RPM: Comunicação Anônima em Redes Par a Par

Robson Costa, Rafael R. Obelheiro, Joni da S. Fraga

[1]Departamento de Automação e Sistemas – Universidade Federal de Santa Catarina (UFSC)
Caixa Postal 476 – 88040-900 – Florianópolis – SC – Brasil

{robson,rro,fraga}@das.ufsc.br

Abstract. *Anonymity is a growing concern in recent Internet-based systems. Traditional mix- and multicast-based anonymity networks have a number of reliability, confidentiality, and performance issues. The large scale of P2P networks can be leveraged to minimize such issues, but these networks have to deal with node churn (a major factor in P2P systems) and the lower trustworthiness of individual nodes. In this paper we introduce RPM, a protocol for anonymous communication in P2P systems. In addition to anonymity, RPM aims at being resistant to churn and lowering the overhead usually found in anonymity systems. Our results show that RPM is effective in satisfying all these requirements, especially with respect to churn resistance.*

Resumo. *O anonimato é um preocupação crescente nos atuais sistemas baseados na Internet. A redes de anonimato tradicionais, baseadas em misturadores ou multicast, possuem limitações de confiabilidade, confidencialidade e desempenho. A ampla escala de redes P2P pode ser usada para minimizar tais limitações, mas essas redes têm de lidar com o churn (um fator importante em sistemas P2P) e a menor confiabilidade dos nós individuais. Este artigo apresenta RPM, um protocolo para comunicação anônima em sistemas P2P. Além do anonimato, o RPM tem por objetivo a resistência ao churn e a redução do custo computacional normalmente associado a sistemas de anonimato. Os resultados mostram que o RPM atinge eficazmente seus objetivos, especialmente com respeito à resistência ao churn.*

1. Introdução

Muitas aplicações na Internet necessitam manter a confidencialidade das suas comunicações. Muitas vezes, essas necessidades vão além do segredo do conteúdo das mensagens trocadas, abrangendo também o segredo a respeito de quem são as entidades que se comunicam. Podemos citar como exemplos os casos de empresas que estão formando uma aliança estratégica e que desejam manter sigilo disso perante seus concorrentes, ou de usuários finais que desejam guardar segredo sobre as pessoas com as quais trocam mensagens instantâneas e *emails*. Em situações como essas, mecanismos criptográficos fim a fim — que garantem confidencialidade de conteúdo — não são capazes, por si só, de fornecer a proteção desejada; para isso, são utilizados **sistemas de anonimato**.

Existem diversas arquiteturas para sistemas de anonimato. A mais simples utiliza um único nó como intermediário entre a origem e o destino [Anonymizer 2007], com comunicações criptografadas entre os nós comunicantes e o intermediário. O grau de segredo oferecido por essa arquitetura é limitado: se o nó intermediário for comprometido

ou apreendido, o segredo de todas as comunicações é violado. Além disso, se esse nó sofre uma falha, a comunicação anônima se torna inviável. Para contornar essas limitações, foram propostas outras arquiteturas baseadas em redes (lógicas) de anonimato. Nessas redes de anonimato, as mensagens percorrem uma seqüência de nós antes de chegar ao seu destino. A idéia é que cada nó que processa a mensagem tenha informações limitadas sobre a sua verdadeira origem ou destino, como forma de evitar que um único nó possa determinar os pares comunicantes na rede ou vincular um certo tráfego a nós específicos. Além disso, a dispersão do tráfego por vários nós dificulta a tarefa de observadores externos que tentam identificar padrões de comunicação na rede.

Existem duas variantes básicas de redes de anonimato, as redes de misturadores e as redes baseadas em *multicast*. Nas redes de misturadores (*mixes*) [Chaum 1981], o nó de origem monta uma seqüência de nós intermediários até o destino e usa camadas de cifragem (*onion encryption*) de tal forma que cada nó intermediário só conheça seu predecessor e seu sucessor na rota, e não a real origem ou destino do tráfego. Nas redes baseadas em *multicast* [Pfitzmann e Waidner 1987], uma mensagem é enviada para um grupo que contém o seu real destinatário; um observador externo é incapaz de inferir a qual dos membros do grupo a mensagem é efetivamente destinada. Embora sejam um avanço significativo em relação a um intermediário único, a maioria das redes de anonimato também sofre de limitações em termos de confiabilidade, confidencialidade e desempenho, uma vez que as funções que garantem o anonimato ficam geralmente centralizadas em um conjunto pequeno de nós. Quanto menor for a rede, mais sérias se tornam essas limitações.

Uma tentativa para resolver os problemas com redes de anonimato clássicas é o uso de redes par a par (P2P). A idéia é compartilhar os recursos dos usuários que desejam comunicações anônimas, aproveitando a ampla escala e a capacidade computacional disponível nas redes P2P para remover as restrições principalmente de escalabilidade das redes clássicas. As redes P2P se caracterizam pelo seu grande número de participantes, tipicamente máquinas voluntárias, o que faz com que haja grandes quantidades de nós que podem ser usados como intermediários entre origem e destino. Muito embora sistemas de anonimato P2P sejam eficazes nesse sentido, um desafio que eles enfrentam é a constante entrada e saída de nós da rede, um fenômeno conhecido como *churn*. Em redes de misturadores, o *churn* faz com que os caminhos tenham que ser reconstruídos com freqüência. Nas redes baseadas em *multicast*, ele agrava o problema de gerenciamento das chaves criptográficas usadas para comunicação com os grupos, que necessitam ser trocadas a cada mudança na composição do grupo. Em ambos os casos, são necessárias operações demoradas e de alto custo computacional, o que impacta negativamente no desempenho das redes. Outro desafio envolvendo redes de anonimato P2P é que, como o roteamento nessas redes é feito na camada de aplicação, torna-se mais fácil observar o tráfego passante, o que facilita a tarefa de espionar as comunicações alheias.

Este artigo propõe RPM (*Random Path+Multicast*), um protocolo para comunicação anônima em redes P2P. O RPM tem por objetivo garantir o anonimato de nós comunicantes perante observadores externos, minimizando o *overhead* e oferecendo resistência ao *churn*. Os resultados obtidos evidenciam que o RPM é capaz de fornecer comunicação anônima com nível elevado de confiabilidade mesmo com altos índices de *churn*.

O artigo está organizado da seguinte forma. A seção 2 apresenta os objetivos e

premissas utilizados, e a seção 3 descreve o protocolo em detalhes. Na seção 4 é feita uma avaliação do protocolo, tanto qualitativa quanto quantitativa, e na seção 5 são discutidos os trabalhos relacionados. A seção 6 apresenta conclusões e perspectivas futuras.

2. Objetivos e Premissas

2.1. Objetivos

O objetivo principal do esquema proposto é possibilitar a comunicação anônima entre nós em uma rede P2P. Mais especificamente, deseja-se garantir os seguintes atributos na terminologia de [Pfitzmann e Hansen 2007]:

- O anonimato de relacionamento entre emissor e receptor perante terceiros;
- O anonimato do emissor perante terceiros e perante o receptor;
- O anonimato do receptor perante terceiros.

Em linhas gerais, a meta é que mesmo nós que observam o tráfego não consigam identificar pares emissor-receptor que estão se comunicando (anonimato de relacionamento), nem quem é o emissor ou receptor de uma mensagem. Além disso, deseja-se ainda garantir que o receptor não possa, através do suporte de comunicação, saber quem é o emissor que está lhe contactando.

Além desses objetivos em termos de anonimato, outros objetivos específicos são a resistência ao *churn*, que permite que os nós se comuniquem mesmo que existam diversos outros nós entrando e saindo da rede P2P, e o baixo *overhead* da solução, caracterizado pelo uso restrito de mecanismos criptográficos, especialmente aqueles baseados em chaves públicas.

2.2. Premissas

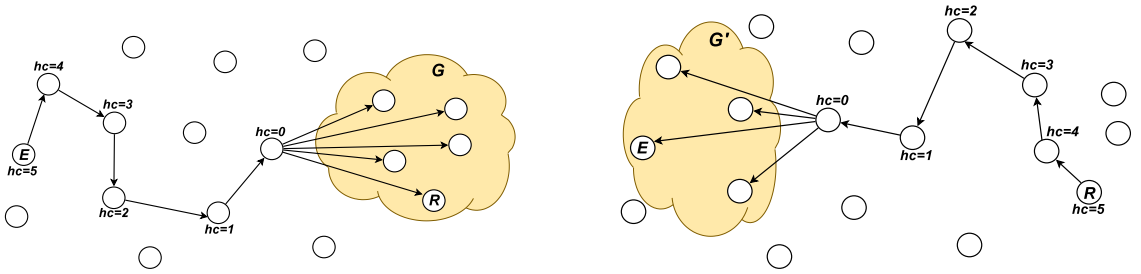
O esquema discutido na seção 3 pressupõe a existência de uma rede P2P que permite a descoberta dos endereços IP dos nós que a constituem, e que possui algum mecanismo para que um nó localize quem é o receptor com quem quer se comunicar. Além disso, considera-se que um nó pode descobrir, anonimamente, a chave pública desse receptor (por exemplo, recuperando um conjunto $\mathscr{C}$ de certificados de um repositório tal que o certificado do receptor $C_R \in \mathscr{C}$).

3. Descrição do Protocolo

3.1. Visão Geral

O RPM (*Random Path+Multicast*) usa roteamento aleatório para enviar mensagens do emissor até o receptor, com cada nó intermediário escolhendo o próximo salto ao acaso dentre seus vizinhos na rede P2P. Na origem, cada mensagem recebe um contador de saltos (*hop count*), que determina por quantos nós intermediários ela é encaminhada antes de ser transmitida até o destino; esse contador é escolhido aleatoriamente para cada mensagem, e decrementado a cada salto. Para garantir o anonimato do receptor, o emissor especifica um grupo de nós (que inclui o receptor) para os quais a mensagem será transmitida quando o contador de saltos chegar a zero (vide figura 1(a)).

O envio de respostas do receptor para o emissor segue o mesmo processo (vide figura 1(b)), sendo que o grupo usado para essas respostas é também definido e incluído previamente pelo emissor na mensagem original.



(a) Envio da mensagem do emissor E ao receptor R (G é o grupo de recepção)

(b) Resposta do receptor R ao emissor E (G' é o grupo de resposta)

Figura 1. Visão geral do protocolo

As mensagens de dados são cifradas usando uma chave simétrica estabelecida dinamicamente no início da comunicação entre origem e destino. Essa chave é alterada periodicamente para minimizar os riscos do seu comprometimento. Mensagens de controle são usadas no estabelecimento e mudança de chaves criptográficas simétricas. Além disso, as mensagens não têm uma identificação do seu emissor em texto claro, evitando assim comprometer o anonimato do emissor.

A seguir, são descritos em detalhes o encaminhamento aleatório de mensagens (seção 3.2), o envio de mensagens de resposta (seção 3.3) e a inicialização de fluxos de dados (seção 3.4).

3.2. Envio das mensagens

O algoritmo 1 mostra a função RANDOM-SEND, usada para transmitir mensagens para um determinado receptor; essa função não é invocada diretamente por uma aplicação, sendo usada como bloco de construção pelas funções que transmitem mensagens de dados ou de controle. Uma mensagem possui o formato $\langle type, hc, G, id, ciphertext \rangle$, onde:

- *type* é o tipo da mensagem, que pode ser de dados ou de controle;
- *hc* é o contador de saltos, que é escolhido aleatoriamente para cada mensagem, e que varia entre hc_{min} e hc_{max};
- G é o grupo de recepção;[1]
- *id* é um identificador único usado pelo receptor para localizar o contexto da mensagem, e que também é definido na inicialização;
- *ciphertext* é o conteúdo a ser transmitido, já cifrado com a chave apropriada (que depende do tipo da mensagem).

No algoritmo 1, é determinado apenas o valor de *hc*, com os demais campos sendo recebidos como parâmetros. A mensagem formada é enviada a um dos vizinhos do emissor, escolhido ao acaso.

O algoritmo 2 ilustra o uso de RANDOM-SEND para transmitir mensagens de dados. Primeiramente, recuperam-se o grupo de recepção G, o grupo de resposta G', o identificador *id* e a chave simétrica K_S associados ao receptor desejado (linhas 2 a 5).[2]

[1] O grupo de recepção G, o grupo de resposta G' e o identificador *id* são definidos na fase de inicialização, e discutidos em detalhes na seção 3.4.

[2] Como será discutido na seção 3.4, os grupos G e G' e a chave K_S são escolhidos pelo emissor, e o identificador *id* pelo receptor, durante a inicialização do fluxo de dados.

Algoritmo 1 Envio aleatório de mensagens

```
1: procedure RANDOM-SEND(type, G, id, ciphertext)
2:     hc ← RANDOM(hc_min, hc_max)        // sorteia um número hc ∈ [hc_min, hc_max]
3:     M ← ⟨type, hc, G, id, ciphertext⟩
4:     send M to random neighbor
5: end procedure
```

A seguir, os dados são cifrados, juntamente com o grupo de resposta, usando a chave K_S (linha 6), e transmitidos usando RANDOM-SEND (linha 7).

Algoritmo 2 Transmissão de mensagens de dados

```
1: procedure SEND-DATA(R, data)
2:     G ← GETRECEPTIONGROUP(R)
3:     G' ← GETRESPONSEGROUP(R)
4:     id ← GETRECEIVERID(R)
5:     K_S ← GETKEY(R)
6:     ciphertext ← ENCRYPT(K_S, {data, G'})
7:     RANDOM-SEND(DATA, G, id, ciphertext)
8: end procedure
```

O algoritmo 3 descreve como as mensagens transmitidas usando RANDOM-SEND são processadas. Ao receber uma mensagem M, o nó verifica o valor de hc. Se $hc = 1$, o nó assume o papel de **nó de saída**, transmitindo a mensagem para os nós que fazem parte do grupo de recepção G (linha 5). Caso $hc > 1$, o nó decrementa hc e envia a mensagem para um vizinho aleatório que não o seu predecessor (linha 7). Se $hc = 0$, o nó pertence ao grupo de recepção G, e pode ser o receptor da mensagem. Se o nó for um possível receptor e a mensagem for de dados, ele tenta localizar a chave simétrica correspondente ao *id* da mensagem (linha 9 a 11). Se conseguir, o nó decifra o *ciphertext* usando essa chave e processa o seu conteúdo (linhas 12 a 14); do contrário, o nó conclui que não é o receptor, e descarta a mensagem.

Cada nó mantém um *cache* das chaves simétricas definidas pelos emissores que se comunicam com ele (como receptor). O *id* é usado nas mensagens de dados para localizar a chave que deve ser usada para decifragem; caso o nó não tenha uma chave correspondente ao *id* fornecido, ele não é o real receptor da mensagem. Com isso, evita-se que cada nó pertencente ao grupo de recepção execute uma operação custosa (como uma decifragem usando sua chave privada) apenas para verificar se ele é realmente o receptor, o que melhora o desempenho da rede.

Como cada nó reescreve o endereço de origem do pacote, não é possível saber qual o caminho percorrido pela mensagem a não ser o nó predecessor e o nó posterior em que a mensagem transita. Também não é possível saber qual o número de saltos efetuados anteriormente pela mensagem, pois este valor é alterado a cada salto realizado e seu valor é escolhido aleatoriamente a cada pacote enviado. Para manter a privacidade na comunicação entre os nós vizinhos são usados canais TLS (*Transport Layer Security*) [Dierks e Rescorla 2006].

Algoritmo 3 Processamento de mensagens transmitidas aleatoriamente

```
1: upon receiving M = ⟨type, hc, G, id, ciphertext⟩ from x do
2:     if M.hc > 0 then
3:         M.hc ← M.hc − 1
4:         if M.hc = 0 then
5:             send M to M.G
6:         else
7:             send M to random neighbor ≠ x
8:     else
9:         if M.type = DATA then
10:            K_S ← GETKEYFROMID(M.id)
11:            if K_S ≠ ⊥ then
12:                payload ← DECRYPT(K_S, M.ciphertext)
13:                SETRESPONSEGROUP(M.id, payload.G′)
14:                deliver(payload.data)
15:        else
16:            // process control message
17: end do
```

3.3. Mensagens de Resposta

O envio de mensagens de resposta segue os mesmos princípios das mensagens originais, mas algumas restrições devem ser consideradas. A principal é que o receptor não conhece o emissor da mensagem original, apenas o grupo de resposta que deve usar para se comunicar com ele. O procedimento usado para enviar mensagens de resposta é mostrado no algoritmo 4.

Esse procedimento recebe como parâmetros a mensagem original M^o e o conteúdo de resposta a transmitir (*response*). A mensagem de resposta usa os grupos de recepção e resposta da mensagem original com papéis invertidos (linhas 2 e 3). A chave simétrica é recuperada a partir do mesmo identificador *id* da mensagem original (linha 4), sendo usada para cifrar o conteúdo juntamente com o grupo de resposta (linha 5). A mensagem resultante é então transmitida usando RANDOM-SEND (linha 6).

Algoritmo 4 Transmissão de mensagens de resposta

```
1: procedure SEND-RESPONSE(M^o, response)
2:     G ← M^o.G′
3:     G′ ← M^o.G
4:     K_S ← GETKEYFROMID(M^o.id)
5:     ciphertext ← ENCRYPT(K_S, {response, G′})
6:     RANDOM-SEND(DATA, G, M^o.id, ciphertext)
7: end procedure
```

3.4. Inicialização do Fluxo de Dados

Antes que seja possível transmitir dados do emissor para o receptor, é necessário realizar a inicialização do fluxo de dados. Nessa fase, o emissor define, além da chave criptográfica simétrica usada para cifragem dos dados, um grupo de recepção e um grupo de resposta. O primeiro é um conjunto de nós, pertencentes à rede P2P, que inclui o

receptor, e o segundo é um conjunto de nós P2P que inclui o próprio emissor. Ambos os grupos são formados escolhendo aleatoriamente um conjunto de nós da rede e agregando o receptor ou o emissor, conforme o caso. Um grupo é representado pelo conjunto de endereços IP dos seus integrantes.

Para descobrir os nós que podem fazer parte dos grupos, pode ser usado um mecanismo de filiação (*membership*) da própria rede P2P se este estiver disponível. Entretanto, tal mecanismo não é indispensável, pois cada nó pode ir formando sua própria idéia da filiação a partir dos seus vizinhos e dos nós que aparecem em grupos de recepção e resposta nas mensagens que ele recebe. Além disso, os vizinhos também podem trocar suas listas de membros periodicamente.

O algoritmo 5 mostra o comportamento do emissor na inicialização. Primeiramente, ele gera aleatoriamente os grupos de recepção e resposta, a chave simétrica K_S e um identificador único id_S (linhas 2 a 5). A seguir, o emissor envia uma mensagem de controle para o receptor contendo K_S, id_S e o grupo de resposta G' que será usado para o retorno de mensagens; esses dados são cifrados com a chave pública K_R^+ do receptor (linha 6). Essa mensagem é transmitida usando RANDOM-SEND (linha 7). O emissor aguarda uma confirmação do receptor antes de começar a transmitir dados, evitando assim o desperdício de recursos computacionais e de banda passante caso o receptor não possa ser alcançado (múltiplos envios ou retransmissões podem ser usados para tornar essa comunicação confiável).

Algoritmo 5 Inicialização do fluxo de dados: código do emissor

```
 1: procedure INIT-FLOW(R)
 2:     G ← GENERATERECEPTIONGROUP(R)
 3:     G' ← GENERATERESPONSEGROUP(R)
 4:     K_S ← GENERATEKEY()
 5:     id_S ← GENERATEUNIQUEID()
 6:     ciphertext ← ENCRYPT(K_R^+, {id_S, K_S, G'})
 7:     RANDOM-SEND(NEW-KEY, G, ⊥, ciphertext)
 8:     set id_S as pending
 9: end procedure
10: upon receiving A = ⟨KEY-ACK, hc, G', id_S, {id_R}_{K_S}⟩ from x do
11:     if A.id_S is pending then
12:         SETRECEIVERID(A.id_S, A.id_R)
13: end do
```

O procedimento do receptor é detalhado no algoritmo 6. Ao receber uma mensagem de controle $\langle$NEW-KEY, hc, G, $\{K_S, id_S, G'\}_{K_R^+}\rangle$, o receptor utiliza sua chave privada K_R^- (linha 2) para extrair a chave simétrica K_S e o grupo de resposta G' (caso a decifragem falhe, o nó é um membro de G que não o real receptor R). Na seqüência, o receptor gera seu identificador id_R (linha 4), associa a chave K_S a esse identificador (linha 5) e envia uma mensagem de confirmação com tipo KEY-ACK para o grupo G' (linhas 6 e 7). Quando o emissor recebe a confirmação, ele associa o identificador do receptor id_R ao fluxo de dados que estava pendente para o identificador id_S (algoritmo 5, linhas 10 a 13).

Os grupos, chaves e identificadores estabelecidos na inicialização do fluxo têm um período de validade. Após esse período, uma nova inicialização é disparada pelo emis-

Algoritmo 6 Inicialização do fluxo de dados: código do receptor

```
1: upon receiving M = ⟨NEW-KEY, hc, G, ciphertext⟩ from x do
2:     payload ← DECRYPT(K_R^-, M.ciphertext)        // M.ciphertext = {K_S, id_S, G'}_{K_R^+}
3:     if decryption is sucessful then
4:         id_R ← GENERATEUNIQUEID()
5:         ASSOCIATEKEY(id_R, payload.K_S)
6:         ciphertext ← ENCRYPT(payload.K_S, id_R)
7:         RANDOM-SEND(KEY-ACK, payload.G', payload.id_S, ciphertext)
8: end do
```

sor para estabelecer novos valores para esses parâmetros. Além disso, por uma questão de eficiência, os parâmetros não são descartados imediatamente após o término de uma transmissão; desta forma, novas comunicações entre o mesmo par emissor-receptor podem utilizar os parâmetros já estabelecidos, desde que eles ainda sejam válidos.

4. Avaliação

A avaliação deste trabalho foi dividida em duas partes, uma análise qualitativa da segurança oferecida (seção 4.1) e uma análise de resultados de simulação (seção 4.2).

4.1. Segurança

É possível, através de uma análise qualitativa, avaliar o quanto o modelo proposto satisfaz os seus objetivos, que são anonimato de relacionamento, emissor e receptor perante terceiros, e anonimato do emissor perante o receptor (vide seção 2.1). Como o anonimato de relacionamento existe sempre que houver anonimato de emissor e de receptor [Pfitzmann e Hansen 2007], são discutidos apenas estes dois últimos, que automaticamente garantem o primeiro.

4.1.1. Anonimato do Emissor

O anonimato do emissor perante o receptor é fornecido pelo uso de grupos de resposta, e está intrinsecamente ligado ao tamanho desses grupos. Quanto maior for o grupo de resposta, mais difícil será de identificar quem é o real emissor; o problema de usar grupos de resposta muito grandes é que isso prejudica o desempenho da rede, devido ao *multicast* realizado pelo nó de saída. Outro aspecto desse tipo de anonimato é que o protocolo de inicialização de fluxos de dados não distingue entre um emissor que vai iniciar uma transmissão de um emissor que quer apenas mudar os parâmetros do fluxo; a única forma do receptor diferenciar os dois casos é se ele tiver poucos fluxos estabelecidos e o novo grupo de resposta enviado na mensagem NEW-KEY tiver uma intersecção com o grupo de resposta de algum dos fluxos estabelecidos (ainda assim, essa inferência tem um grau de imprecisão).

O anonimato do emissor perante observadores que pertençam à rede P2P é dado pela variação do contador de saltos *hc*; como um nó nunca sabe ao certo o valor inicial de *hc*, ele não pode determinar com clareza quem é o emissor, já que não existe nenhuma outra informação que o identifique na mensagem. Porém, se um adversário controla todos os vizinhos de um nó x, ou uma fração significativa deles, ele pode descobrir que esse nó

é um emissor pela repetição de valores de *id* e *G* nas mensagens transmitidas por *x*. O anonimato de emissor perante nós externos à rede P2P é dado pelo uso de canais TLS entre os nós P2P.

4.1.2. Anonimato do Receptor

Assim como acontece no anonimato do emissor perante o receptor, o anonimato do receptor perante outros nós na rede P2P é dado pelo grupo de recepção, e está ligado ao tamanho desse grupo. Um adversário que controle um nó que receba uma mensagem pode agir como nó de saída malicioso, zerando *hc* e enviando a mensagem separadamente para cada nó do grupo de recepção, e observando se essa mensagem suscita alguma resposta. Existem duas dificuldades para implementar um ataque desse tipo, porém. A primeira é o próprio roteamento aleatório: esse ataque exige que o nó malicioso receba uma mensagem, e isso não é garantido. A outra dificuldade é que o adversário precisa ser capaz de observar todo o tráfego do nó contactado, e mesmo assim ele pode chegar a conclusões equivocadas (especialmente se o volume de tráfego na rede for elevado).

Um outro ataque que pode ser tentado, especialmente quando há pouco tráfego na rede, é a intersecção de grupos de recepção. Esse ataque é facilitado pelo uso de um *id* fixo em cada mensagem, que pode ser usado por observadores externos para identificar mensagens que pertencem ao mesmo fluxo. A solução adotada para esse problema foi manter o grupo de recepção constante para todas as mensagens com mesmo *id*, de modo que se torna impossível fazer essa intersecção dos grupos (caso contrário, o nó que aparecesse em todos os grupos com o mesmo *id* seria provavelmente o receptor). Além disso, o roteamento aleatório e a troca periódica do *id* também dificultam esse tipo de ataque. Uma outra solução seria transmitir sempre a chave simétrica cifrada com a chave pública do receptor; conforme explicado na seção 3.2, a desvantagem disso seria o desempenho.

4.2. Resultados Experimentais

4.2.1. Descrição dos Experimentos

Para avaliar quantitativamente o grau de anonimato e a resistência ao *churn* do mecanismo proposto, foram realizados alguns experimentos de simulação. Para isso foi utilizado o PeerSim [PeerSim 2007], um simulador de redes P2P baseado em Java. O PeerSim implementa um simulador baseado em ciclos, no qual o protocolo é executado em uma série de ciclos pré-definidos. Nos experimentos, cada simulação durou 30 ciclos.

A topologia das redes P2P segue o modelo ScaleFreeBA com grau de conectividade 3, ou seja, uma topologia *scale-free* [Albert e Barabási 2002] onde cada nó possui no mínimo três vizinhos. Como o simulador mantém sempre o mesmo resultado quando utilizada uma mesma semente geradora de números aleatórios, foram então criadas 100 sementes distintas a fim de se obter os resultados de 100 redes diferentes. Para o cálculo de valores médios foram descartadas as dez melhores e as dez piores amostras para evitar a influência de valores discrepantes (*outliers*).

Foram simulados três tamanhos diferentes de redes (2.000 nós, 6.000 nós e 10.000 nós) e três tamanhos diferentes de grupos de recepção (4, 6 e 8 nós). Além disso, o *churn* foi variado de 0% a 90%, a intervalos de 10%. Esta taxa de *churn* representa

a porcentagem de nós que são substituídos na topologia durante a simulação. Dos 30 ciclos da simulação, o *churn* ocorre nos 20 ciclos do meio (de 6 a 25), de modo a não interferir com a inicialização do fluxo de dados. Em cada ciclo, $1/20$ do total de nós da rede são escolhidos ao acaso e substituídos; por exemplo, em uma rede com 2.000 nós e 50% de *churn*, 50 nós são trocados em cada ciclo. A localização e a vizinhança dos nós substitutos são determinadas aleatoriamente (seguindo o modelo ScaleFreeBA), de modo que a topologia da rede muda a cada ciclo.

O padrão de tráfego simulado consiste no envio de um único arquivo através da rede P2P. O tamanho do arquivo utilizado foi de 1 Mb, fracionado para transmissão em pacotes de no máximo 512 bytes. Para cada pacote transmitido, o contador de saltos assumia um valor entre 3 e 8, inclusive. O emissor e o receptor não podiam ser removidos (devido ao *churn*) durante a simulação. Os resultados que foram obtidos com um único fluxo de dados podem ser facilmente extrapolados para vários fluxos simultâneos, pois nenhuma das medidas coletadas é influenciada pelos existência de outros fluxos.

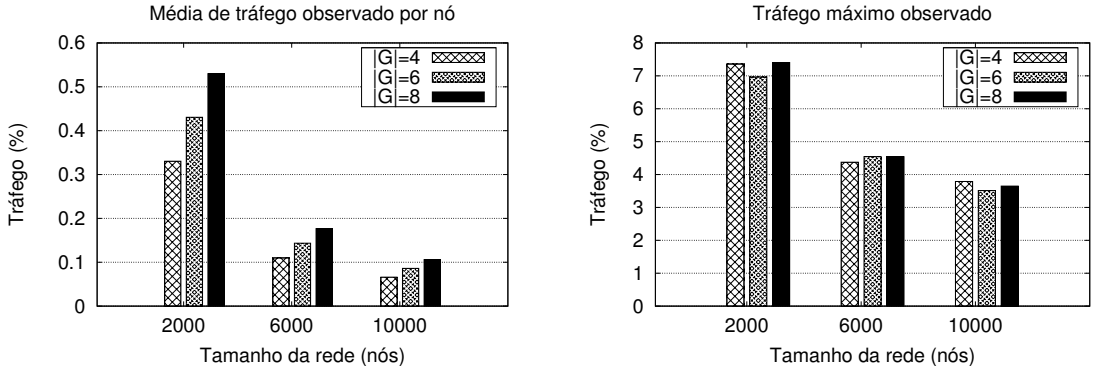
4.2.2. Resultados Obtidos

Primeiramente, mediu-se quanto do tráfego referente a uma transmissão foi observado em cada nó intermediário. O objetivo disso é avaliar a resistência da rede a análise de tráfego, uma vez que este tipo de análise depende intrinsecamente da quantidade de tráfego que pode ser observado. Os gráficos na figura 2(a) mostram a porcentagem média de tráfego observada em cada nó (obtida pela razão entre o tráfego recebido pelo nó e o total transmitido pelo emissor), considerando apenas nós que não sejam o emissor ou o receptor. Os resultados demonstram que nós isolados têm acesso a uma fração bastante pequena do tráfego, inferior a 0,6% em todas as situações analisadas e abaixo de 0,2% para redes com mais de 2.000 nós. Isso torna a análise de tráfego por nós isolados extremamente difícil, e comprova que o roteamento aleatório é bastante eficaz em dispersar o tráfego entre os diferentes nós da rede.

As tendências observadas na figura 2(a) situam-se dentro do esperado. A fração de tráfego observada é inversamente proporcional ao tamanho da rede, pois com o aumento da rede cresce o número de nós disponíveis para serem usados como intermediários, e cada nó passa a receber uma proporção menor de tráfego. Por outro lado, o tráfego observado é diretamente proporcional ao tamanho do grupo de recepção, pois quanto maior o grupo mais cópias das mensagens são enviadas (pelo nó de saída) e observadas.

O gráfico da figura 2(a) mostra o tráfego médio observado em cada nó. Para avaliar o quanto essa média é representativa, e verificar qual a situação de pior caso, analisou-se também a porcentagem máxima de tráfego observada por um nó qualquer em todas as situações simuladas. Os resultados obtidos são mostrados na figura 2(b) (a escala no eixo y é diferente daquela da figura 2(a)). O pior caso de todos, como seria de se esperar pelas tendências já discutidas, ocorreu em uma rede de 2.000 nós com grupo de recepção de tamanho 8; entretanto, mesmo neste caso, a porcentagem de tráfego observada foi de 7,4%, um índice que ainda torna a análise de tráfego uma tarefa difícil.

Em segundo lugar, foi mensurada a confiabilidade média da rede frente a diferentes níveis de *churn*, com o intuito avaliar a resistência da rede a este fenômeno. Os

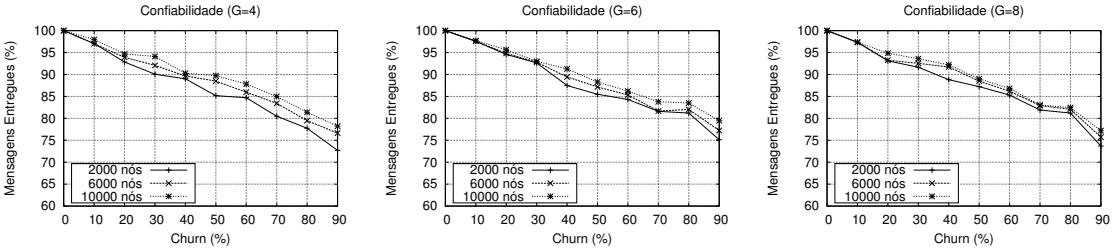


(a) Tráfego médio (b) Tráfego máximo

Figura 2. Tráfego observado por nó

resultados são mostrados na figura 3. A confiabilidade é dada pela razão entre o número de mensagens entregues no receptor e o número de mensagens enviadas pelo emissor. Como esperado, a confiabilidade é de 100% para redes sem *churn* e vai caindo à medida em que este aumenta. A queda, porém, é gradual, e mesmo com 90% de *churn* a confiabilidade permanece alta, com aproximadamente 75% das mensagens sendo entregues ao receptor em todos os cenários simulados. Esses resultados atestam que o esquema de roteamento aleatório utilizado é bastante resistente ao *churn*.

Analisando as curvas em cada um dos gráficos 3(a), 3(b) e 3(c), e comparando-se os gráficos entre si, percebe-se que não há uma correlação forte entre a confiabilidade e o tamanho da rede ou dos grupos de recepção. Parece existir uma pequena tendência de que redes maiores sejam mais confiáveis, mas uma análise dos dados coletados revela que essa relação não é estatisticamente significativa.



(a) $|G| = 4$ (b) $|G| = 6$ (c) $|G| = 8$

Figura 3. Confiabilidade das transmissões

Outros resultados foram também obtidos, mas serão apenas brevemente comentados por questões de espaço. Uma análise da latência na transmissão de mensagens, medida em número de saltos entre o emissor e o nó de saída, revelou uma oscilação em torno de 4,8 saltos em todos os cenários, ligeiramente inferior à média dos valores mínimo (3) e máximo (8) para *hc*, que é de 5,5. Outro fator analisado foi a influência do *churn* no tráfego médio observado em cada nó (os dados da figura 2 são para *churn* zero). Neste caso, verificou-se que o tráfego observado foi reduzido com o aumento do *churn*, o que se explica pela saída de nós da rede antes do final da transmissão.

5. Trabalhos Relacionados

Nas redes de misturadores (*mixnets*) propostas por [Chaum 1981], um emissor determina a seqüência de nós a ser percorrida por uma mensagem de tal forma que cada nó conhece apenas seu antecessor e seu sucessor na rota. O anonimato é garantido por criptografia em camadas (*onion encryption*): o emissor cifra a mensagem sucessivas vezes, e cada nó intermediário remove uma camada de cifragem para descobrir o próximo salto. Existem diversas implementações desse modelo básico [Freedman e Morris 2002, Rennhard e Plattner 2002, Dingledine et al. 2004], cada uma com pequenas variações em relação ao original. Comparadas ao esquema proposto, essas experiências apresentam como principais desvantagens a suscetibilidade ao *churn*, que pode fazer com que muitos caminhos tenham que ser reconstruídos (uma operação com alto custo), e (em alguns casos) o desempenho, já que operações criptográficas custosas são usadas extensivamente. A vantagem das redes de misturadores é que elas oferecem um anonimato de receptor mais forte, já que os nós intermediários não têm nenhuma noção de quem pode ser o receptor; por outro lado, quando o receptor pode ser externo à rede de anonimato [Dingledine et al. 2004], existe um nó de saída que conhece a identidade desse receptor.

Algumas propostas [Zhu e Hu 2004, Zhuang et al. 2005] tratam o problema do *churn* em redes de misturadores fazendo com que cada camada de cifragem seja endereçada a um grupo de nós, e não mais a um nó apenas. Essa abordagem introduz o problema de gerenciar a filiação (*membership*) dos grupos e as chaves usadas para cifrar mensagens para os grupos (públicas ou simétricas).

O Crowds [Reiter e Rubin 1998] é um sistema onde nós cooperam enviando mensagens em benefício de outros com o objetivo de esconder a identidade do emissor perante o receptor. É usada criptografia de chave simétrica, mas sem cifragem em camadas. O Hordes [Shields e Levine 2000] estende o Crowds introduzindo a idéia de grupos de resposta. Ambos utilizam um servidor centralizado para gerenciar a filiação do sistema e como centro de distribuição de chaves, o que introduz pontos únicos de falha. Além disso, o Crowds requer que, uma vez definidos, os caminhos mudem apenas a uma freqüência baixa (a cada 24 h), e o Hordes exige IP *multicast* para os grupos de resposta.

O P^5 [Sherwood et al. 2002] é um sistema baseado em *broadcast* hierárquico, em que os nós se juntam a grupos de diferentes tamanhos para ocultar o emissor e o receptor. Muito embora seja possível escolher o tamanho do grupo de forma a estabelecer um equilíbrio adequado entre anonimato e desempenho, o uso exclusivo de chaves públicas e de tráfego de fundo (tráfego inócuo usado apenas para manter uma taxa de transmissão constante e independente das reais necessidades de comunicação do nó) comprometem o desempenho do sistema como um todo.

Alguns protocolos de roteamento anônimo têm sido propostos para redes *ad hoc* sem fio [Kong e Hong 2003, Araújo 2005]. Os objetivos são tornar essas redes resistentes a análise de tráfego e manter o sigilo sobre a localização e a movimentação dos seus nós.

Os trabalhos mais próximos do esquema proposto são aqueles que não dependem da construção prévia de caminhos através da rede [Bansod et al. 2005, Han e Liu 2006]. No MuON [Bansod et al. 2005], o emissor gera um cabeçalho para cada mensagem a ser transmitida. Esse cabeçalho é difundido usando um protocolo epidêmico (*gossip*); o nó que conseguir decifrar (usando sua chave privada) um item do cabeçalho é o recep-

tor da mensagem, e solicita a mensagem completa ao seu dono, também identificado no cabeçalho. Nós que não são o receptor podem também, ao acaso, solicitar a mensagem completa, o que garante o anonimato do receptor. Ao receber uma mensagem completa, o nó passa a se anunciar como dono dessa mensagem nos cabeçalhos que difunde na rede, o que garante o anonimato do emissor. O resultado final é que os cabeçalhos são difundidos em toda a rede, e a mensagem completa só é enviada para alguns nós. A desvantagem do MuON em relação ao RPM é que essa difusão do cabeçalho faz com que todos os nós tenham que executar uma decifragem usando sua chave privada para cada pacote recebido, independente de serem o receptor ou não; no RPM, além das mensagens serem difundidas apenas para o grupo de recepção, o uso do *id* evita que outros nós processem mensagens inutilmente.

O Rumor Riding (RR) [Han e Liu 2006] é um protocolo para operação anônima em redes P2P não estruturadas. Na transmissão, cada mensagem é cifrada com uma chave simétrica; a mensagem cifrada e a chave são enviadas em pacotes separados para vizinhos diferentes, e percorrem caminhos aleatórios na rede. Quando esses pacotes se encontram em um nó qualquer, este decifra a mensagem usando a chave fornecida e transmite a mensagem decifrada (se for uma consulta, esta é difundida usando inundação; no caso de ser uma mensagem direcionada a um receptor específico, o nó abre uma conexão TCP com um *proxy* que responde pelo receptor). Para que a probabilidade de que os pares de pacotes (chave e mensagem cifrada) se encontrem seja significativa, cada nó da rede P2P mantém um *cache* contendo as mensagens recebidas recentemente; cada pacote de chave que chega é comparado com o conteúdo desse *cache* para ver se essa chave se aplica a alguma das mensagens armazenadas. Além disso, os resultados experimentais revelam que, na prática, o RR requer o uso de replicação de tráfego e de um TTL (*time-to-live*, que representa o número máximo de saltos dos pacotes) alto; uma boa probabilidade de encontro é obtida com $TTL > 30$ e de 2 a 6 réplicas de cada pacote. O RPM, por outro lado, dispensa o uso de replicação de tráfego, usa caminhos consideravelmente mais curtos, não possui *overhead* associado à manutenção de *caches* de pacotes e ao processamento das mensagens recebidas contra esses *caches*, e reduz (através do *id*) a necessidade de operações criptográficas em nós que não o receptor; todos esses fatores representam um ganho significativo de desempenho para o RPM. A principal desvantagem do RPM em relação ao RR é o menor anonimato do receptor.

6. Conclusão

Este artigo apresentou RPM, um esquema para comunicação anônima em redes par a par. O RPM utiliza roteamento aleatório para conferir resistência ao *churn* característico de redes P2P e um conjunto de mecanismos para reduzir o alto custo geralmente associado aos protocolos usados para garantir o anonimato nas comunicações. Os resultados demonstram que o RPM atinge plenamente os objetivos a que se propõe, particularmente no tocante ao *churn*.

Como extensão deste trabalho, propõe-se investigar um método para melhorar o anonimato do receptor, tornando os grupos opacos aos nós intermediários, e ampliar a experimentação realizada.

Referências

Albert, R. e Barabási, A.-L. (2002). Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74(1):47–97.

Anonymizer (2007). The Anonymizer. `http://www.anonymizer.com/`.

Araújo, A. M. (2005). ANDSR: Protocolo de roteamento anônimo para redes ad hoc. Dissertação de mestrado, Engenharia Elétrica, COPPE/UFRJ.

Bansod, N., Malgi, A., Choi, B. K., e Mayo, J. (2005). MuON: Epidemic based mutual anonymity. In *Proc. 13th International Conference on Network Protocols (ICNP)*, pp. 99–109.

Chaum, D. L. (1981). Untraceable electronic mail, return addresses and digital pseudonyms. *Communications of the ACM*, 24(2):84–88.

Dierks, T. e Rescorla, E. (2006). The transport layer security (TLS) protocol ver. 1.1. RFC 4346.

Dingledine, R., Mathewson, N., e Syverson, P. F. (2004). Tor: The second-generation onion router. In *Proc. 13th USENIX Security Symposium*, pp. 303–320, San Diego, CA, USA.

Freedman, M. J. e Morris, R. (2002). Tarzan: a peer-to-peer anonymizing network layer. In *Proc. 9th ACM Conference on Computer and Communications Security (CCS-9)*, pp. 193–206.

Han, J. e Liu, Y. (2006). Rumor riding: Anonymizing unstructured peer-to-peer systems. In *Proc. IEEE International Conference on Network Protocols (ICNP)*, pp. 22–31.

Kong, J. e Hong, X. (2003). ANODR: Anonymous on demand routing with untraceable routes for mobile ad-hoc networks. In *Proc. 4th ACM international symposium on Mobile ad hoc networking & computing (MobiHoc'03)*, pp. 291–302, New York, NY, USA.

PeerSim (2007). Peersim: A peer-to-peer simulator. `http://peersim.sf.net/`.

Pfitzmann, A. e Hansen, M. (2007). Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management – a consolidated proposal for terminology. Version 0.30. `http://dud.inf.tu-dresden.de/Anon_Terminology.shtml`.

Pfitzmann, A. e Waidner, M. (1987). Networks without user observability. *Computers & Security*, 6(2):158–166.

Reiter, M. K. e Rubin, A. D. (1998). Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security*, 1(1):66–92.

Rennhard, M. e Plattner, B. (2002). Introducing MorphMix: Peer-to-peer based anonymous Internet usage with collusion detection. In *Proc. 2002 ACM Workshop on Privacy in the Electronic Society*, pp. 91–102. New York, NY, USA.

Sherwood, R., Bhattacharjee, B., e Srinivasan, A. (2002). P5: A protocol for scalable anonymous communication. In *IEEE Symposium on Security and Privacy*, pp. 58–72.

Shields, C. e Levine, B. N. (2000). A protocol for anonymous communication over the Internet. In *Proc. 7th ACM Conference on Computer and Communications Security*, pp. 33–42. New York, NY, USA.

Zhu, Y. e Hu, Y. (2004). TAP: A novel tunneling approach for anonymity in structured P2P systems. In *Proc. International Conference on Parallel Processing (ICPP)*, pp. 21–28.

Zhuang, L., Zhou, F., Zhao, B. Y., e Rowstron, A. (2005). Cashmere: Resilient anonymous routing. In *Proc. 2nd Symposium on Networked Systems Design and Implementation (NSDI)*.