



GNUnet - Uma *Framework Peer-to-Peer*

Marcos Daniel Marado Torres
marado@student.dei.uc.pt

Tiago Jorge Limpo Serra
tserra@student.dei.uc.pt

DEInix - Clube de Utilizadores de Sistemas Operativos Unix-Based
Departamento de Engenharia Informática
Universidade de Coimbra
Vila Franca - Pinhal de Marrocos
3030 Coimbra, Portugal

Conteúdo

1	O GUNet	2
2	Objectivos, Filosofia e Implementação	3
2.1	Porquê ligações peer-to-peer?	3
2.2	Mas porquê uma framework?	5
2.3	O GUNet e o Anonimato	5
2.4	Encriptação no GUNet	8
2.5	O sistema de Routing do GUNet	8
2.5.1	Descarregar conteúdo do GUNet em dois passos	9
2.5.2	Processar Pedidos	10
2.5.3	Propagação de Pedidos	10
2.5.4	Routing Aleatório	12
2.5.5	Hot Path Routing	13
2.5.6	Parametrizando o ataque – soluções para o futuro	15
2.6	Autenticação/Não-Repúdio	17
2.7	Accounting	17
2.7.1	Queries de prioridade zero	18
2.8	Collections	18
2.9	Protocolos de Comunicação	19
2.9.1	Peer-to-Peer	20
2.9.2	Comunicação Cliente-Servidor	22
2.9.3	Serviços de Transporte	28
3	Aplicações	29
3.1	O sistema de partilha de ficheiros	29
3.2	Aplicações Emergentes	32
4	Bibliografia	34

Capítulo 1

O GNUnet

O GNUnet¹ é uma *framework* para conexões *peer-to-peer* seguras que não usa nenhum serviço centralizado, ainda em desenvolvimento e em formato aberto, pertencendo ao projecto GNU². Encontra-se neste momento na sua versão 0.6.4a, sendo que ainda existe muito a fazer a nível conceptual e de implementação até poder ser considerada uma tecnologia estável. No entanto o seu carácter inovador faz com que seja um projecto promissor.

Um primeiro serviço implementado acima da camada de rede permite partilha de ficheiros de uma forma anónima e sem censura. O GNUnet usa um modelo simples e económico, baseado em excessos, para alocar recursos. Os nós no GNUnet monitorizam o comportamento dos outros nós a respeito do uso de recursos; os nós que contribuem para a rede são compensados com um melhor serviço.

Acima desta *framework* é virtualmente possível a implementação de qualquer tipo de aplicação que tenha necessidades de comunicação através de uma rede. No entanto, enquanto o seu conceito vai sendo desenvolvido, cada vez mais surge a necessidade de planear qual a melhor forma de desenhar essas aplicações de forma a tirar partido das potencialidades do GNUnet, principalmente no que diz respeito à encriptação e ao anonimato.

¹<http://www.ovmj.org/GNUnet>

²<http://www.gnu.org/>

Capítulo 2

Objectivos, Filosofia e Implementação

“ Estou certo que já notou que as tecnologias que estão por detrás dos computadores e das redes têm muitas falhas. Claro, existem interfaces pouco intuitivas e *crashes* frequentes no sistema. Além desses problemas facilmente detectáveis, existem algumas falhas fundamentais no desenho e implementação dos Sistemas Operativos, Aplicações e Protocolos. Recorrendo a essas mesmas falhas, um atacante pode roubar dados, controlar sistemas ou simplesmente causar o caos. ”

Ed Skoudis¹

Foi exactamente porque a maioria dos Sistemas e Protocolos usados hoje em dia não foram criados a pensar na Segurança Informática, estão assentes sobre outros Protocolos que não possuem mecanismos seguros, ou porque simplesmente o seu objectivo primário foi atingido da forma mais simples e/ou eficiente, à custa da Segurança, que se sentiu a necessidade de criar o GUNet: algo que, pela sua natureza, é intrinsecamente seguro.

2.1 Porquê ligações peer-to-peer?

As redes peer-to-peer são usadas em grande escala para partilhar todo o tipo de conteúdos digitais com outros utilizadores. Foi demonstrado pela primeira vez com o Gnutella² que este tipo de redes peer-to-peer podem ser organizadas de modo descentralizado, para que não exista tecnicamente nenhuma maneira de desligar a rede. Graças a isso, o Gnutella e algumas redes topologicamente similares tornaram-se uma plataforma popular de partilha de conteúdos legais ou ilegais (i.e. registados ou sujeitos

¹Citação do livro “Counter Hack - A Step-by-Step Guide to Computer Attacks and Effective Defenses”, presente na bibliografia.

²<http://www.gnutella.com/>

a censura). Enquanto que pesquisar por conteúdos na rede Gnutella é relativamente anónimo, as respostas a essas pesquisas não são anónimas, visto que o endereço IP da pessoa que tem o conteúdo pesquisado é sempre exposto. Graças a esse facto é possível processar utilizadores com conteúdos ilegais ou que de outra forma quebrem a lei. Visto este factor desencorajar os utilizadores a partilhar conteúdos ilegais, a censura é indirectamente aplicável.

O GUNet tem aplicações que vão muito além das possibilidades do Napster³, Gnutella ou Freenet⁴. O motor de pesquisas permite que uma larga porção da WWW⁵ migre o seu conteúdo para o GUNet, as corporações podem decidir alojar dados importantes descentralizadamente no GUNet e muitos utilizadores irão gostar do serviço visto este oferecer um nível de privacidade que iria de outra forma tornar-se difícil de atingir dada a actual configuração da Internet.

Porque deveriam sites na Internet migrar para o GUNet? Em primeiro lugar, o GUNet iria permitir-lhes distribuir os seus conteúdos por todo o globo. Não haveria necessidade de um servidor central a receber milhões de visitas num curto espaço de tempo. Em vez disso, o conteúdo será automaticamente distribuído, reduzindo o tempo de *download* para os utilizadores. A importância deste factor é evidenciado com a rápida adopção verificada pelo sistema de BitTorrent⁶. Além disso, para fornecer conteúdos acedidos frequentemente, os nós dos publicadores de conteúdo irão ganhar crédito no GUNet, Os utilizadores também irão apreciar a possibilidade de descarregar conteúdos web sem terem a preocupação da possibilidade de terceiros a construir perfis dos seus hábitos de navegação. Com a emergência de empresas especializadas em publicidade dirigida na Internet (como por exemplo a DoubleClick⁷ ou o Passport⁸), os consumidores estão cada vez mais preocupados com a privacidade dos seus dados pessoais. O GUNet assegura que a ligação entre o contribuidor de conteúdos e o consumidor desses mesmos conteúdos é completamente anónima, acabando assim com a possibilidade de empresas usarem dados privados para os seus próprios propósitos sem consentimento explícito do utilizador.

Pequenos contribuidores de conteúdos ainda encaram actualmente outro problema: DNS⁹. De forma a atingir alguma visibilidade, eles não têm apenas que pagar uma quantia aos monopólios de DNS, mas também que encarar advogados a tentar processá-los pelo uso de marcas registadas potencialmente desconhecidas. A ocupação de domínios por vezes não faz sentido, até mesmo para médias empresas. O site www.gatt.org é um grande exemplo disso. No GUNet os domínios deixam de ter significado. Quão mais regularmente um *site* for pedido, mais rapidamente ele irá aparecer. Resultados múltiplos para um mesmo pedido são possíveis.

³<http://www.napster.com/>

⁴<http://freenet.sourceforge.net/>

⁵World Wide Web

⁶<http://bittorrent.com/>

⁷<http://www.doubleclick.com/us/>

⁸<http://www.passport.net>

⁹Domain Naming Service

Os créditos no GUNet poderão ser trocados por dinheiro. Os ISP's¹⁰, por exemplo, poderão ser classificados pelo seu crédito no GUNet. Conforme os servidores dos ISP's vão divulgando os pedidos dos seus clientes com esse crédito, os clientes desse ISP irão obter um serviço concordante com o crédito do seu ISP. Os ISP's terão modos de atingir mais crédito: ou aumentando a sua largura de banda ou a sua capacidade de armazenamento para ganhar crédito directamente do GUNet, ou trocando crédito (por exemplo com divulgadores de conteúdos ou com outros ISP's que decidam que têm mais crédito que aquele que necessitam). Empresas especializadas em cópias de segurança podem usar o GUNet para fazer cópias de dados importantes. Eles lucram pelo anonimato (por exemplo, a Microsoft não pode decidir apagar dados armazenados para a IBM) e pela distribuição: não existe um ponto de falha. E, vez de contruir sistemas de redundância, a rede irá atingir a distribuição e a redundância automaticamente. Isto é particularmente importante visto que a redundância local nem sempre ajuda: basta considerar os recentes problemas energéticos na California para nos apercebermos que uma grande disponibilidade de conteúdos é atingida melhor a uma escala global.

2.2 Mas porquê uma framework?

Se, por um lado, o objectivo é criar o melhor sistema de partilha de ficheiros anónimo disponível, por outro lado fazer um sistema mais poderoso do que um simples sistema de partilha de ficheiros, através da criação de uma *framework*, não só atinge outros objectivos, como também acaba por, de certa forma, melhorar o sistema de partilha de ficheiros.

Se outras aplicações peer-to-peer usarem a framework GUNet e fizerem passar o seu tráfego em canais GUNet encriptados ligação a ligação, o tráfego na rede GUNet e o número de nós participantes aumenta – o que poderá melhorar o anonimato de forma significativa. Além disso, aplicações adicionais irão ajudar o projecto GUNet com mais programadores e *testers* o que deverá tornar o núcleo do GUNet ainda mais sólido. Finalmente, a equipa de desenvolvimento do GUNet acredita que certas características gerais do GUNet, em particular a descoberta de nós, encriptação ligação a ligação, autenticação e abstracção da camada de transporte irão ser ferramentas importantes para outros projectos de software livre, pelo que ao fazê-los facilmente acessíveis irá beneficiar o desenvolvimento de software livre como um todo.

2.3 O GUNet e o Anonimato

O GUNet é constituído por nós que comunicam entre si. Cada nó escolhe um par de chaves, que é usado na identificação, autenticação e encriptação da comunicação entre os nós. Para esconder quais nós é que estão realmente a comunicar entre si, o GUNet

¹⁰Internet Service Providers

usa uma abordagem do tipo MIX¹¹: os nós são intermediários que enviam mensagens para outros nós.

O termo “comunicação anónima” é usualmente entendido como uma comunicação para a qual é impossível terceiros identificarem os participantes envolvidos. Mas na realidade a comunicação anónima é mais do que isso: numa comunicação anónima é suposto garantir-se que os dados transferidos não podem ser relacionados com o emissor da mensagem ou o receptor da mesma, mas apenas com os nós que participam imediatamente na sua difusão (e que podem ser apenas intermediários). A comunicação deve ser confidencial no sentido em que apenas o receptor conhece o conteúdo da mensagem. O emissor e os intermediários não devem ter meios para determinar o conteúdo. Além disso, o emissor do conteúdo deve ter meios plausíveis de negar uma acusação de publicação desses mesmos conteúdos, mesmo que todos os nós (excepto o do emissor) por onde a mensagem tenha passado até chegar ao seu destino pertençam a um “atacante” (considerando “atacante”, neste caso, aquele que quer provar a acusação de publicação) e registem todas as transacções passadas por eles. Este tipo de requisitos de anonimato (vulgarmente denominado de “Anonimato de nível três”, visto que nem o emissor conhece o receptor, nem o receptor conhece o emissor, nem os intermediários sabem quem é o emissor nem o receptor) é difícil de atingir, especialmente se as partes envolvidas na comunicação tiverem de supostamente identificar-se para as camadas de baixo nível dos protocolos que providenciam a identificação e a confidencialidade. Isto verifica-se particularmente quando nós maliciosos estão envolvidos e têm a capacidade de expôr os pacotes que conseguem descriptar.

Anonimato é não conseguir distinguir entre um indivíduo e um possivelmente grande grupo. Um objectivo central do AFS, inicialmente chamado assim por significar “Anonymous File Sharing”, do GUNet é fazer com que todos os utilizadores (nós) criem um grupo e fazer com que as comunicações entre esse grupo sejam anónimas, ou seja, ninguém (menos o emissor) deve saber qual dos nós do grupos originou a mensagem. Deve ser impossível para um adversário distinguir entre o nó original e os outros nós. Em particular, mesmo os nós não devem ter a hipótese de reconhecer de que nó é que a mensagem é originária.

Claro que, na prática, poderá ser possível para um adversário poderoso fazer algumas análises e atribuir potencialmente altas probabilidades a um nó da rede de ser o emissor da mensagem para uma sub-rede de nós. O AFS tenta tornar isso o mais difícil possível¹²¹³¹⁴. O grau de anonimato no GUNet depende dos recursos (principalmente largura de banda) que o indivíduo tem disponível.

No caso de um adversário extremamente poderoso tentar quebrar o anonimato de um nó, o GUNet disponibiliza um sistema de recusa, querendo dizer com isto que a comunicação é secreta no sentido em que apenas o destinatário sabe a chave para descriptar a mensagem. O emissor e os intermediários não têm a possibilidade de determinar

¹¹<http://www.ovmj.org/GUNet/papers/mix-acc.ps>

¹²<http://www.ovmj.org/GUNet/download/aff.ps>

¹³<http://www.ovmj.org/GUNet/download/pet/>

¹⁴<http://petworkshop.org/>

os verdadeiros dados que estão a ser transmitidos. A partir do momento que a mensagem está rede, o remetente da mensagem pode muitas vezes não ter conhecimento do conteúdo da mensagem, visto que o conteúdo pode ter migrado para esse nó, tornando o emissor da mensagem indistinguível do intermediário. Visto que os intermediários não têm meios de descriptar o conteúdo, não são legalmente responsáveis por eles (se usar a internet para usar mails encriptados o seu ISP tipicamente não será responsável pelo conteúdo que os seus servidores transmitem; no GNUnet cada nó tem o papel de um ISP, fornecendo serviços de Internet a outros nós).

O GNUnet tenta atingir o anonimato tornando praticamente impossível aos nós que estão a disponibilizar conteúdos de identificar que tipo de conteúdos estão a servir. Além disso, os pedidos são encriptados pelo cliente e não podem ser decifrados pelos intermediários ou o nó final que disponibiliza os conteúdos. Tornar impossível aos intermediários descriptar o conteúdo é atingido através da criação de uma *hash* da *string* pesquisada usando uma função unidireccional¹⁵ e usando o *hashcode* como um índice do ficheiro pedido. De forma a permitir pesquisas razoáveis, os *hashcodes* podem ser combinados usando os operadores lógicos *and*, *or* e *not*. Isto aumenta as capacidades de pesquisa do GNUnet em relação a outros sistemas como o Freenet. Por outro lado, usando uma combinação de pequenas palavras fazem com que ataques que recorram a dicionários sejam mais fáceis.

Apesar de tudo, a escolha das palavras-chave cabe aos utilizadores: são eles que devem decidir entre usabilidade (palavras-chave pequenas) e segurança (palavras-chave grandes).

Os dados armazenados no GNUnet são divididos em pequenos pedaços para os quais é individualmente criada uma *hash*. Estes *hashcodes* servem como chaves para as partes distribuídas do ficheiro. Os *hashcodes* são agrupados em nós indireccionais, semelhantes aos inodes do UNIX. Falando na generalidade, desde o momento em que o conteúdo está fragmentado, nenhum publicador (excepto aquele que está a inserir o conteúdo desde o início) vai ter ficheiros completos. Além disso, enquanto os nós indireccionais e os nós de dados não são facilmente distinguíveis, é difícil para o publicador de conteúdos determinar se os clientes estão a pedir dados ou meta-dados. Dividir o conteúdo em pedaços uniformes de *1k* faz com que também seja muito mais fácil trocar conteúdos. Ao contrário de outros sistemas onde o ficheiro completo tem de migrar (limitando a possibilidade de ficheiros grandes serem movidos), migrar partes de um ficheiro no GNUnet necessita apenas que um pacote seja enviado. Em geral, o conteúdo deve migrar entre nós que efectuem pedidos. Conteúdo requisitado frequentemente deverá ser armazenado em nós com grande largura de banda, enquanto que dados raramente acedidos devem migrar para nós de capacidade inferior.

Dividir os dados em pequenos pedaços tem ainda a vantagem de efectuar downloads do GNUnet estar-se usufruir da distribuição; o ficheiro pode ser descarregado em paralelo de vários nós, criando pouco custo para cada nó (à excepção do receptor). A

¹⁵O GNUnet usa o RIPEMD160:
<http://www.esat.kuleuven.ac.be/~bosselae/ripemd160.html>

desvantagem é que o conteúdo deve ser multiplicado visto que os nós podem não estar disponíveis. Por outro lado, isto também ajuda a garantir que o conteúdo não é imediatamente perdido quando um nó se desliga da rede.

2.4 Encriptação no GUNet

O GUNet usa RSA para a encriptação assimétrica e o Blowfish para as trocas simétricas.

O RSA foi escolhido porque parece ser a escolha mais apropriada (isto é, difícil de quebrar e livre de patentes) para os propósitos do GUNet. O Blowfish foi escolhido porque o protocolo pretende ser para implementações no mais variado *software*, e o Blowfish está disponível gratuitamente além de ser rápido.

O RIPE160 foi escolhido para a primitiva de *hash* porque esta é a função com funções de *hashing* maiores (em termos de saídas) suportado pelo OpenSSL. A escolha de 160 *bits* faz com que as colisões sejam muito improváveis, mesmo quando existem muitos conteúdos na rede.

O CRC32 foi escolhido como função para efectuar o *checksum* dos dados porque existem exactamente 32 bits a mais nos *inodes* de 1024 *bytes* se cada *inode* guardar 51 *hashes* de RIPE160.

2.5 O sistema de Routing do GUNet

A partir do momento em que o UDP¹⁶ é usado, o atraso da ligação à rede não pode ser medido através do simples envio de pacotes (especialmente visto que o receptor pode não responder instantaneamente ou de todo). Por isso, a melhor forma de obter informação de *routing* poderá ser olhar para os endereços IP e assumir que os endereços mais próximos estão de facto mais próximos do nó local. O recente worm *Code Red*¹⁷ foi relativamente sucedido usando esta técnica. Claro que esta heurística não impede o uso de melhores implementações para recolher mais informação (por exemplo através do uso do *ping* (pacotes ICMP¹⁸) ou novos sub-protocolos do GUNet). É da responsabilidade de cada nó otimizar este comportamento. Apesar de tudo, cada nó é avaliado pela percepção que os outros nós têm do seu desempenho. Cada pacote enviado no GUNet tem um determinado *overhead*. Isto inclui encriptação e desencriptação, comunicação do cabeçalho do pacote e o processamento actual do pacote. De forma a manter o número de pacotes baixo, o GUNet cria *buffers* de saída para cada nó de destino, tentando atingir um tamanho óptimo para os pacotes. Por omissão, o tamanho óptimo para os pacotes é de 1472 *bytes* (o tamanho óptimo para o *ethernet*¹⁹). Manter os dados

¹⁶User Datagram Protocol

¹⁷<http://www.cert.org/advisories/CA-2001-19.html>

¹⁸Internet Control Message Protocol

¹⁹<http://www.ethermanage.com/ethernet/ethernet.html>

de envio por nó em *buffers* é particularmente importante para os pedidos, desde que o nó possa escolher livremente a quem enviar o pedido. Os bons candidatos para efectuar pedidos são os nós que tenham pedidos de saída pendentes por não terem sido enviados visto o tamanho mínimo dos pacotes ainda não ter sido atingido.

A heurística usada para decidir a quem enviar dados está aberto a melhorias.

Mudanças para esses detalhes de implementação num nó de modo a melhorar a performance não são uma violação do protocolo GUNet.

De modo a evitar que o GUNet se torne uma vítima das *firewalls* modernas, os nós de GUNet devem ter a hipótese de reencaminhar as suas comunicações em protocolos cujas hipóteses de abandono pela comunidade da Internet sejam mínimas. Exemplos disso são os protocolos HTTP, SMTP e FTP. Se os nós de GUNet puderem substituir a camada protocolar do UDP e comunicar através destes canais comuns, as *firewalls* e outras restrições como *traffic shapers* ou *traffic droppers* poderão mostrar-se ineficientes.

Actualmente o GUNet consegue, num cenário em que o UDP não pode ser usado, usar o TCP para contornar o problema. A implementação do protocolo SMTP já foi feita, no entanto a implementação do protocolo HTTP para este efeito ainda se encontra com alguns problemas²⁰ e a implementação do protocolo FTP, apesar de desejável, não está programada para inclusão, estando-se à espera que alguém contribua para o projecto com a sua implementação²¹.

2.5.1 Descarregar conteúdo do GUNet em dois passos

Para descarregar conteúdo do GUNet precisamos de efectuar dois passos:

- Descobrir RBlocks: Os RBlocks são descobertos usando pedidos de pesquisa contendo uma lista de chaves de *hash* tripla: $H(H(H(K_j)))$. Um nó que receba um pedido e que tenha um RBlock guardado sobre um destes valores devolve o RBlock encriptado $EH(K_j)(R)$ juntamente com o $H(H(K_j))$ para provar que ele possui realmente o conteúdo alojado para este pedido. O nó que fez o pedido pode descriptar o RBlock usando $H(K_j)$.
- Descarregar o Conteúdo: Depois de um RBlock ter sido descoberto, o nó pode descarregar o ficheiro correspondente criando vários pedidos de *download*. O nó começa por usar a *hash* de pedido incluída nas folhas. Os IBlocks são usados para descarregar e descriptar os DBlocks do ficheiro.

O nó pode também usar pedidos múltiplos para receber vários IBlocks ou DBlocks de uma só vez. Um pedido múltiplo é constituído por várias *hashes* de *download* e a *superhash* correspondente para acelerar os *lookups*.

Temos então dois tipos de pedidos: de pesquisa e de *download*. Ambos os tipos de pedidos contêm três valores adicionais que são usados para processar o pedido:

²⁰detalhes em http://www.ovmj.org/~mantis/bug_view_page.php?bug_id=765

²¹como se pode ver em

<http://lists.gnu.org/archive/html/gnunet-developers/2004-12/msg00005.html>

- Um endereço de retorno;
- Uma prioridade;
- Um TTL²².

2.5.2 Processar Pedidos

Cada nó estabelece uma fila de mensagens para cada nó vizinho. Antes de um pedido ou de uma resposta é enviada para um nó vizinho, é temporariamente alojado na fila representando este nó. Cada fila é processada em intervalos aleatórios, depois todas as mensagens à espera nesta fila são enviadas para o nó correspondente. Um nó que receba um pedido procede da seguinte forma:

- Se o conteúdo pedido (RBlock, IBlock ou DBlock) é encontrado localmente, é criada uma resposta e colocada na fila de mensagens correspondente ao endereço de retorno presente no pedido. De outra forma (mas sempre no caso de um pedido de pesquisa), o pedido é enviado para outros nós.
- Dependendo da prioridade e largura de banda disponível, o nó selecciona aleatoriamente n vizinhos e coloca o pedido nas suas filas de mensagens. O pedido é ajustado da seguinte forma:
 - A nova prioridade é calculada como $p(n + 1)$, onde o p é a prioridade antiga.
 - O nó ou guarda o endereço de retorno do nó predecessor ou substitui-o com o seu próprio endereço. Neste último caso, o nó adiciona o pedido à sua tabela de *routing* actual.

É importante mencionar que os nós intermediários não sabem que pedidos estão relacionados (à excepção dos pedidos múltiplos). Por isso não existe nenhum caminho pré-definido que seja usado para pedidos de reencaminhamento relacionados. De qualquer forma, o processo de selecção tem tendência a escolher os vizinhos que tenham respondido recentemente.

2.5.3 Propagação de Pedidos

A propagação de pedidos tem de resolver dois problemas. Em primeiro lugar, os pedidos devem ser redireccionados de forma a que não passem duas vezes pelo mesmo nó. Por outro lado, o reencaminhamento dos pedidos deve tentar ser eficiente e deve ser escalável. Existem vários esquemas elaborados que foram desenhados em termos de reencaminhamento de pedidos e de conteúdos de uma forma a que o *lookup* seja rápido. Uma abordagem particularmente interessante é a escolhida pelo Freenet, onde

²²Time To Live

conteúdos similares (em termos da chave depois de *hashed*) são alojados em nós topologicamente próximos. Usando este método os nós podem adivinhar em que área da rede o conteúdo pode estar alojado. Actualmente o GUNet não tenta usar esquemas tão avançados como o descrito, mas poderão vir a existir implementações para esquemas destes futuramente.

Para passar pedidos de um nó para outro, há que resolver três problemas:

- Seleccionar um sub-grupo dos nós conhecidos como possíveis nós a fazer a reencaminhamento;
- Seleccionar a prioridade do pedido a fazer reencaminhamento;
- Registrar todos os pacotes reencaminhados que voltam pelo nó escolhido para o reencaminhamento.

No GUNet, cada pedido tem associado a si dois valores inteiros que ajudam o nó a decidir cada uma destas questões.

O primeiro valor inteiro é a prioridade do pedido. Indica quão importante é o pedido para o envio (ou reencaminhamento) do pedido. Um valor de zero indica que este pedido deverá ser apenas considerado se existir largura de banda em excesso.

Como primeiro passo para este processo, o nó que recebe o pedido considera o *ranking* do nó que envia o pedido e ajusta a prioridade. Se a prioridade era superior ao crédito do nó que enviou o pedido, a prioridade é reduzida a esse valor.

Depois, o crédito do nó que enviou o pedido é diminuído pela restante prioridade do pedido (cobrança pelo serviço) se a largura de banda disponível de momento for baixa. Nesta altura o nó verifica se o conteúdo está disponível localmente e envia, eventualmente, uma resposta.

Se o nó está ocupado e a prioridade é baixa, o nó pode decidir não responder, mesmo que tenha os dados pedidos.

Se o nó não tiver o conteúdo pedido, então considera o segundo inteiro presente no pedido.

Este segundo inteiro dá o TTL para o pedido. O nó emissor pode tê-lo marcado arbitrariamente alto, por isso o nó local pode primeiro reduzir o TTL para um valor razoável. Se o TTL for zero o pedido expirou e é ignorado. Se o TTL for superior a zero, então o nó adiciona o pedido (emissor, *hashcode* e TTL) à sua tabela local de pedidos.

Se um pedido com TTL superior já existir na tabela, o novo pedido é adicionado mas não reencaminhado: uma resposta ao tal pedido anterior poderá surgir a qualquer momento ou o pedido é devolvido ao nó local.

Em geral, a tabela de reencaminhamento é usada para alojar informação sobre pedidos reencaminhados de forma a que os nós possam reencaminhar depois as respostas.

Dependendo da nova prioridade do pedido e da largura de banda disponível, o nó selecciona n outros nós aleatoriamente para reencaminhamento. O nó diminui então ao TTL para uma quantidade razoável (maior que o tempo esperado tendo em conta

o atraso na rede e o processamento dos pedidos). Ele diminui a prioridade do pedido para $p(n + 1)$ e reencaminha o pedido com o TTL e a prioridade ajustados para esses n nós. O número para n depende da carga actual da rede. Se a carga for grande, o nó pode decidir reencaminhar o pedido para poucos nós. O TTL do pacote é decrementado ao longo do tempo em cada nó (por exemplo, $TTL -$ por cada 30 segundos). Uma vez o TTL atingindo zero, a entrada é removida da tabela de reencaminhamento (o pedido é considerado sem sucesso).

2.5.4 Routing Aleatório

Com o reencaminhamento aleatório os nós a receber pedidos são seleccionados aleatoriamente.

Seja k o número de nós vizinhos ao nó A .

Por cada pedido o nó escolhe aleatoriamente m dos k vizinhos e envia-lhes o pedido.

Aí, a probabilidade de um certo vizinho ser seleccionado uniformemente é $P_{rnd}(A) = \frac{m}{k}$ para todos os vizinhos de A .

De notar que a probabilidade P_{rnd} é específica do nó e desconhecida para um atacante.

Análise à tolerância a ataques

Para testar que nó vizinho B_i do precedente nó A está mais perto do emissor, compara-se o número de pedidos recebidos desses nós.

Seja q_A o número de pedidos relacionáveis que o atacante recebe de A . Então, o atacante pode esperar receber o mesmo número de pedidos relacionáveis de um nó já testado B no mesmo intervalo.

Se o nó testado B estiver mais perto do emissor, o atacante espera receber $q_B = \frac{q_A}{P_{rnd}(A)}$ pedidos relacionáveis deste nó.

Caso isso não aconteça, o nó testado está mais distante do emissor e o atacante espera receber apenas $q'_B = q_A \cdot P_{rnd}(B)$ pedidos relacionáveis deste nó. É importante que $P_{rnd}(A)^{-1} \gg P_{rnd}(B)$ se verifique para que haja sucesso neste teste simplificado. Por isso, quanto menor for P_{rnd} , mais fácil será para o atacante usar este teste para distinguir que nó está mais perto do emissor.

- Em geral o P_{rnd} é muito pequeno no GUNet, visto que cada nó tem muitos nós vizinhos, mas poucos deles são seleccionados aleatoriamente. Neste caso, o nó que está mais perto do emissor pode ser determinado eficientemente.
- Por outro lado, se a carga no nó for baixa, o nó começa a intercalar as filas de mensagem vazias com pedidos recebidos e o P_{rnd} aumenta. Se os nós estiverem finalmente a espalhar os pedidos para todos os vizinhos, até o teste mais exacto falha, visto que $P_{rnd}(A)^{-1} = P_{rnd}(B) = 1$.

Devido a estes aspectos, o ataque torna-se mais difícil com pouca carga. Se a carga na rede for baixa, o atacante tem de aumentar a carga para o ataque ter sucesso. De notar que os nós com pouca carga estão a aceitar pedidos com prioridade zero e não é necessário crédito para aumentar a carga.

Número de pedidos necessários

Para estimar quantas mensagens o emissor pode enviar sem ser identificado, assume-se que P_{rnd} é igual para todos os nós na rede GUNet. Seja l a distância do atacante ao emissor.

- O atacante vai receber pedidos relacionáveis do nó intermediário i com uma probabilidade de P_{rnd}^i , onde $1 \leq i \leq l$. O primeiro pedido é então recebido depois do emissor ter enviado $m_i \geq \frac{1}{P_{rnd}^i}$ pedidos.
- O atacante vai receber pedidos relacionáveis de vizinhos do emissor com uma probabilidade de P_{rnd}^2 . O primeiro pedido é, portanto, recebido depois do emissor ter enviado $m_0 = \frac{1}{P_{rnd}^2}$ pedidos.

Por tudo isto, um limite inferior para o número total de pedidos necessários para determinar o emissor é

$$m \geq \frac{1}{P_{rnd}^2} + \sum_{i=1}^l \frac{1}{P_{rnd}^i}$$

2.5.5 Hot Path Routing

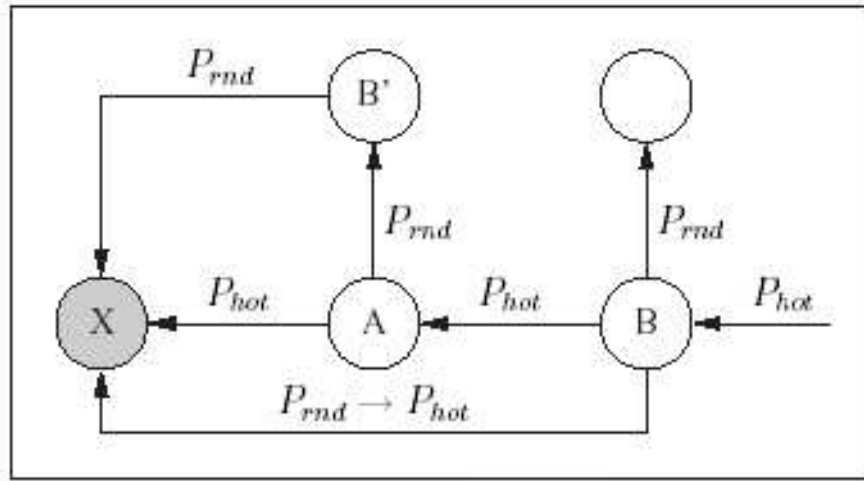
Através do *hot path routing*, os nós que vão receber um pedido são seleccionados pelo número de respostas válidas a pedidos recentes.

Os vizinhos que respondem mais frequentemente terão mais probabilidade de receber mais pedidos.

A probabilidade que um nó A tem de seleccionar o seu vizinho I é $P_{hot}(A, I) = \frac{r}{q}$, onde q é o número de pedidos que o nó A enviou recentemente (em termos temporais, num determinado intervalo de tempo) a I e r é o número de respostas válidas. Daí, se o atacante é um "hot node", o atalho para o ataque torna-se consideravelmente mais fácil:

- Mais pedidos são reencaminhados para o atacante e portanto menos pedidos são necessários no total. Se um nó está num "hot path", então $P_{hot} > P_{rnd}$, caso contrário $P_{hot} \approx P_{rnd}$.
- Devido ao routing aleatório, o "hot path" não consegue comportar todos os pedidos, atribuindo-os a nós vizinhos que não estão no "hot path".

Ambas as propriedades ajudam o atacante a determinar o nó emissor.



Esta figura mostra um “hot path” onde o atacante é um “hot node”. Neste caso, o atacante pode testar eficientemente que vizinho do nó a está perto do emissor.

Análise à tolerância a ataques

Para testar quais dos vizinhos B_i do actual nó precedente A está mais perto do emissor, compara-se o número de pedidos recebidos desses nós. Seja q_A o número de pedidos relacionáveis que o atacante X recebeu de A . Aí o atacante pode esperar receber o mesmo número de pedidos relacionáveis de um nó testado B no mesmo intervalo de tempo. Se o nó testado B estiver mais perto do emissor, o atacante está à espera de receber

$$q_B = q_A \cdot \frac{P_{rnd}(B)}{P_{hot}(A, X) \cdot P_{hot}(B, A)} \approx q_A \cdot \frac{P_{rnd}}{P_{hot}^2}$$

pedidos relacionáveis deste nó. Além disso, como mostrado na figura anterior, a probabilidade de receber localmente vai aumentar ao longo de tempo para P_{hot} , visto que o nó B apreende que o atacante é um “hot node”, enquanto o nó A pode não ser capaz de responder aos pedidos de B sem a ajuda do atacante. Por outro lado, se o nó testado está mais distante do emissor, o atacante apenas espera receber

$$q'_B = q_A \cdot \frac{P_{rnd}(A) \cdot P_{rnd}(B)}{P_{hot}(A, X)} \approx q_A \cdot \frac{P_{rnd}^2}{P_{hot}}$$

pedidos relacionáveis deste nó.

Para que este teste simplificado funcione, é importante que

$$\frac{P_{rnd}}{P_{hot}^2} \gg \frac{P_{rnd}^2}{P_{hot}}$$

Visto que $P_{hot} \geq P_{rnd}$ e P_{rnd} são pequenos, o nó que está perto do emissor pode ser determinado de forma eficiente.

Número de pedidos necessários

Para fazermos uma estimativa de quantas mensagens o emissor pode enviar sem ser identificado, assume-se que P_{rnd} e P_{hot} são iguais para todos os nós do GUNet. Seja l o tamanho do “hot path”:

- O atacante irá receber pedidos vindos do nó precedente com uma probabilidade de P_{hot}^l . O primeiro pedido é portanto recebido depois do emissor ter enviado $m_l \geq \frac{1}{P_{hot}^l}$ pedidos.
- O atacante vai receber pedidos do nó intermediário i no “hot path” com uma probabilidade de $P_{hot}^{i-1} P_{rnd}$, onde $1 \leq i \leq l - 1$. O primeiro pedido é portanto recebido depois do emissor ter enviado $m_i \geq \frac{1}{P_{hot}^{i-1} P_{rnd}}$ pedidos.
- O atacante vai receber pedidos por vizinhos do emissor com uma probabilidade de P_{rnd}^2 . O primeiro pedido é portanto recebido depois de o emissor ter enviado $m_0 = \frac{1}{P_{rnd}^2}$ pedidos.

Resumindo, um limite inferior no número total de pedidos necessários para determinar o emissor é

$$m \geq \frac{1}{P_{hot}^l} + \frac{1}{P_{rnd}^2} + \frac{1}{P_{rnd}} \cdot \sum_{i=0}^{l-2} \frac{1}{P_{hot}^i}$$

2.5.6 Parametrizando o ataque – soluções para o futuro

Estivemos a falar de várias possibilidades de ataque ao anonimato do GUNet revelando o emissor de um certo conteúdo na rede usando o sistema de routing do GUNet, estimando o número de pedidos necessários para encontrar o emissor desses mesmos pedidos. Estes “limites mínimos necessários” são apenas uma estimativa, imprecisa, devido a divertos factores (como a não consideração dos TTL, das prioridades nos pedidos, a possibilidade de *loops*, *et cetera*).

Vamos agora tentar discutir a eficiência do ataque descrito na prática. Assumiremos parâmetros mais realistas: a distância inicial do atacante ao emissor é $l = 6$ e cada nó na rede usa $P_{hot} = 0.9$ e $P_{rnd} = 0.1$.

Se o reencaminhamento aleatório está a ser usado, o GUNet é seguro, visto que o atacante está relativamente longe do emissor e recebe cada pedido enviado pelo emissor apenas com uma probabilidade de 10^{-6} . Daí, é muito improvável que o atacante consiga identificar um *download*. A questão é, no entanto, se o caminho mais curto do emissor ao atacante é realmente relativamente longo.

Seja n o número total de nós, c o número de atacantes, e k o número de vizinhos. Assumindo que cada nó escolhe os seus vizinhos aleatoriamente, então a probabilidade de um nó se conectar a i atacantes é

$$p_i = \frac{\binom{c}{i} \binom{n-c}{k-i}}{\binom{n}{k}}$$

Para $k \ll n$ a probabilidade de um nó não estar conectado a um atacante pode ser simplificada para $p_0 = (1 - \frac{c}{n})^k$. A não ser que a fracção $\frac{c}{n}$ seja negligenciável, o número de vizinhos tem um grande impacto no tamanho do caminho mais curto para um atacante. É interessante verificar que tendo poucos vizinhos não só aumenta o tamanho do caminho, mas também aumenta o P_{rnd} , o que tem dois efeitos: por um lado torna-se mais difícil para um atacante testar os nós vizinhos, mas por outro lado mais pedidos são redireccionados para o atacante, o que o ajuda a não ser que $P_{rnd} = 1$.

Por isso, parece importante que a escolha dos vizinhos feita por cada nó não seja feita de forma aleatória, mas usando uma métrica de confiança²³ para minimizar o risco de conexão com um atacante. Além disso, este tipo de métricas tornam os próprios ataques descritos muito menos improváveis, visto que um nó pode rejeitar ligações de nós dos quais não temos confiança.

Enquanto a segurança do reencaminhamento aleatório continua a ser controversa, o ataque aqui é sucedido, se um atacante tiver hipótese de explorar um “hot path”. Com os parametros do exemplo acima, são precisas apenas 82 multiqueries para determinar o emissor e portanto fazer o download de um ficheiro de aproximadamente 2 MB já é potencialmente perigoso.

Mesmo que uma implementação do ataque precise realmente um múltiplo do número de pedidos estimado, isto mostra que o “hot path routing” é potencialmente uma fraqueza. A melhor estratégia para um atacante é popular a rede GUNet com vários nós com grande largura de banda. Visto que os “hot paths” deixam sempre alguns pedidos para os nós vizinhos, o atacante pode de forma bastante eficiente testar que nós estão no “hot path”, até que o emissor seja descoberto.

Resumidamente, o “hot path routing” não só melhora a performance do GUNet, mas também aumenta a eficiência deste tipo de ataques. Visto que o GUNet é bastante ineficiente sem o “hot path routing”, sugere-se uma melhoria (a ser implementada futuramente) de forma a tornar os “hot paths” mais seguros:

- Para prevenir o ataque mais complexo referido anteriormente, os “hot paths” não devem deixar pedidos para nós que não estejam no “hot path”. Assim, os “hot paths” têm de ser muito mais estáticos, e todos os nós do “hot path” têm de ter conhecimento de que pedidos são relacionáveis.

²³como as descritas por Raph Levien e Alex Aiken no artigo *Attack-resistant trust metrics for public key certification* presente nos proceedings do sétimo *USENIX Security Symposium*, de 1998

- Mesmo com os “hot paths” mais estáticos como apresentados no ponto anterior, o ataque básico (o primeiro a ser descrito) pode ser usado para descobrir o emissor. Visto que este ataque necessita de créditos, deve-se realçar que o atacante ganha esses créditos respondendo a pedidos.

Conclui-se então que os “hot paths” deverão ser bastante estáticos, ou seja, não devem ser possíveis atalhos nos “hot paths”.

Até que uma implementação como a descrita esteja concluída, deve-se ter uma implementação sem recorrer ao “hot path routing”, a custo de eficiência, para que não se tenha falhas quanto ao anonimato.

2.6 Autenticação/Não-Repúdio

O GUNet fornece provas de integridade e de origem dos dados, isto é, ambos os nós de uma comunicação estabelecem uma relação genuína.

Tecnicamente, o não-repúdio é assegurado através do uso de uma chave pública que é usada para validar uma assinatura digital. Esta assinatura terá obrigatoriamente de ter sido feita pela chave privada correspondente, que apenas o emissor possui.

2.7 Accounting

O GUNet é baseado num sistema económico, onde cada nó associa cada outro nó conhecido a um valor local chamado crédito. O objectivo deste sistema económico é impedir ataques de flooding limitando os recursos disponíveis a um atacante:

Sejam A e B dois nós. A associa B com um crédito c_B e vice-versa. Depois de B receber uma query com prioridade p de A , verifica em primeiro lugar se A tem crédito suficiente.

- Se $c_A = 0$ o query é ignorado, caso contrário a prioridade p' passa a ser p .
- Se $p' > c_A$ a prioridade da query é reduzida para $p' = c_A$.

Depois B processa a query com a prioridade $p = p'$ e encarrega A de reduzir o seu crédito para $c_A = c_A - p$. No entanto, se B tem largura de banda excessiva, pode decidir não encarregar A . A decisão de quando e quanto B diminui o crédito de A é invisível para A . Se B devolve uma resposta válida, B espera que A aumente o seu crédito para $c_B = c_B + p$. Novamente, A pode decidir não doar a B por retornar uma resposta válida. A decisão de quando e quanto A aumenta o crédito de B é invisível para B .

2.7.1 Queries de prioridade zero

Se um nó redirecciona uma query para outro nó sem para isso doar ao nó precedente, pode atribuir à query prioridade zero, porque este não quer ser doado pelo nó seguinte. Logo, um nó que recebe uma query de prioridade zero não pode ganhar nenhum crédito ao responder a esta query. Mas se a carga no nó for baixa, responder a uma query de prioridade zero não degrada este nó.

2.8 Collections

Uma collection é um namespace gerido automaticamente.

Um namespace é um conjunto de ficheiros que foram assinados pelo mesmo pseudónimo.

Um pseudónimo é essencialmente uma chave RSA (pública-privada). De notar que um pseudónimo não está limitado por nó. Podem existir múltiplos pseudónimos para cada utilizador, e os utilizadores podem partilhar chaves dos seus pseudónimos.

Ficheiros que foram assinados e colocados num namespace podem ser actualizados. As actualizações são identificadas como autênticas se a mesma chave secreta foi usada para assinar a actualização. Os namespaces são também úteis para estabelecer uma reputação, visto todo o conteúdo do namespace provir da mesma entidade (o que não quer dizer que seja a mesma pessoa).

A raiz do namespace aponta para uma directoria contendo todos os ficheiros inseridos pelo utilizador desde a inicialização da collection. A raiz é actualizada esporadicamente de cada vez que um novo ficheiro é adicionado. A criação da directoria e a actualização do namespace são efectuadas automaticamente em cada inserção e não requer qualquer intervenção por parte do utilizador.

As collections são anunciadas com a palavra-chave "collection".

Para iniciar uma collection basta fazer:

```
$ gnunet-pseudonym -a -C NICKNAME -D DESCRIPTION -r AUTHORNAME
$ gnunet-insert FILENAME
$ gnunet-search collection
$ gnunet-gtk
```

No interface selecciona-se "Advanced-Search Namespace".

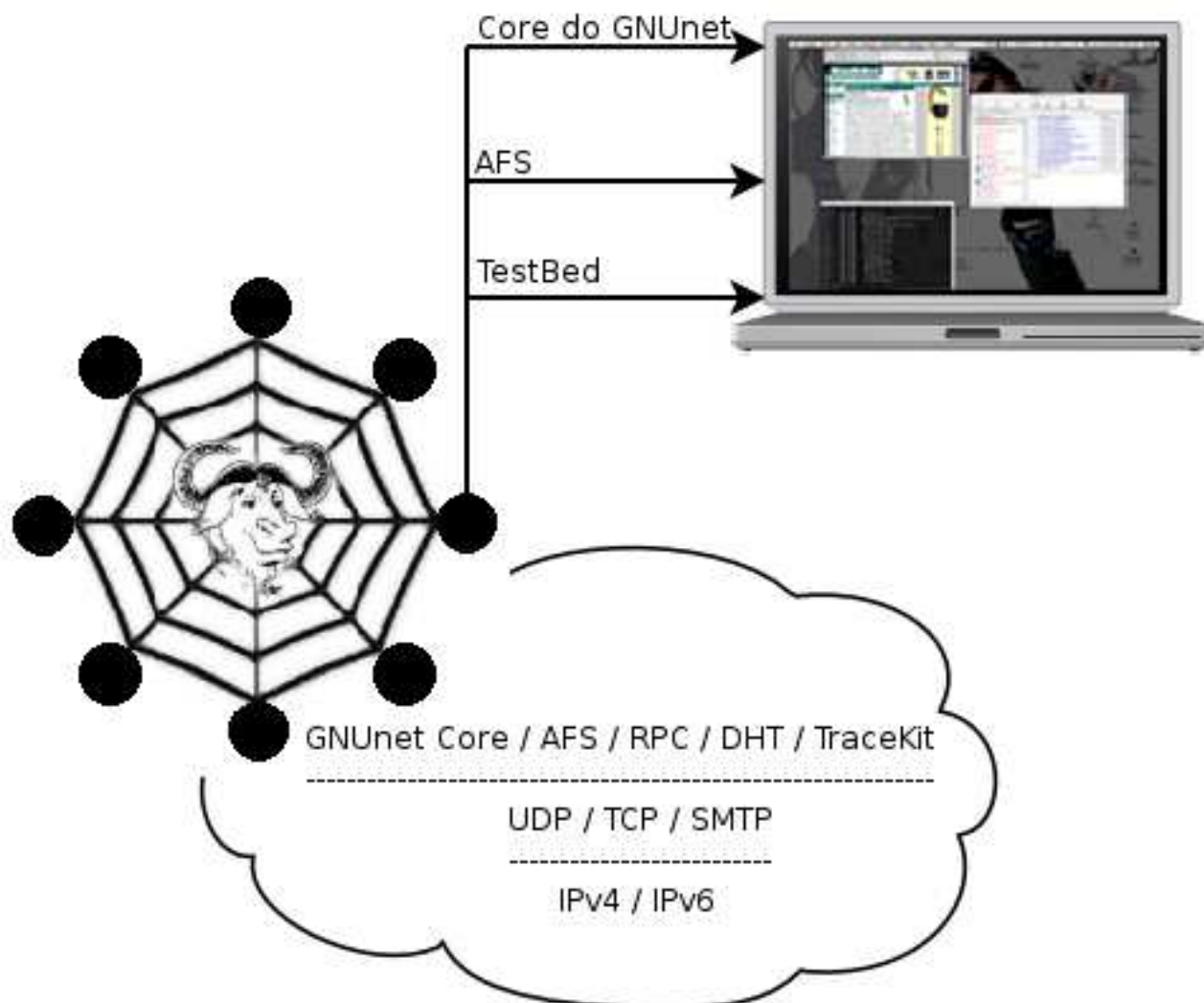
Na caixa de diálogo, selecciona-se o NICKNAME da lista de identificadores de namespaces. O identificador da chave de procura será preenchido automaticamente de forma a que este aponte para a raiz do namespace. A procura deverá retornar uma directoria que conterà o ficheiro FILENAME. Depois de adicionar ficheiros adicionais, directorias adicionais com mais ficheiros irão aparecer na procura.

Para terminar a collection, usa-se:

```
$ gnunet-pseudonym -E
```

2.9 Protocolos de Comunicação

Descrevem-se nesta secção os Protocolos de Comunicação existentes no GNUet: tanto *Peer-to-Peer* (a comunicação feita entre os nós da rede GNUet), Cliente-Servidor (a comunicação feita entre as aplicações que usam a framework GNUet e os nós da rede GNUet) e os Serviços de Transporte suportados para toda a comunicação.



Esta imagem representa um possível computador a usar várias aplicações que recorrem à Framework do GNUet.

Através dos protocolos para a comunicação Cliente-Servidor, essas aplicações comunicam com a rede GNUet através das mensagens definidas no Núcleo do GNUet, mensagens AFS ou as mensagens de TestBed, explicadas posteriormente.

Depois disso, os nós comunicam entre si usando as metodologias explicadas de seguida.

Finalmente, todas essas comunicações são feitas recorrendo aos Serviços de Transporte descritos no final desta secção.

2.9.1 Peer-to-Peer

Como regra geral para a comunicação inter-nodal, qualquer nó pode recusar uma resposta a qualquer momento. Isto pode ser resultado de perda de pacotes, porque o nó está ocupado, ou porque o nó recusa uma resposta por motivos de desconfiança (ou qualquer combinação destes).

As asserções subjacentes dos protocolos são as seguintes:

- Todos os nós possuem um relógio sincronizado (UTC), mas pequenas variações têm de ser toleradas (até alguns minutos). O GUNet não contém meios para fazer a sincronização dos relógios;
- Qualquer nó pode ser malicioso e falhar a qualquer momento; a rede pode não ser de confiança e não fornece qualquer tipo de autenticação;
- As chaves privadas dos nós são sempre secretas e a implementação de primitivas criptográficas é robusta;

Camada UDP: Protocolos de confidencialidade e autenticação

Para explicar os protocolos de confidencialidade e autenticação, vamos descrever as mensagens de HELO e SKEY do protocolo GUNet. É suposto estas estabelecerem um canal de comunicação confidencial e autenticado.

- HELO

Este protocolo descreve como é que um nó se liga à rede GUNet. É assumido que o nó conhece pelo menos outro nó participante da rede. Isto é conseguido fornecendo ao nó o IP e o porto de outro nó. Considere-se A o novo nó e B um nó já existente.

- $A \rightarrow B : E_B(A, S_A(@A, t))$ onde A é a chave pública de A , S_A é a mensagem assinada pela chave privada de A , $@A$ é o endereço de rede actual de A e t é o tempo de duração deste endereço até se tornar inválido. E_B é a encriptação com a chave pública de B ; B é conhecido à priori.

Primitivas criptográficas: Criptografia com chaves públicas, 2048 bits, RSA, a mesma chave para assinar e encriptar.

- B verifica a assinatura e adiciona A à sua lista de nós conhecidos bem como o endereço correspondente de A , o *timestamp* e a assinatura. Mais tarde, B pode decidir dizer a outro nó da existência de A . O timestamp t que A usou na mensagem original limita o espaço temporal nas quais estas propagações são permitidas (A pode necessitar de alterar $@A$).

– $B \rightarrow C : E_C(A, S_A(@A, t))$.

- SKEY

Após A e B terem trocado as suas chaves publicas através do protocolo HELO, podem trocar as chaves de sessão K . Pode ser inicializado por A ou B :

– $B \rightarrow A : B, E_A(K), t, S_B(H(E_A(K), t))$ onde K é a chave de sessão que foi escolhida por B e foi criada na data t .

– $A \rightarrow B : E_K(H(K))$ (acknowledgement)

A e B guardam K na sua tabela de conexões actual. Comunicações futuras são encriptadas com K :

– $A \rightarrow B : E_K(M, t)$

– $B \rightarrow A : E_K(M, t)$

Cada mensagem contém um *timestamp* t que descreve quanto tempo esta mensagem é suposto ser válida. Desta forma um atacante não causa grandes danos através de ataques de *replay*.

GProxy

O GProxy é um serviço por cima da camada básica do GUNet. O GProxy permite aos utilizadores partilharem ficheiros. Os objectivos centrais da GProxy são:

- Anonimato: tanto o emissor como os nós intermediários não conseguirão descobrir o que o receptor está à procura (excepto quando conseguem adivinhar).
- Distribuição: nenhum nó deve ser forçado a partilhar um ficheiro completo, pequenas partes de $1K$ devem ser suficientes.
- Segurança: o nó que armazena os dados não poderá de maneira nenhuma desencriptar os dados, podendo assim negar o conhecimento dos seus conteúdos.
- Eficiência: mesmo que encriptados, os mesmos conteúdos (eventualmente inseridos independentemente por dois nós sob diferentes chaves) devem dizer respeito aos mesmos nomes de ficheiros com o mesmo conteúdo, reduzindo assim requisitos de armazenamento.
- Encriptação on-the-fly: deve ser possível partilhar partes de ficheiros não encriptados do disco, sem ter para isso de encriptar o ficheiro completo todas as vezes.
- Inserção de conteúdo:

A inserção de conteúdo descreve os passos necessários a inserir conteúdo num nó local. Não descreve formas de distribuir o conteúdo através da rede ou como o descarregar.

- O utilizador fornece à *proxy* local o conteúdo de C , uma lista de palavras-chave K e uma descrição D (e opcionalmente o pseudónimo P) a usar.
- A *proxy* separa C em blocos B_i , cada um ocupando $1K$, e computa os valores da *hash* $H_i = H(B_i)$ e $H_i^2 = H(H(B_i))$. *Padding* aleatório é adicionado quando necessário.
- Depois é encriptado cada bloco ($1 - keycrypto$) resultando $E_i = E_{H_i}(B_i)$.
- A *proxy* armazena E_i sob o nome H_i^2
- Se houver mais do que um H_i , a *proxy* agrupa $51H_i$ valores com um CRC32 dos dados originais a um novo bloco de $1K$. *Padding* aleatorio é adicionado quando necessário. Todos os blocos de $1K$ obtidos através deste método são então tratados tal como em 2.
- Se houver apenas um *hashcode* H_1 , a *proxy* contrói um nó-raiz, contendo H_1 , a descrição D , o comprimento original de C , um *checksum* CRC e um P opcional e uma assinatura. Tudo isto tem de ser inferior a $1K$ (o comprimento da descrição pode ser diminuído se necessário). O nó-raiz resultante R é *padded* e encriptado para cada *keyword* K resultando $R_K = E_{H(K)}(R)$.
- Finalmente, para cada K , o resultado R_K é armazenado sob $H(H(K))$.

2.9.2 Comunicação Cliente-Servidor

Esta secção descreve o protocolo que é usado nas conexões TCP fiáveis entre o *gnunetd*²⁴ e os clientes (ou seja, programas que usem a *framework* do GNUet, como o *gnunet-gtk*²⁵).

O TCP é orientado a ligações, mas o GNUet quebra as ligações em registos.

As diferentes mensagens entre o GNUet e os clientes têm todas o seguinte formato²⁶:

00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
tamanho do registo(NBO)	tipo de mensagem CS(NBO)
(tamanho -4) bytes de dados	

As constantes do tipo de mensagem estão definidas no ficheiro `gnunet_util.h` e todas têm nomes começados por `CS_PROTO_`. O sufixo `REQUEST` é usado para mensagens do cliente para o *gnunetd*. O `REPLY` é usado para mensagens do *gnunetd* para o cliente. Visto o *gnunetd* não dever enviar dados sem terem sido pedidos antes, o cliente deve simplesmente ignorar as mensagens de `REPLY` que não tenham sido pedidas. Se o *gnunetd* receber um `REQUEST` que não entende, fecha a ligação (para prevenir o cliente

²⁴*gnunetd* é o *daemon* do *gnunet* - o nó

²⁵o *gnunet-gtk* é o cliente para o sistema de partilha de ficheiros do GNUet

²⁶Quando nestas representações se lê "NBO" estamos a referir-nos à "network byte order". Antes de ler estes campos ele devem ser convertidos para o "host byte order" usando o `htons` para valores de dois *bytes* e o `htonl` para valores de quatro *bytes*. Antes de escrever nestes campos, os valores deverão ser convertidos usando as funções `ntohs` e `ntohl` respectivamente.

de bloquear numa resposta que poderá nunca vir). A implementação das condições descritas encontra-se no ficheiro `tcpio.c`.

As mensagens cliente-servidor podem dividir-se em três grupos, descritos de seguida.

Mensagens de núcleo do GNUet

As únicas mensagens independentes da aplicação e sempre disponíveis entre o cliente e o servidor são as seguintes:

- RETURN_VALUE

A mensagem RETURN_VALUE é usada para comunicar valores de retorno simples (inteiros) a pedidos TCP. É sempre enviada em resposta a outro pedido, específico. O formato da mensagem RETURN_VALUE é:

00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
8 (tamanho)(NBO)	0 (tipo)(NBO)
o valor de retorno(NBO)	

- GET_STATISTICS

Com esta mensagem, o cliente pode pedir estatísticas ao servidor. O servidor responde sempre com uma mensagem do tipo STATISTICS. O formato da mensagem GET_STATISTICS é:

00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
4 (tamanho)(NBO)	1 (tipo)(NBO)

- STATISTICS

A informação estatística disponível poderá mudar entre versões diferentes do GNUet. Depende ainda de que módulos estão carregados e a ser usados. O formato desta mensagem é:

00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
variable (tamanho)(NBO)	2 (tipo)(NBO)
hora de início (início do gnetd)(NBO)	
hora de início (início do gnetd)(NBO)	
contadores totais (número de valores estatísticos que dos quais o gnetd controla)(NBO)	
contadores de estado (número de valores estatísticos transmitidos nesta mensagem)(NBO)	
valores (unsigned long long[contadores de estado])(NBO)	
descrições (contadores de estado de strings terminadas em zero descrevendo os valores)(NBO)	

O servidor envia potencialmente várias mensagens do tipo STATISTICS até o total dos contadores de estado em todas as mensagens igualar o total de contadores

anunciado. A última string de descrição também é terminada por zeros. Por isso o último carácter em cada mensagem de `STATISTICS` é 0.

- `TRAFFIC_QUERY`

Com esta mensagem, o cliente pode pedir estatísticas de tráfego ao `gnunetd`. O período de tempo especificado na mensagem especifica que tráfego é interessante para um pedido (em termos de não ser mais antigo que o período de tempo das unidades de tempo).

O servidor responde com uma mensagem do tipo `TRAFFIC_INFO`.

- `TRAFFIC_INFO`

Com esta mensagem, o servidor envia informação de tráfego para o cliente.

- `SHUTDOWN_REQUEST`

Com esta mensagem, o cliente pode pedir ao `gnunetd` para se desligar.

O servidor responde com uma mensagem do tipo `RETURN_VALUE` com o conteúdo `OK` se se for desligar.

- `GET_OPTION_REQUEST`

Com esta mensagem, o cliente pode pedir o valor de uma opção da configuração do `gnunetd`.

O servidor responde com uma mensagem do tipo `GET_OPTION_REPLY`.

- `GET_OPTION_REPLY`

Com esta mensagem, o servidor notifica o cliente do valor de uma opção.

Mensagens do AFS

O protocolo do GNUet para comunicações cliente-a-nó define actualmente as seguintes mensagens para o sistema de Partilha Anónima de Ficheiros:

- `QUERY`

O cliente envia uma mensagem deste tipo quer esteja a pesquisar ou a descarregar. Espera-se que o servidor faça a pesquisa e devolva os resultados ao cliente. O servidor pode enviar qualquer número de resultados com qualquer atraso (se bem que tipicamente não existem resultados depois de TTL segundos e o cliente volta a fazer o pedido), incluindo nenhum.

- `RESULT_3HASH`

Se o `gnunetd` encontrar o resultado de uma pesquisa feita por um `QUERY`, devolve os dados obtidos encapsulados nesta mensagem. A mensagem contém a *hash* dupla do pedido para que o resultado seja identificado.

- `RESULT_CHK`
Se o `gnunetd` encontra um resultado de um download pedido por um `QUERY`, devolve os dados obtidos encapsulados nesta mensagem.
- `INSERT_CHK`
Os clientes enviam esta mensagem para inserir resultados de uma pesquisa na rede. O servidor é suposto guardar os dados e partilhá-los com outros nós. De notar que existem duas formas de partilhar dados, via inserção e via indexação. A indexação apenas se aplica a *downloads* e nunca a resultados de pesquisa. A importância desta mensagem diz ao servidor quão importante é guardar este conteúdo, para que o servidor possa decidir que conteúdo descartar se estiver a ficar sem espaço disponível. Se uma inserção for bem sucedida, o servidor responde com uma mensagem do tipo `RETURN_VALUE` contendo a mensagem `OK`, caso contrário responderá com `YSERR`.
- `INSERT_3HASH`
Os clientes enviam esta mensagem para inserir resultados de pesquisas na rede. O servidor supostamente irá guardar esses dados e partilhá-los com outros nós. A importância desta mensagem diz ao servidor quão importante é guardar este conteúdo, para que possa decidir que conteúdo descartar se estiver a ficar sem espaço. Se a inserção for bem sucedida, o servidor responde com uma mensagem do tipo `RETURN_VALUE` contendo a mensagem `OK`, caso contrário responderá com `YSERR`.
- `INDEX_BLOCK`
Os clientes enviam esta mensagem para inserir conteúdo na rede. A mensagem de indexação não contém os dados mas descreve uma forma do servidor os encontrar "on-demand". A estrutura `ContentIndex` contém um índice para a lista de nomes de ficheiros e o offset no ficheiro onde o bloco correspondente ao pedido pode ser encontrado. Se a inserção for bem sucedida, o servidor responde com uma mensagem do tipo `RETURN_VALUE` contendo a mensagem `OK`, caso contrário responderá com `YSERR`.
- `INDEX_FILE`
Esta mensagem é usada pelo cliente para adicionar um ficheiro à lista de ficheiros partilhados. O `gnunetd` responde a este pedido com um índice na lista de ficheiros que pode ser usado nas mensagens do tipo `CS_INDEX_BLOCK`. O ficheiro é identificado pela hash `RIPE160MD` dos conteúdos. DE notar que espere-se que o cliente copie ou faça um atalho do ficheiro para a directoria de indexação com as mensagens apropriadas. Tentar criar um atalho deverá ser feito antes do `INDEX_FILE`, mas fazer uma cópia é algo que deve ser feito depois. O servidor responde com o valor `YSERR` se os parâmetros de quota (`INDEX_QUOTA`) não permitirem o upload do ficheiro. O cliente deverá usar o tamanho 0 se um atalho simbólico for usado e por isso a verificação da quota puder ser dispensada.

- INDEX_SUPER

Esta mensagem adiciona um hash-code a um dos bloom filters, tornando o conteúdo verdadeiramente disponível.

- DELETE_CHK

A mensagem CS_DELETE_CHK usa o mesmo formato que a mensagem CS_INSERT_CHK, mas o tipo é marcado como 16. O módulo de AFS ao verificar isso apaga o bloco CHK correspondente. É enviada então uma mensagem RETURN_VALUE para indicar sucesso ou não.

- DELETE_3HASH

A mensagem CS_DELETE_3HASH usa o mesmo formato que a mensagem CS_INSERT_CHK, só que o tipo é marcado como 17. O módulo de AFS ao verificar isso apaga o bloco correspondente. É enviada então uma mensagem RETURN_VALUE para indicar sucesso ou não. Esta mensagem actualmente ainda não é usada porque ainda não existe uma boa forma de descobrir os RBlocks que correspondem a um determinado ficheiro quando estamos perante palavras-chave e meta-dados fornecidos pelo utilizador.

- UNINDEX_BLOCK

Esta mensagem usa o mesmo formato que a mensagem CS_INDEX_BLOCK, mas o tipo é marcado como 18. O módulo de AFS ao verificar isso remove a entrada correspondente da base de dados. É enviada então uma mensagem RETURN_VALUE para indicar sucesso ou não.

- UNINDEX_FILE

Esta mensagem usa o mesmo formato que a mensagem CS_INDEX_FILE, mas o tipo é marcado como 19. O módulo de AFS ao verificar isso remove o ficheiro correspondente da lista de ficheiros. É enviada então uma mensagem RETURN_VALUE para indicar sucesso ou não.

- UNINDEX_SUPER

Esta mensagem usa o mesmo formato que a mensagem CS_INDEX_SUPER, mas o tipo é marcado como 20. O módulo de AFS ao verificar isso remove a hash correspondente do bloomfilter para as super-hashes. É enviada então uma mensagem RETURN_VALUE para indicar sucesso ou não.

- NSQUERY

Esta mensagem é usada pelos clientes para iniciar uma pesquisa num namespace. Difere dos pedidos normais por também conter o identificador do namespace N . Além disso, um NSQUERY pode apenas conter um identificador pedido. O SBlock identificador R que é usado num NSQUERY é derivado de um identificador de namespace N e a chave K que é usada para encriptar o SBlock correspondente através da seguinte fórmula:

$R := H(K) \text{ XOR } N$

- INSERT_SBLOCK

Esta mensagem é usada pelos clientes para inserir um SBlock na rede.

De notar que actualmente ainda não existe a mensagem correspondente para apagar um SBlock.

- RESULT_SBLOCK

Esta mensagem é enviada pelo servidor para o cliente sempre que uma resposta a um NSQUERY é encontrada.

De notar que actualmente ainda não existe a mensagem correspondente para apagar um SBlock.

- UPLOAD_FILE

Esta mensagem é usada por um cliente para fazer upload de uma parte de um ficheiro a ser indexado. O ficheiro é identificado pela sua hash RIPE160MD. Cada mensagem de upload contém um número variável de bytes começando a partir de um certo offset. O gnetd irá guardar o bloco respeitante ao ficheiro na directoria de indexação. De notar que o cliente ainda tem de enviar os pedidos de indexação para blocos individuais. O gnetd assinala sucesso ou erro através de uma mensagem do tipo RETURN_VALUE.

- LINK_FILE

Em vez de criar uma cópia, tenta criar um link simbólico. O nome de ficheiro dado é o nome de destino. O gnetd deve verificar se o ficheiro está acessível e se os conteúdos correspondem à hash RIPE160MD dada. O gnetd assinala sucesso ou erro através de uma mensagem do tipo RETURN_VALUE.

- GET_AVG_PRIORITY

Este pedido é usado pelo cliente para determinar a prioridade média actual dos pedidos de outros nós que estão na tabela de reencaminhamento. O servidor responde com uma mensagem do tipo RETURN_VALUE contendo a prioridade média (ou SYSERR em caso de erro).

Mensagens do Testbed

O protocolo de *Testbed* encapsula todas as suas mensagens em dois tipos de mensagens, CS_TESTBED_REQUEST e CS_TESTBED_REPLY. De notar também que o módulo de *testbed* não define por si só mensagens nó a nó. O protocolo de *testbed* é usado para instrumentalizar a rede *peer-to-peer*, mas não usa os protocolos *peer-to-peer* para o atingir. A configuração do *testbed* é por si só inerentemente cliente-servidor.

2.9.3 Serviços de Transporte

O GNUnet usa um tipo especial de mensagens de forma a fazer corresponder chaves públicas ao seu endereço correspondente. Para as camadas de transporte do TCP e UDP, um endereço é um IP e um porto. Outros mecanismos de transporte (HTTP, SMTP, etc) dispõem de outros tipos de endereços. De notar que qualquer nó pode ter endereços múltiplos para os mais variados mecanismos de transporte.

A camada de transporte tem, como o protocolo IP, semânticas de melhor-esforço. Não existe garantia de que uma mensagem seja entregue. Descrevere-mos de seguida como escrever o seu próprio mecanismo de transporte para a framework do GNUnet.

A melhor forma de implementar um novo mecanismo de transporte é começar com código existente que seja semanticamente o mais próximo possível do novo mecanismo. O critério mais importante é saber se a conexão é *stateful* e bidirecional (TCP, HTTP) ou *stateless* e unidirecional (UDP, SMTP). Visto a fiabilidade e latência serem geridos pelas aplicações, estes critérios são irrelevantes ao escolher uma implementação existente.

Todos os mecanismos de transporte têm de definir um número identificativo no ficheiro `ports.h`. De notar que este número é usado apenas internamente pelo GNUnet e não tem de corresponder ao protocolo subjacente de forma alguma. É claro que ajuda aos utilizadores se puderem saber que os transportes 25, 17 ou 6 corresponderem aos já conhecidos portos do protocolo IP. Para além da mensagem propriamente dita, o mecanismo de transporte necessita de comunicar:

- A identidade do emissor (a hash da chave pública)
- Se a mensagem está ou não encriptada
- O CRC da mensagem
- O tamanho da mensagem

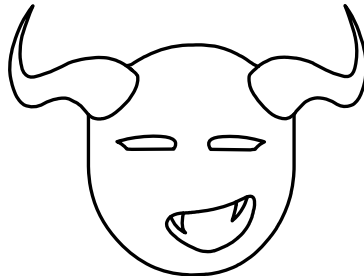
A implementação da camada de transporte define um formato apropriado para para os dados a transmitir nesta mensagem. O formato de encapsulamento pode ser diferente para cada tipo de transporte. Dependendo da implementação do mecanismo de transporte, nem todos estes necessitam de ser transmitidos explicitamente em cada mensagem. Por exemplo, um transporte *stateful* como o TCP pode enviar apenas a identidade do emissor uma vez e usar mensagens especiais para trocar entre mensagens encriptadas e não encriptadas.

Capítulo 3

Aplicações

Sendo as potencialidades do GNUet tão vastas, rapidamente apareceu um enorme conjunto de projectos que se prendem, pelo menos em conceito, não só à implementação do mais variado tipo de software para o uso do GNUet, como principalmente em tirar partido das potencialidades que esta *framework* lhes permite, não tentando modificar aplicações já existentes para usar o GNUet mas sim desenhá-las atendendo a este novo paradigma.

3.1 O sistema de partilha de ficheiros



A primeira aplicação desenvolvida para usufruir do GNUet foi um sistema de partilha de ficheiros, o que acabou por contribuir para o desenvolvimento da *framework* em si, não só a nível tecnológico como também porque dando um uso prático ao GNUet acabou por oferecer-lhe uma enorme base de utilizadores e programadores.

Os maiores avanços tecnológicos que este sistema ofereceu ao GNUet foi a nível protocolar, principalmente pela criação do AFS, um protocolo de *routing* anónimo usado acima da rede GNUet.

Apresentamos de seguida uma breve tabela comparativa entre o sistema de partilha de ficheiros do GNUet e outros sistemas peer-to-peer:

Rede	GNUnet	Napster	Direct Connect	FastTrack	eDonkey	Gnutella	Freenet
Pedidos Distribuídos	sim	não	hubs	super-nós	DHT (eMule)	sim	sim
Download de várias fontes	sim	não	sim	sim	sim	sim	não
Economia	sim	não	não	não	sim	não	não
Anonimato	sim	não	não	não	não	não	sim
Plataforma	sistemas Un*x	todas	Win32	muitas	muitas	todas	JVM
Protocolo de Transporte	UDP, TCP, SMTP, HTTP	TCP	TCP	UDP, TCP	UDP, TCP	TCP	TCP
Formato do Pedido	palavras-chave ou URI's de AFS	palavras-chave	ficheiro, THEX	ficheiro, SHA	ficheiro, MD4	ficheiro, SHA	chave secreta
Routing	dinâmico (directo, indirecto)	sempre directo	sempre directo	sempre directo	sempre directo	sempre directo	sempre indirecto
Licença	GPL	GPL (knapster)	comercial	GPL (giFT)	GPL (eMule)	GPL (gtk-gnutella)	GPL

Hoje em dia existem três sistemas de topo similares em funcionalidade ao GNUnet. O Napster é um sistema de partilha de ficheiros distribuídos limitado apenas a ficheiros mp3 onde a distribuição é coordenada por um servidor central. O Gnutella é um sistema de partilha de ficheiros baseado no protocolo HTTP desprovido de um mecanismo de procura centralizado ou suporte criptográfico. O Freenet é um sistema de partilha de conteúdos distribuído que faz uso de encriptação nos nós de forma a impedir os servidores de descriptarem o conteúdo que servem.

A maior desvantagem do Napster é o facto de realmente a sua comunicação não ser encriptada, sendo por isso bastante simples interceptar a comunicação. Como o Napster é centralizado, é também extremamente fácil desligar o sistema ou interceptar a informação dos utilizadores. A limitação a mp3's é apenas uma decisão empresarial e não uma limitação técnica. O conteúdo a partilhar não migra automaticamente, pelo que o receptor tem de adicionar conteúdo manualmente para a lista de partilha.

A vantagem da implementação do Napster é o baixo *overhead* do protocolo de distribuição e da rápida e fiável procura. O Napster tem sido usado principalmente como um meio a facilitar a violação de direitos de autor dando a oportunidade aos utilizadores de partilharem música.

No Gnutella a ausência de um mecanismo de procura centralizada e o imenso *overhead* das procuras distribuídas é sem dúvida uma das suas maiores desvantagens. A comunicação não é anónima. O protocolo deixa, por isso, vaziar informação de procura e identifica os nós que contêm a informação. Por outro lado, a ausência de centralização pode ser uma vantagem do Gnutella o que torna mais difícil o ataque à rede. Tal como no Napster, o conteúdo do Gnutella é explicitamente fornecido pelos nós. Como tal, a informação não desaparece magicamente da rede. Como o Napster, o conteúdo não migra automaticamente no Gnutella. O Gnutella tem também alguns problemas com o "*freeloading*"; utilizadores que podem descarregar quantidades enormes de conteúdo

sem para isso contribuírem, fazendo diminuir os recursos de rede e armazenamento da rede sem compensação. Isto pode tornar os nós com mais recursos a tornarem-se servidores centrais, mesmo sendo a rede “descentralizada”.

O Freenet não sofre ps problemas de centralização associados ao Napster. Aliás, o sistema de encriptação impede que servidores individuais identifiquem o conteúdo do dados armazenados ou transmitidos por estes. No entanto, ao contrário do Napster e do Gnutella, é possível que ficheiros armazenados no Freenet desapareçam em detrimento de outros ficheiros sem a intervenção dos utilizadores.

O Freenet tem a vantagem de qualquer servidor individual não ter conhecimento directo do conteúdo dos dados que armazena ou distribui. Pode, por isso, aliviar os administradores destes servidores de responsabilidades. No entanto, a solução do GNUet onde um único nó geralmente não tem como reconstruir o ficheiro (mesmo assumindo que o nó descobre a chave) parece melhor. Se o nó tivesse de pesquisar toda a rede para completar o ficheiro (e descobrir a chave), seria bastante mais difícil desafiar o administrador do nó a armazenar pequenas partes do conteúdo. O Freenet encripta o conteúdo usando chaves que identificam o recurso. Se a chave for conhecida, o ficheiro associado pode ser descriptado. Existem diferentes tipos de chaves no Freenet. Os diversos tipos de chaves são usados para funcionalidades adicionais tais como assinatura de conteúdos, *namespaces* pessoais ou divisão de conteúdos. Visto a estrutura da chave estar exposta directamente ao utilizador, a utilização do sistema requer conhecimentos aprofundados. Pelo que sabemos, apenas os tipos de chave mais simples são usados (i.e *unsigned* e *namespaces* globais) são usados em grande escala. Recentemente, foram introduzidas chaves que permitem a actualização de conteúdos.

As *queries* de procura no Freenet são serializadas. Embora isto reduza o *overhead* do tráfego, aumenta o tempo de procura. Por outro lado, o conteúdo é propagado para trás no caminho de procura. Isto aumenta o anonimato dos participantes; uma comunicação pode ser simplesmente parte do caminho de procura com nenhum dos participantes sendo o receptor ou emissor original. Para caminhos de procura longos, esta propagação pode aumentar dramaticamente o tráfego global da rede. Ao contrário do GNUet, o caminho de propagação é fixo, nós individuais não podem decidir se devem redireccionar a resposta ou fazer um atalho no caminho. O Freenet fornece uma funcionalidade de anonimato semelhante ao GNUet mas usa mais largura de banda para atingir este objectivo. Uma desvantagem significativa da implementação actual do Freenet não permite a partilha directa de ficheiros sem as encriptar e inserir primeiro. Logo, para assegurar a preservação do conteúdo, um administrador de um nó tem de manter uma cópia não encriptada do ficheiro bem como o conteúdo encriptado do servidor de Freenet. Permitir que o servidor de Freenet partilhe ficheiros directamente pode aumentar a estabilidade e disponibilidade do conteúdo dramaticamente, especialmente para conteúdo onde o administrador do nó não ter de temer a interferência de autoridades.

O Freenet encontra-se em desenvolvimento. Versões recentes tiveram problemas com demasiada utilização do CPU. Um dos problemas do Freenet é que se encontra implementado em Java, necessitando que cada nó corra uma Java Virtual Machine (JVM). Os

requisitos de memória de uma JVM é intolerável em muitos nós potenciais.

O Mojo Nation é um sistema de partilha de ficheiros onde os nós necessitam de fornecer largura de banda e espaço de armazenamento de forma a ganhar Mojo, micro-creditos. O Mojo pode então ser usado para requisitar serviços a outros nós. Isto protege a rede de “freeloaders” (pessoas que usam a rede mas não contribuem). Esta metodologia é semelhante ao *ranking* do GUNet, mas não permite a novos utilizadores das capacidades máximas e não fornece anonimato. O Mojo Nation é, como o Napster, um produto comercial. O Site web não fornece qualquer tipo de especificação de como é conseguida a autenticação.

As três implementações sofrem do mesmo problema que consiste no armazenamento de um único ficheiro em toda a rede. Isto é um problema particular para ficheiros grandes o que pode provocar que um servidor forneça uma largura de banda excessiva num único cliente. Mais, a distribuição dos *queries* de procura é um problema comum. O Napster e o Gnutella têm procura por nomes de ficheiros, o que pode não ser apropriado. O Freenet requer o uso de chaves únicas que podem não ser triviais de reconhecer. Os servidores de chaves (*keyservers*) na rede Freenet tentam resolver este problema fornecendo índices para todas as chaves disponíveis. A desvantagem dos *keyservers* é que estes têm de ser administrados; e muitas vezes acontece que estes indexam conteúdo que já não se encontra disponível.

Até agora, pelo que sabemos, o GUNet é o primeiro sistema que permite *queries* booleanas e fornece anonimato. Finalmente, nenhuma das redes pode dar qualquer garantia de quanto tempo o conteúdo vai estar disponível depois do nó que inseriu o conteúdo inicialmente se desligar da rede. Para o Napster e o Gnutella, isto geralmente significaria o fim da disponibilidade deste conteúdo no sistema. Mesmo que o conteúdo tenha migrado para outros servidores, a decisão de apagar o conteúdo pode surgir a qualquer momento visto a substituição de conteúdo seguir o algoritmo de LRU¹. O GUNet permite que nós operem um pouco melhor sendo conscientes do conteúdo que armazenam. Como o conteúdo raro é geralmente mais valioso, os nós podem assim otimizar as suas políticas de armazenamento. No entanto, heurísticas para isto necessitam ainda de ser desenvolvidas. Alcançar garantias de que o conteúdo seja preservado é algo que foi pesquisado noutros projectos. Por exemplo, o projecto FreeHaven usa um *Buddy system* onde o conteúdo é separado em duas partes e cada uma delas verifica se a outra ainda se encontra na rede. No entanto, nenhum sistema completamente distribuído pode garantir que o conteúdo nunca será perdido.

3.2 Aplicações Emergentes

Como vimos ao longo deste artigo, o GUNet tem enormes potencialidades no que diz respeito ao seu uso, como framework, por parte das mais variadas aplicações, logo que tenham requisitos de rede.

¹Least Recently Used

Na realidade, construir uma aplicação que recorra à framework do GNUet não é difícil: pelo contrário a implementação seria trivial e, em muitos casos, mais simples do que a própria implementação de aplicações recorrendo aos protocolos típicos de transporte (TCP, UDP, etc.), visto que a framework do GNUet já providencia por si todos os aspectos relacionados com reencaminhamento, segurança, encriptação, anonimato, não-repúdio, etc..

O grande desafio é, então, construir aplicações tentando “esquecer” o paradigma actual das redes e da Internet; construir aplicações que usufruam ao máximo das capacidades da framework usada.

Como exemplo disso, e a par do sistema de partilha de ficheiros já desenvolvido, existem já outras prespectivas em vista (se bem que, convém reafirmar, qualquer aplicação que recorra à rede poderá usufruir desta framework), como um Browser², um Sistema de Conversação em tempo real³ e um sistema de correio electrónico⁴.

É também possível melhorar e simplificar o desenvolvimento de novas aplicações que recorram à framework do GNUet, através da construção de bibliotecas de interface com a framework do GNUet. O próprio desenvolvimento do GNUet prevê nisso quanto à criação de uma biblioteca em C, e já existem projectos para o desenvolvimento do mesmo em outras linguagens⁵.

²endereçoado muitas vezes, incluindo em

<http://lists.gnu.org/archive/html/gnunet-developers/2003-06/msg00042.html>

³endereçoado muitas vezes, incluindo em

<http://lists.gnu.org/archive/html/gnunet-developers/2003-06/msg00042.html> e em <http://lists.gnu.org/archive/html/gnunet-developers/2004-04/msg00025.html>

⁴endereçoado muitas vezes, incluindo em

<http://lists.gnu.org/archive/html/gnunet-developers/2004-06/msg00012.html>

⁵como por exemplo Python: <http://student.dei.uc.pt/~marado/software/pygnunet/>

Capítulo 4

Bibliografia

GNUnet

- GNUnet GNU Website
 - <http://www.gnu.org/software/gnunet/>
 - Página Oficial do projecto GNUnet na GNU
- The GNUnet Webpage
 - <http://www.ovmj.org/GNUnet>
 - Página Oficial do projecto GNUnet
- GNUnet FAQ
 - <http://www.ovmj.org/GNUnet/faq.php3>
 - Secção de perguntas frequentes sobre o GNUnet e respectivas respostas. Daqui foi já extraído o quadro comparativo entre vários sistemas de Partilha de Ficheiros.
- GNUnet Documentation
 - <http://www.ovmj.org/GNUnet/documentation.php3>
 - Resenha de documentação oficial referente ao GNUnet.
- GNUnet doxygen documentation
 - <http://www.ovmj.org/GNUnet/doxygen/html/>
 - Documentação gerada automaticamente relativa à última versão de todo o código associado à última versão lançada do GNUnet.
- An Encoding for Censorship-Resistant Sharing

- <http://www.ovmj.org/GNUnet/download/ecrs.ps>
- Este *paper* descreve a codificação do conteúdo para o sistema de partilha de ficheiros do GNUUnet. É descrito um conjunto de técnicas para codificar o conteúdo de forma a poder ser facilmente distribuído, pesquisável e obtido. O esquema de encriptação permite aos utilizadores inserir o mesmo conteúdo sob diferentes chaves; contudo chaves múltiplas levam a que hajam cópias praticamente idênticas no sistema, reduzindo os requisitos de armazenamento. As chaves podem ser escolhidas pela sua linguagem natural e podem ser combinadas com pedidos booleanos. Os pedidos e o conteúdo não podem ser descriptados pelos intermediários sem adivinharem a chave. No entanto, os intermediários conseguem verificar que uma certa resposta pertence a um certo pedido. A codificação do conteúdo produz variados blocos de pequenas dimensões, que podem ser facilmente distribuídos sobre vários nós. Isto permite à rede fazer *load balance*. Este *paper* descreve como otimizar *lookups* e estende o esquema com directorias e *namespaces*.
- A Transport Layer Abstraction for Peer-to-Peer Networks
 - <http://www.ovmj.org/GNUnet/download/transport.ps>
 - Este artigo descreve a abstração de transporte utilizada por GNUUnet. A implementação do transporte utilizando SMTP é descrita em detalhes, em particular as considerações de segurança que ocorrem quando este protocolo é utilizado. Este artigo também compara os desempenhos das camadas de transporte utilizando UDP, TCP, e SMTP.
- An Excess-Based Economic Model for Resource Allocation in Peer-to-Peer Networks
 - <http://www.ovmj.org/GNUnet/download/ebe.ps>
 - Este *paper* descreve o modelo económico atrás do GNUUnet. Descrevemos como a *accountability* é atingida numa rede anónima. Este modelo é baseado em confiança, e é usado apenas para a alocação de recursos. O modelo económico do GNUUnet é menos poderoso e muito mais simples que a maioria dos esquemas para dinheiro digital. Ao contrário destes esquemas, nós não existe como requisito uma autoridade certificadora.
- A Quick Introduction to Bloom Filters
 - <http://www.ovmj.org/GNUnet/download/bloomfilter.ps>
 - Este *paper* dá-nos uma breve descrição da estrutura de dados de um bloom filter. Os Bloom filters são usados no GNUUnet para fazer um set de testes rápido que precisam de ser executados por cada pedido.
- gap – practical anonymous networking

- <http://www.ovmj.org/GNUnet/download/aff.ps>
- <http://www.ovmj.org/GNUnet/download/pet/>
- Este artigo descreve como o anonimato é conseguido no GNUet. É descrito um novo esquema para transferência anónima de dados que atinge melhores garantias de anonimato do que os tradicionais esquemas de indirectão, além de ser mais eficiente. Enquanto os blocos de construção da nossa técnica usada para atingir o anonimato são idênticos aos de trabalhos anteriores, nós oferecemos uma nova perspectiva de como atingir o anonimato.
- help-gnunet mailing list
 - <http://lists.gnu.org/mailman/listinfo/help-gnunet>
 - Mailing list de ajuda sobre o GNUet
- gnunet-developers mailing list
 - <http://mail.gnu.org/mailman/listinfo/gnunet-developers>
 - Mailing list sobre o desenvolvimento do GNUet
- GNUet Bugtrack
 - <http://www.ovmj.org/~mantis/>
 - Sistema de repositório de *bugs* do GNUet
- GNUet Forum
 - <http://shell.franken.de/~dg1nsw/board/punbb/>
 - Forum oficial do GNUet

Segurança

- Counter Hack – A Step-by-Step Guide to Computer Attacks and Effective Defenses
 - Ed Skoudis, Prentice Hall Series in Computer Networking and Distributed Systems
 - Um excelente livro sobre segurança informática, quais os problemas existentes nos paradigmas actuais e como os colmatar.

Anonimato

- Towards measuring anonymity
 - <http://www.ovmj.org/GNUnet/papers/tmAnon.ps>

- Este artigo descreve o porquê de ser insuficiente apenas computar probabilidades para o set de anonimato. Os autores propõem o uso da entropia, o número de bits de informação adicional que o adversário precisa para quebrar o anonimato, visto que a métrica para como é que um dado sistema é anónimo.
- Peer to peer networks: Tarzan: a peer-to-peer anonymizing network layer
 - http://portal.acm.org/ft_gateway.cfm?id=586137&type=pdf&coll=portal&dl=ACM&CFID=29891908&CFTOKEN=84105450
 - Tarzan é uma *overlay* para uma rede *peer-to-peer* anónima. Porque proporciona serviço IP, o Tarzan é uma aplicação transparente, tolerante a falhas, altamente escalável e de fácil gestão.
- Towards an Information Theoretic Metric for Anonymity
 - <http://www.ovmj.org/GNUnet/papers/set.ps>
 - Este artigo faz um estudo relativo a problemas inerentes ao anonimato.
- Free Riding on Gnutella
 - <http://www.linuxdevcenter.com/pub/a/linux/2000/10/09/rt.html>
 - Estudo do tráfego do *gnutella*, afirmando que apenas metade dos ficheiros partilhados por esta rede provêm de 1% dos nós tornando-a mais como uma rede cliente-servidor do que *peer-to-peer*.

Encriptação

- Applied Cryptography
 - <http://www.schneier.com/book-applied.html>
 - Livro que abrange o tema da criptografia moderna, com incidência em como é que se pode usar a criptografia em comunicações electrónicas de forma a manter a privacidade dos dados.
- On the Utility of Distributed Cryptography in P2P and MANETs
 - <http://citeseer.ist.psu.edu/688081.html>
 - Os sistemas *peer-to-peer* gozam de muitos recursos e são escaláveis por natureza visto não dependerem de uma autoridade central, no entanto o facto dessa mesma autoridade não existir constitui novos desafios no campo da segurança. Este artigo reflecte sobre como resolver esses desafios.
- Practical Techniques for Searches on Encrypted Data

- http://www.ovmj.org/GNUnet/papers/searching_in_encrypted_data.pdf
- Os autores deste artigo fazem um estudo de como usar informação de proximidade de forma a otimizar o reencaminhamento em redes *peer-to-peer*.

Routing

- Exploiting network proximity in peer-to-peer overlay networks
 - <http://www.ovmj.org/GNUnet/papers/location.pdf>
 - Artigo que fala nas várias maneiras de usar a informação da proximidade entre nós para otimizar o encaminhamento em redes *peer-to-peer*.
- Onion Routing
 - http://www.fact-index.com/o/on/onion_routing.html
 - Onion Routing é uma técnica de comunicação anónima, baseada em *Mix networks* com algumas modificações, entre elas o conceito de “*routing onions*” responsáveis por conterem informação de encaminhamento em *layers* criptográficos.
- Distributed Data Location in a Dynamic Network
 - http://www.ovmj.org/GNUnet/papers/tapestry_CSD-02-1178.pdf
 - Duas propriedades importantes de uma infraestrutura de reencaminhamento: localização e adaptação rápida à entrada e saída de nós. Demonstra como estas duas propriedades podem ser alcançadas.
- Peer-to-peer: Graph-theoretic analysis of structured peer-to-peer systems: routing distances and fault resilience
 - <http://portal.acm.org/citation.cfm?id=863999>
 - Este artigo examina propriedades teóricas de grafos em redes *peer-to-peer* existentes e propõe uma nova infraestrutura baseada no diâmetro ótimo de grafos de *Bruijn*.

Peer-to-Peer

- Peer-to-Peer : Harnessing the Power of Disruptive Technologies
 - <http://www.amazon.com/exec/obidos/tg/detail/-/059600110X/002-6046920-9898411?v=glance>

- Livro usado como referência geral sobre todos os assuntos relacionados com redes *Peer-to-Peer*. Fala sobre as redes *Peer-to-Peer* em geral, vários projectos *peer-to-peer*, metadados, performance, confiança, não-repúdio, reputação, segurança e interoperabilidade usando *gateways*.
- Discovering P2P
 - http://www.amazon.com/exec/obidos/tg/detail/-/0782140181/ref=pd_rhf_f_1/002-6046920-9898411?v=glance&n=507846&no=*&st=*
 - Livro que fala sobre redes *Peer-to-Peer* e que refere aspectos como protecção da propriedade intelectual e privacidade.
- Peer-to-Peer: Building Secure, Scalable, and Manageable Networks
 - http://www.amazon.com/exec/obidos/tg/detail/-/0072192844/ref=pd_rhf_f_2/002-6046920-9898411?v=glance&n=507846
 - Livro que explora a aplicação prática das tecnologias *peer-to-peer* e como implementá-las. Possui definições, exemplos reais, possibilidades de aplicação prática e vários tipos de soluções para os problemas levantados nestes tipos de rede.
- PlanetP: Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities
 - <http://www.ovmj.org/GNUnet/papers/planetp.pdf>
 - O PlanetP é um sistema ponto-a-ponto onde a pesquisa de conteúdos é feito maioritariamente localmente. Cada nó sabe qual o conteúdo disponível em quais outros nós. A informação de índices é representada de forma compacta usando filtros de bloom e distribuídos pela rede usando mecanismos de push e pull.