3. **mdnsctl**

O **mdnsd** é um *daemon*, portanto um programa em espaço de usuário sem terminal controlador, responsável por toda a implementação dos protocolos MDNS/DNS-SD, ele responde as requisições de MDNS oriundas da rede local assim como efetua todas as requisições feitas pelos outros programas do *host* local. As responsabilidades do **mdnsd** podem ser melhor visualizadas a partir de seus três pontos de entrada:

1. *Socket MDNS*: é *socket* que opera na porta UDP 5353, ele escuta por requisições de MDNS da rede local e as responde de acordo.
2. *Socket de Controle*: é o socket que recebe as requisições realizadas pelos programas utilizando as funcionalidades de MDNS/DNS-SD.
3. *Kernel Routing Socket*: é um *socket* da família AF_ROUTE para receber as notificações de alteração de estado das interfaces locais, tais como *link-up/link-down*, mudança de endereço e afins.

A **libmdns** é uma biblioteca (*shared library*[3]) que fornece a API para que os programas realizem requisições ao **mdnsd**. Esta API implementa uma série de funções para comunicação com **mdnsd** através de seu *socket* de controle, quando um programa deseja resolver um nome via MDNS. Por exemplo, ele utiliza funções da biblioteca que enviam uma mensagem para o *socket* de controle e aguarda por uma resposta, enquanto o **mdnsd** fará o que for necessário para atender a requisição e enviará uma resposta ao requerente.

O **mdnsctl** é um programa com interface em linha de comando para *debug* e utilização simples do MDNS/DNS-SD, este é basicamente uma interface para a utilização das funções da biblioteca, com ele pode se realizar uma consulta MDNS, enumerar serviços ou publicar um serviços na rede local, assim como outras pequenas funções como *dump* do *cache* e etc.

A programação do **mdnsd** é orientada a eventos, e não há nenhum tipo de concorrência, portanto não há múltiplas *threads* de controle. *Threads* são geralmente uma fonte de *bugs* e um complicador de depuração(21), e nesta implementação não há nada no problema que justifique a perda do determinismo pela adição de *threads*, sendo que todo processamento é baseado no tratamento de eventos, como por exemplo uma mensagem em algum *socket* ou o estouro de um *timeout*. É utilizada a biblioteca *libevent*(22) criada e desenvolvida por *Niels Provos*, ela fornece toda a abstração necessária de eventos, no seu funcionamento normal, registram-se *callbacks* que deverão ser chamadas caso cada evento ocorra, não há concorrência e todo processamento é feito no processo corrente. A *libevent* é na verdade uma abstração dos mecanismos de *multiplexing* de IO do UNIX como por exemplo poll(2)(23) e select(2)(24).

A comunicação feita pelo *socket* de controle, ou seja, o *socket* utilizado pela biblioteca **libmdns** para comunicação com o **mdnsd** é feita através do *framework* de IPC (Inter Process Communication) IMSG(25), criada e desenvolvida por *Henning Brauer*. É um *framework* bastante simples sendo basicamente uma abstração para se enviar e receber estruturas de dados diretamente, todo o *buffering* das mensagens é feito pelo próprio *framework*.

Os eventos enviados pelo **mdnsd** são assíncronos, isto é necessário por exemplo, quando se deseja realizar a enumeração de serviços de um determinado protocolo, um programa qualquer solicita a enumeração, e a medida que novos serviços vão sendo aprendidos pelo **mdnsd**, uma nova notificação é enviada ao programa, portanto, tal comunicação precisa ser assíncrona.

[3]Biblioteca que é vinculada dinâmicamente no momento da execução do programa

3.1.4.3 Status

O OpenMDNS, proposto neste trabalho, encontra-se em fase de desenvolvimento, que teve início em Fevereiro de 2010 tendo em vista ser abordado no Trabalho de Conclusão.

Atualmente já é possível realizar algumas das operações propostas neste trabalho, tais como resolver *hostnames*, enumerar serviços e resolvê-los. Foram feitos testes nas arquiteturas: alpha, i386, amd64 e sparc64. Foi utilizado uma máquina com o sistema operacional Linux e a implementação MDNS/DNS-SD Avahi para testes de interoperabilidade.

Ainda não é possível publicar serviços através da biblioteca *libmdns*, sendo esta a principal funcionalidade ausente no momento.

Em Maio de 2010 o OpenMDNS foi apresentado à dois integrantes da equipe do OpenBSD: Marco Peereboom e Nicholas Marriot, e estes fizeram diversos comentários, críticas e realizaram revisões de código, apontando *bugs* e imperfeições. Em Junho de 2010 o projeto foi levado à discussão entre a equipe do OpenBSD e outros membros demonstraram interesse. Sendo assim crê-se que o trabalho possui chances de ser aceito no futuro na base do OpenBSD. A inclusão do projeto representaria um marco importante, pois rapidamente milhares de pessoas estariam indiretamente utilizando o projeto, reportando suas experiências e *bugs* encontrados.

Portanto o objetivo deste trabalho é finalizar a implementação do OpenMDNS. Para isto será implementado a funcionalidade de publicação de serviços e outras secundárias, além dos itens descritos nas seções do *draft* MDNS(1) e *draft* DNS-SD(2) que ainda não foram implementados.

Na tabela 1 é apresentado o estado da implementação de cada seção descrita no *draft* MDNS(1) e na tabela 2 é apresentado o mesmo para o *draft* DNS-SD (2). O estado da implementação de cada seção é descrito como *I = "Implementado"*, *NI = "Não Implementado"*, *PI = "Parcialmente Implementado"* e *NS = "Não será implementado"*.

Neste trabalho serão implementados as seções descritas como *NI* e *PI*

3.1.4.4 Feedback

Desde que o OpenMDNS tornou-se funcional, diversos foram os *feedbacks* recebidos por usuários da comunidade.

Estes relatos são importantes a medida que ajudam no teste do projeto em ambientes

desconhecidos, facilitando muito o descobrimento de *bugs* e afins.

Também é fato que tais relatos são motivantes a medida que percebe-se se o rumo do trabalho está correto ou não.

Em especial agradou muito o relato enviado por Marco Peereboom, arquiteto *senior* de uma importante empresa que desenvolve dispositivos de *storage* e um dos membos mais respeitados da equipe do OpenBSD. Ele é o criador e arquiteto dos projetos *softraid(4)*(26) e *acpi(4)*(27). A seguir é apresentado uma cópia da conversa por *email*:

```
On Tue, Nov 09, 2010 at 11:11:05PM -0200, Christiano F. Haesbaert wrote:
> On 9 November 2010 21:50, Marco Peereboom <slash@peereboom.us> wrote:
> M: > I cnahaz publish?
> >
>
H> I found out that my first publish design was plain flawed and I
H> decided to step back and rethink.
H> Avahi does some crazy <censurado>when it comes to publishing features, I'm
H> trying to match all those feature without compromising the hole
H> design.
H> I've a new design in a piece of paper that looks promising.
H> There's also no point in implementing publishing without the proper
H> support for multiple interfaces, so the design gets trickier.
H> Thing is I'll be busy until December (exams and thesis), so publishing
H> will have to wait a little :/.
H> As soon as I get the publishing done I'll spend some time fixing bugs
H> and then will move forward to make a proper libmdns.

M: Cool.  I appreciate good design.

>
> M> I love the way it works and want to use it badly.
> >
>
H> Great ! That makes panda real happy :-) !
H> Please feel free to tell me what you need and I can try to make it a priority.
```

We are working on a product that requires backend infrastructure. I want to make the entire backend infrastructure use mdns to publish the services (maybe even users) on the network. I want zeroconf the whole thing.

I am by no means an expert on mdns but bare with me for a second.

We are going to for example have disks inside a machine and the idea is to advertise that disk using mdns. I know at least that is possible however I am wondering if I could reflect status in mdns. For example the disk went up in flames; can it say "don't talk to me"?

Another thing I though off is users. I am going to have multiple machines that host different users. Can I advertise on which machine marco has his account?

I have countless other questions but I think with these answered that I need to start reading the spec much more closely. Is there a quick and dirty guide you know of?

btw, what threw me for a loop is that publish is in the ui but didn't work ;-) A man page would have been awesome ;-)

Thanks a lot for this code it is super cool.

4 Conclusão

Os problemas endereçados pelos protocolos MDNS/DNS-SD estão presentes no dia a dia da maioria das pessoas, ao analisar os protocolos acredita-se que estes proporcionam uma solução viável aos mesmos, ressaltando ainda o fato que empresas como a Apple o vem utilizando a algum tempo com um moderado grau de sucesso.

Espera-se que ao término deste trabalho, o MDNS ganhe popularidade e que obtenha o *status* de RFC, contribuindo também para a disseminação do mesmo.

Apesar das alternativas existentes serem maduras e respeitáveis, acredita-se que algo mais simples pode surgir ao término deste trabalho, partindo de outras crenças e *background* quanto ao *design* de *software* em si.

Acredita-se que este trabalho apesar de exercitar os conceitos teóricos da Ciência de Computação possue um valor prático bastante considerável a medida que o objetivo do mesmo é resolver um problema real e do cotidiano.

O interesse dos membros do OpenBSD é um fato importante a ser ressaltado, assim como o fato de alguns poucos usuários já utilizarem o OpenMDNS de uma forma ou de outra, o *feedback* de usuários e *designers* de *software* mais experientes mostra-se de grande valor a medida que possibilita que o autor tenha uma maior gama de opções assim como sofrer de influências positivas quanto ao desenvolvimento do OpenMDNS.

A conclusão deste trabalho revela-se importante também a medida que é um desafio na formação do autor, este que nunca se envolveu em um projeto próprio deste porte.

Espera-se que ao término deste trabalho, o autor possa sair com um sentimento de vitória e conclusão, mais que isso, espera-se que outras pessoas possam se beneficiar do esforço deste trabalho, seja de uma forma, ou de outra.

Tabela 1: Estado da Implementação das Seções do Draft MDNS(1)

Seção	Item	Status
4	Reverse Address Mapping	I
5.1	One-Shot Multicast DNS Queries	NS
5.2	One-Shot Queries, Accumulating Multiple Responses	I
5.3	Continuous Multicast DNS Querying	I
5.4	Multiple Questions per Query	I
5.5	Questions Requesting Unicast Responses	NI
5.6	Direct Unicast Queries to port 5353	NI
6.1	Known Answer Suppression	I
6.2	Multi-Packet Known Answer Suppression	I
6.3	Duplicate Question Suppression	NI
7	Responding	PI
7.1	Negative Responses	NI
7.2	Responding to Address Queries	PI
7.3	Responding to Multi-Question Queries	I
7.4	Response Aggregation	I
7.5	Wildcard Queries (qtype "ANY" and qclass "ANY")	NI
7.6	Legacy Unicast Responses	NI
8	Probing and Announcing on Startup	I
8.1	Probing	I
8.2	Simultaneous Probe Tie-Breaking	NI
8.2.1	Simultaneous Probe Tie-Breaking for Multiple Records	NI
8.3	Announcing	I
8.4	Updating	NI
9	Conflict Resolution	NI
10.2	Goodbye Packets	NI
10.3	Announcements to Flush Outdated Cache Entries	NI
10.4	Cache Flush on Topology change	NI
10.5	Cache Flush on Failure Indication	NI
10.6	Passive Observation of Failures (POOF)	NI
11	Source Address Check	I

Tabela 2: Estado da Implementação das Seções do Draft DNS-SD(2)

Seção	Item	Status
4	Service Instance Enumeration (Browsing)	I
5	Service Name Resolution	I
8	Flagship Naming	NI
9	Service Type Enumeration	I
12	Discovery of Browsing and Registration Domains	NS

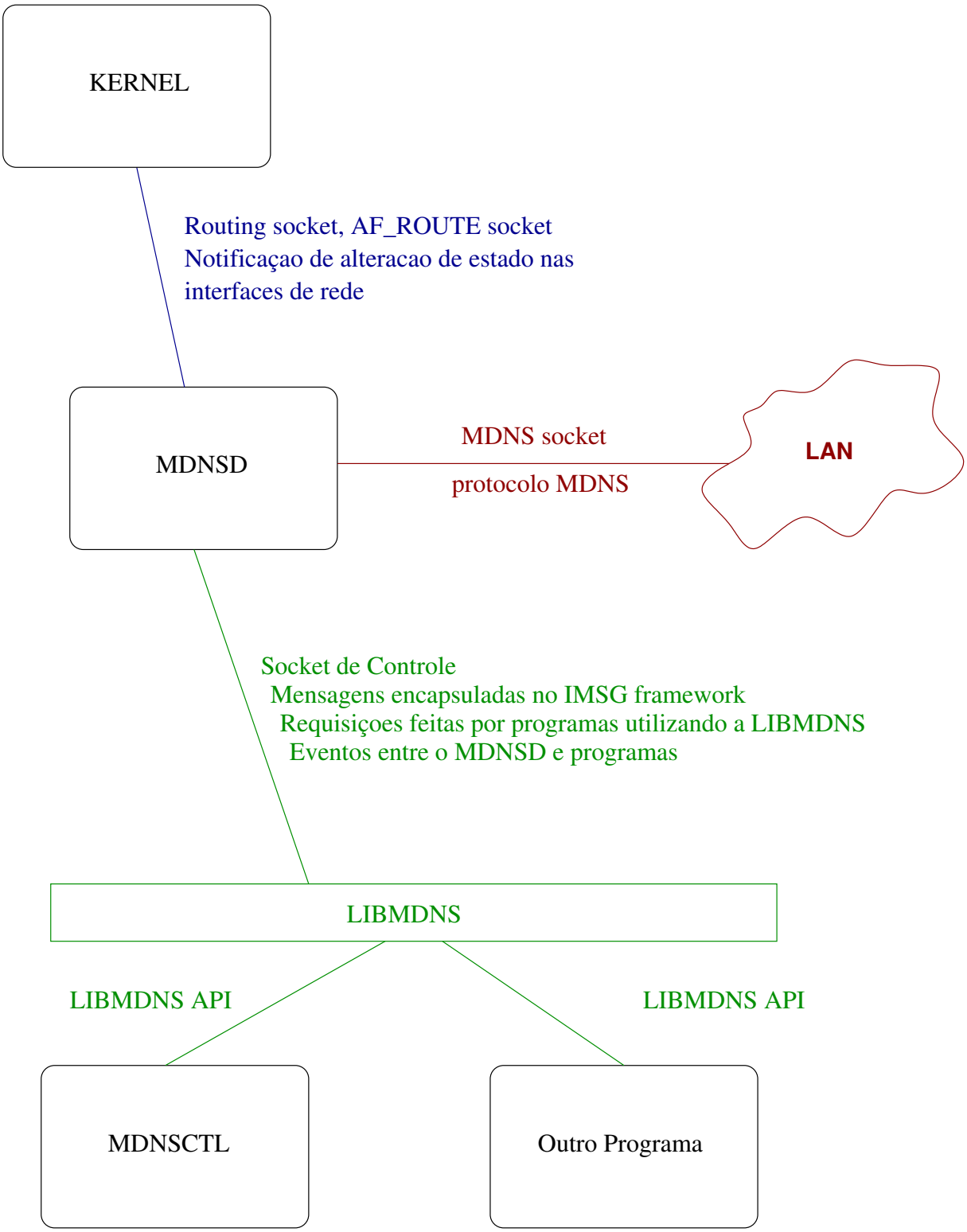


Figura 1: Arquitetura do OpenMDNS

Referências

1 CHESHIRE, M. K. S. *Multicast DNS*. [S.l.], set. 2010. At http://files.multicastdns.org/draft-cheshire-dnsext-multicastdns.txt.

2 CHESHIRE, M. K. S. *DNS Service Discovery*. [S.l.], maio 2010. At http://files.dns-sd.org/draft-cheshire-dnsext-dns-sd.txt.

3 MOCKAPETRIS, P. V. *RFC 1035: Domain names — implementation and specification*. nov. 1987. Disponível em: <ftp://ftp.internic.net/rfc/rfc1035.txt>.

4 BONJOUR. Disponível em: <http://developer.apple.com/networking/bonjour/download/>.

5 APACHE-2 License. Disponível em: <http://www.apache.org/licenses/LICENSE-2.0.html>.

6 AVAHI. Disponível em: <http://www.avahi.org>.

7 OPENBSD, Operating System. Disponível em: <http://www.openbsd.org>.

8 OPENBSD Policy. Disponível em: <http://www.openbsd.org/policy.html>.

9 ISC License. Disponível em: <http://opensource.org/licenses/isc-license.txt>.

10 NetBIOS Working Group. *RFC 1001: Protocol standard for a NetBIOS service on a TCP/UDP transport: Concepts and methods*. mar. 1987. See also STD0019 . Status: STANDARD. Disponível em: <ftp://ftp.internic.net/rfc/rfc1001.txt, ftp://ftp.math.utah.edu/pub/rfc/rfc1001.txt>.

11 GETADDRINFO Library Call. Disponível em: <http://www.openbsd.org/cgi-bin/man.cgi?query=getaddrinfo>.

12 GETHOSTBYNAME Library Call. Disponível em: <http://www.openbsd.org/cgi-bin/man.cgi?query=gethostbyname>.

13 OPENBGP. Disponível em: <http://www.openbgp.org>.

14 OPENOSPF. Disponível em: <http://www.openbsd.org>.

15 OPENSSH. Disponível em: <http://www.openssh.com>.

16 DHCP Client. Disponível em: <http://www.openbsd.org/cgi-bin/man.cgi?query=dhclient>.

17 FORK System Call. Disponível em: <http://www.openbsd.org/cgi-bin/man.cgi?query=fork>.

18 NETBSD, Operating System. Disponível em: <http://www.netbsd.org>.

19 THE OpenBSD packages and ports system. Disponível em: <http://www.openbsd.org/faq/faq15.html>.

20 EXPLOIT Mitigation Techniques. Disponível em: <http://www.openbsd.org/papers/ven05-deraadt/>.

21 LEE. The problem with threads. *COMPUTER: IEEE Computer*, v. 39, 2006.

22 LIBEVENT. Disponível em: <http://monkey.org/ provos/libevent/>.

23 POLL System Call. Disponível em: <http://www.openbsd.org/cgi-bin/man.cgi?query=poll>.

24 SELECT System Call. Disponível em: <http://www.openbsd.org/cgi-bin/man.cgi?query=select>.

25 IMSG Framework. Disponível em: <http://www.openbsd.org/cgi-bin/man.cgi?query=imsg¿*nit*¿.

26 SOFTRAID. Disponível em: <http://www.openbsd.org/cgi-bin/man.cgi?query=softraid>.

27 ACPI. Disponível em: <http://www.openbsd.org/cgi-bin/man.cgi?query=acpi>.

28 MOCKAPETRIS, P. *Domain Names - Concepts and Facilities*. [S.l.], nov. 1987. At http://info.internet.isi.edu/in-notes/rfc/files/rfc1034.txt.

29 CHESHIRE, M. K. S. *Requirements for a Protocol to Replace AppleTalk NBP*. [S.l.], set. 2010. At http://files.dns-sd.org/draft-cheshire-dnsext-nbp.txt.

30 CHESHIRE, M. K. S. *Requirements for a Protocol to Replace AppleTalk NBP*. [S.l.], mar. 2010. At http://files.multicastdns.org/draft-cheshire-dnsext-nbp-08.txt.

31 THEO De Raadt Interview. Disponível em: <http://www.thejemreport.com/content/view/369/>.

32 BSD License. Disponível em: <http://www.opensource.org/licenses/bsd-license.php>.

33 Brooks, Jr., F. P. *The Mythical Man Month: Essays on Software Engineering*. first. Addison-Wesley, 1975. 200 p. Disponível em: <http://www.amazon.com/Mythical-Man-Month-Essays-Software-Engineering/dp/0201006502>.

34 RAYMOND, E. S. *The Art of UNIX Programming*. pub-AW:adr: Addison-Wesley, 2004. xxxii + 525 p. ISBN 0-13-124085-4.

35 STEVENS, R. W. *UNIX Network Programming*. [S.l.]: Prentice Hall PTR, 1990. (Software Series).

36 STEVENS, R. W. *TCP/IP Illustrated, Volume 1: The Protocols*. Reading: Addison Wesley, 1994. ISBN 0-201-63346-9.

37 STEVENS, W. R. *TCP/IP Illustrated, Volume 2*. [S.l.]: Addison Wesley, 1994.

38 STEVENS, W. R. *Advanced Programming in the UNIX Environment*. [S.l.]: Addison-Wesley, 1992.

39 HUSEMANN, M. Fighting the lemmings. Unspecified.