Uma Abordagem para Incorporação Flexível de Aspectos de Segurança em Aplicações *Peer-to-Peer*

André Detsch, Luciano Paschoal Gaspary, Marinho Pilla Barcellos, Ricardo Nabinger Sanchez

[1] Programa de Pós-Graduação em Computação Aplicada (PIPCA)
Universidade do Vale do Rio dos Sinos (UNISINOS)

Resumo. *A ampla adoção de aplicações Peer-to-Peer (P2P) em ambientes além do popular compartilhamento de arquivos demanda a satisfação de vários requisitos de segurança. Importantes avanços têm sido realizados em direção à incorporação de segurança em sistemas P2P, com a proposta de mecanismos para tratar de vulnerabilidades específicas. Entretanto, abordagens existentes carecem de flexibilidade, porque não possuem mecanismos suficientes para lidar com um amplo espectro de requisitos de forma integrada. Além disso, elas obrigam o usuário/aplicação a manipular uma interface de programação complexa, bem como passar por um processo de configuração desgastante. Este artigo lida com estas questões, apresentando a P2PSL (de P2P Security Layer). P2PSL permite a integração gradual e flexível de funcionalidades de segurança em aplicações P2P. Para mostrar a viabilidade conceitual e técnica da abordagem proposta, a P2PSL foi implementada, experimentalmente avaliada, e por fim incorporada como estudo de caso em uma infra-estrutura de computação em grade P2P.*

Abstract. *The widespread adoption of Peer-to-Peet (P2P) applications in environments beyond ordinary file sharing demands the fulfillment of several security requirements. Important steps have been taken towards security in P2P systems, with relevant mechanisms being proposed in the past to address specific vulnerabilities. However, existing approaches lack flexibility, since they do not (include enough mechanisms to) tackle a wide range of requirements in an integrated fashion. In addition, they oblige the user/application to manipulate a complex programming interface, as well as going through a cumbersome configuration process. To address these issues, we present P2PSL (P2P Security Layer), which allows gradual and flexible integration of security functionality into P2P applications. To show concept and technical feasibility, we have implemented P2PSL, assessed the overhead it induces, and incorporated the layer into a P2P-based grid computing infrastructure.*

1. Introdução

Aplicações *peer-to-peer* (P2P) não estão mais limitadas a usuários domésticos, e começam a ser aceitas em ambientes acadêmicos e corporativos. Embora compartilhamento de arquivos e aplicações de mensagens instantâneas sejam os exemplos mais tradicionais, eles já não são os únicos a se beneficiar de uma estrutura P2P. Por exemplo, aplicações de médio porte, cujos grupos são compostos de dezenas ou centenas de nodos (tal como compartilhamento de recursos [1] e trabalho cooperativo [2]), estão se tornando cada vez mais comuns. Este também é o caso na arena corporativa, onde sistemas P2P permitem que instituições troquem serviços [3].

Embora aplicações P2P possam contribuir para o compartilhamento de recursos e colaboração em larga escala, em ambientes geograficamente distribuídos com controle descentralizado e acoplamento fraco, a sua diversificação e disseminação são dificultadas pela

atual falta de segurança. Ainda é difícil desenvolver aplicações P2P que atendam a múltiplas combinações de aspectos de segurança, a saber, *confidencialidade*, *autenticidade*, *integridade*, *autorização*, *auditoria*, *não-repúdio*, *reputação* e *anonimato*. Há quatro razões para tal. Primeiro, os esquemas atuais para segurança de aplicações P2P cobrem apenas aspectos específicos (como por exemplo autenticação e reputação) e não podem ser facilmente integrados em um sistema único. Segundo, não conseguem isolar os aspectos de segurança da aplicação. Ao invés disso, obrigam o usuário ou desenvolvedor a lidar com uma interface de programação potencialmente complexa, e passar por um penoso processo de configuração.

Terceiro, as abordagens na literatura demandam um comportamento simétrico e uniforme de todos os *peers* que executam uma aplicação. Esta limitação é indesejável em algumas aplicações P2P, quando os requisitos de segurança podem variar significativamente entre usuários. Para ilustrar com um exemplo trivial, tome-se o Skype, uma aplicação de Voz-sobre-IP: um dado *peer* pode estabelecer diferentes tipos de comunicação com outros *peers*, cada um com seus requisitos de segurança próprios (por exemplo, usuários domésticos e corporativos podem ter diferentes necessidades).

A quarta e última razão é que os esquemas atuais oferecem pouco ou nenhum suporte para implantação gradual, porque eles precisam estar disponíveis em todos os *peers* de uma aplicação. É muito difícil, se não impossível, impor uma adoção abrupta de um novo esquema de segurança em um sistema de larga escala, de acoplamento fraco. Ao invés disso, é importante para o sucesso de sistemas P2P que os mesmos permitam a coexistência entre pares com e sem suporte de um esquema de segurança.

Este artigo apresenta a *Peer-to-Peer Security Layer*, ou P2PSL, que permite a inclusão de funcionalidades de segurança em aplicações P2P e trata dos problemas de falta de *integração*, *isolamento*, *assimetria* e *implantação gradual*, recém mencionados. A P2PSL isola a implementação de aspectos de segurança e sua configuração tanto da aplicação P2P como do *middleware* de comunicação subjacente. Cada *peer* pode especificar requisitos de segurança distintos (de acordo com diferentes graus de restrição) para cada canal de comunicação estabelecido com outros *peers*. Além disso, a implantação da P2PSL pelos *peers* que compõem a aplicação pode ser feita gradualmente. A implementação está baseada em JXTA [4], um conjunto consolidado de protocolos para o desenvolvimento de sistemas P2P, que tem sido amplamente usado pela comunidade.

O restante deste artigo está organizado da seguinte maneira. A Seção 2. discute trabalhos relacionados em segurança para sistemas P2P. A Seção 3. e a Seção 4. explicam a P2PSL e o protocolo de negociação para configuração dinâmica dos *peers*, respectivamente. A Seção 5. enfatiza os aspectos de implementação. A Seção 6. apresenta resultados de desempenho obtidos com a implementação, enquanto a Seção 7. demonstra o uso da P2PSL através de um estudo de caso em computação em grade. A Seção 8. encerra o artigo com considerações finais e perspectivas de trabalhos futuros.

2. Trabalhos Relacionados

O projeto de mecanismos para simplificar o desenvolvimento e a implantação de aplicações P2P seguras é de fundamental importância para ampliar o uso de tais aplicações. Vários esforços foram realizados nessa direção, tais como JXTA e PtPTL (*Peer-to-Peer Trusted Library*) [5]. JXTA oferece funcionalidades ligadas a criptografia, assinaturas e funções *hash* para o desenvolvimento de aplicações P2P seguras. Contudo, obriga o programador a incluir explicitamente no código fonte trechos para tratar os mecanismos de segurança desejados, le-

vando a uma complexidade adicional no processo de desenvolvimento. PtPTL, por sua vez, consiste em uma API baseada em OpenSSL que permite o estabelecimento de relações de confiança entre nodos individuais de uma rede *peer-to-peer*, bem como a criação de grupos seguros. Ambos JXTA e PtPTL são limitados em relação aos mecanismos de segurança oferecidos: autenticação, confidencialidade e integridade. Ao cogitar o emprego de outros mecanismos tais como não repúdio, autorização, auditoria e reputação, o desenvolvedor da aplicação não encontra nessas soluções nenhum suporte ou apoio.

A mesma limitação é observada em grande parte dos trabalhos em segurança de aplicações P2P, onde destacam-se Kim [6] e Park et al [7]. Em [6], é proposta uma abordagem para controlar *peers* que desejem passar a fazer parte de um grupo. A abordagem é constituída de dois componentes: *Group Charter* e *Group Authority*. O primeiro é um documento digital que informa as regras para um *peer* ser aceito em um grupo, enquanto o segundo é um sistema responsável por garantir o cumprimento dessas regras. Um *Group Authority* se comporta como uma autoridade certificadora, emitindo certificados que determinam se um *peer* está ou não autorizado a fazer parte de um determinado grupo. Já em [7], Park et al definem uma arquitetura de controle de acesso baseado em RBAC (*Role-Based Access Control*). Os autores definem um *middleware* genérico que opera ao estilo de um serviço de diretórios, indexando os recursos disponíveis na rede *peer-to-peer*. Um *peer* interessado em prover algum serviço ou recurso à rede P2P deve registrá-lo junto ao referido *middleware*. Sempre que um *peer* deseja acessar um dado serviço ou recurso, ele envia uma requisição ao *middleware* com a intenção de descobrir que *peer* fornece o mesmo e se tem permissão para acessá-lo. Em ambos os artigos, dois requisitos de segurança recebem atenção especial: autenticação e autorização. No entanto, a exemplo do que ocorre com as abordagens enumeradas no parágrafo anterior, tratam-se de implementações isoladas que não podem ser facilmente estendidas ou integradas com as de outros mecanismos de segurança.

No que se refere à configuração de mecanismos de segurança em aplicações P2P, é particularmente importante adotar soluções mais flexíveis, descentralizadas e próximas do usuário final (usuários das estações finais são os próprios administradores) em virtude da preocupação com a escalabilidade das mesmas. Contudo, esta não é a realidade dos trabalhos recém mencionados [6, 7], que exigem que as políticas sejam armazenadas e mantidas em servidores centralizados. Abordagens propostas pelas comunidades de *web services* [8] e computação em grade [9] também pecam nesse sentido. Tais propostas ainda dependem de um grande esforço de configuração, que é frequentemente realizado de maneira centralizada.

3. P2P *Security Layer* (P2PSL)

P2PSL consiste em uma camada de segurança que é implementada e configurada de maneira independente da aplicação P2P e do *middleware* de comunicação subjacente. Os requisitos de segurança são satisfeitos por módulos que implementam diferentes técnicas de segurança. Tais módulos são combinados como peças de um quebra-cabeça para fornecer segurança de maneira flexível e assimétrica. Em consonância com a natureza intrinsecamente descentralizada de aplicações P2P, a definição e a configuração dos módulos a serem empregados são realizadas de maneira autônoma em cada *peer*, pelo usuário local (auxiliado por uma ferramenta gráfica, como será explicado na Seção 5). A configuração da camada de segurança é feita com base em *perfis*, cada um servindo a diferentes necessidades de segurança. Nesse contexto, *peers* remotos podem ser associados a um dos perfis definidos; *peers* ainda desconhecidos são automaticamente associados a um perfil *default*.

A Figura 1 ilustra um exemplo de uma rede de *peers* usando P2PSL. As configurações para Peer2 e Peer3 são apresentadas em detalhe. Observe o Perfil B definido pelo Peer2. O mesmo define que autenticação deve ser utilizada em todas as mensagens enviadas, enquanto autenticação e confidencialidade são exigidas de todas as mensagens recebidas. O Peer2, então, associa o Peer3 a esse perfil.

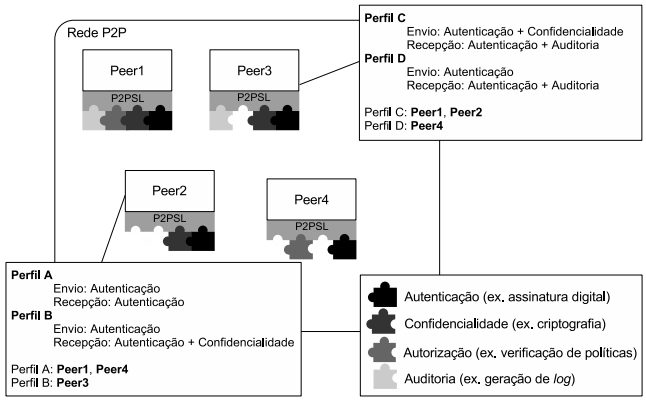


Figura 1. Rede de *peers* usando P2PSL.

Durante a operação do sistema, a instância de P2PSL associada com o *peer* se comporta como um *wrapper* entre a aplicação P2P e o *middleware* de comunicação subjacente. Sempre que mensagens são recebidas ou enviadas, os módulos envolvidos são acionados para satisfazer os requisitos de segurança estabelecidos. A Figura 2 ilustra esse processo, onde o Peer2 utiliza o Perfil B para enviar e receber mensagens do Peer3. No exemplo, mensagens de saída são repassadas para um módulo de assinatura digital. Por sua vez, mensagens que chegam ao Peer2 oriundas do Peer3 passam pelos módulos de assinatura digital e criptografia. Se a mensagem perpassa esses módulos de maneira bem sucedida, ela é entregue à aplicação. Caso contrário, dependendo das características do módulo, a mensagem é silenciosamente descartada.

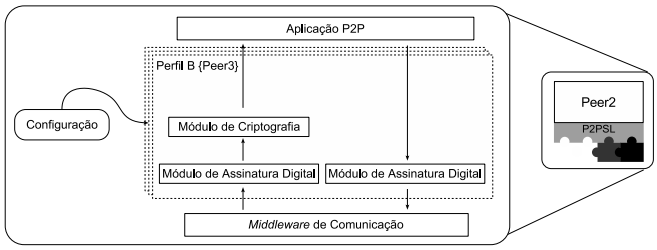


Figura 2. Instância da P2PSL em um *peer*.

Dependendo do requisito de segurança sendo assegurado, um módulo pode provocar mudanças na mensagem. Este é o caso do exemplo recém ilustrado, em que uma assinatura é adicionada às mensagens enviadas do Peer2 para o Peer3.

3.1. Módulos de Segurança

Os módulos de segurança são as peças-chave de P2PSL. Cada técnica de segurança implementada é representada por um desses módulos. Por exemplo, o suporte à auditoria de mensagens é tipicamente fornecido por um módulo de geração de *logs*; autenticação e integridade de mensagens, através de um módulo de assinatura digital; e controle de acesso a recursos (autorização), através de um módulo de verificação de políticas de acesso.

Os módulos implementam uma interface genérica, com métodos para a verificação de mensagens recebidas e a preparação de mensagens a serem enviadas. Graças a essa interface, a adição de novos módulos à camada pode ser realizada de forma bastante simplificada. Quando invocado pela camada de segurança, o módulo realiza o processamento necessário (possivelmente alterando o conteúdo da mensagem) e informa se a operação foi bem sucedida ou não.

3.2. Repositório de Caracterização

Conceitualmente, os módulos de segurança diferem entre si em relação aos parâmetros de entrada e à dinâmica de sua utilização. Um repositório independente é utilizado para lidar com essa heterogeneidade e evitar que se tenha que integrar diretamente à P2PSL as características particulares de cada módulo idealizado. O repositório é implementado através de um arquivo XML que contém a caracterização dos módulos disponíveis. Ele define os parâmetros e a forma de uso de cada módulo, bem como a combinação de módulos que satisfaz cada um dos requisitos de segurança.

Os parâmetros de cada módulo devem ser configurados para que o mesmo se comporte adequadamente. Exemplos de parâmetros são a identificação de uma chave a ser usada em uma cifragem ou assinatura digital, o nível de detalhe a ser empregado pelo módulo de geração de *logs*, e o caminho para um arquivo que armazene as políticas de acesso a serem usadas por um módulo de autorização. Alguns dos parâmetros especificados localmente podem ser importantes para outros *peers* participantes da rede. Por exemplo, se um *peer* deseja cifrar uma mensagem usando criptografia assimétrica, ele precisa conhecer antecipadamente a chave pública do *peer* destinatário. Essas informações são trocadas através de um protocolo de negociação (explicado na Seção 4.).

P2PSL associa a cada módulo quatro características básicas (especificadas através de atributos). O primeiro atributo (`export_requirement`) define se o *peer* remetente precisa realizar modificações na mensagem sendo enviada para permitir que o mesmo módulo possa ser usado no *peer* destinatário. Por exemplo, em módulos que envolvem autenticação e criptografia, as mensagens sendo enviadas precisam ser necessariamente modificadas para que o destinatário possa executar o mesmo módulo quando a mensagem for recebida. Por outro lado, na geração de *logs* e verificação de políticas de acesso, o conteúdo original da mensagem é suficiente para que os respectivos módulos possam ser aplicados pelo *peer* destinatário no recebimento da mensagem. O segundo atributo (`obligatory_if_applied`) informa se é obrigatório que o módulo seja usado no recebimento de uma mensagem para recuperar os dados originais. Este é o caso do módulo de criptografia, mas não do de autenticação, uma vez que esse último apenas adiciona uma assinatura à mensagem. O terceiro e quarto atributos (`allow_on_bcast_sending` e `discard_on_failure`) indicam, respectivamente, a possibilidade de utilizar o módulo em transmissões *broadcast* e a necessidade de descartar uma mensagem quando a sua verificação pelo módulo falhar.

Conforme mencionado anteriormente, o repositório armazena, além de parâmetros e atributos de cada módulo, o mapeamento entre requisitos e módulos de segurança. Note-se que esse mapeamento não é 1:1, uma vez que um módulo pode atender a mais de um requisito de segurança, e a satisfação de um requisito pode exigir o uso de diversos módulos. Por exemplo, a combinação de um módulo para verificação de integridade via sumários SHA-1 com outro para assinatura desses sumários representa uma opção para autenticação de mensagens.

3.3. Repositório de Configuração

O repositório de configuração contém todas as informações exigidas pela camada de segurança para aplicar corretamente os módulos especificados pelo usuário. Ele é implementado através de um arquivo XML e sua função é armazenar as configurações relativas a cada perfil. Além da definição dos perfis, o arquivo XML armazena a lista de *peers* remotos e seus requisitos, bem como configurações padrões para os módulos disponíveis localmente.

Um atributo importante configurado para cada perfil é o `respect_remote_requirements`. Quando esse atributo é habilitado, requisitos de *peers* remotos (i.e. módulos de segurança) são automaticamente satisfeitos (usando os respectivos módulos) quando mensagens são trocadas com *peers* pertencentes ao perfil correspondente. Note que o conjunto de módulos aplicados, nesse caso, será a união dos módulos definidos localmente pelo perfil e aqueles demandados (por um perfil) pelo *peer* remoto.

4. Negociação de Requisitos de um *Peer*

Redes P2P são tipicamente dinâmicas, heterogêneas e assimétricas em termos de segurança. Em função dessas propriedades, é impraticável configurar manualmente a camada de segurança de um *peer* com relação a cada outro *peer*. P2PSL trata desse problema de duas maneiras. Primeiro, ela permite que os usuários classifiquem *peers* de acordo com perfis (conforme apresentado na seção anterior). Segundo, a camada de segurança inclui mecanismos que guiam e automatizam o processo de configuração, permitindo que um grande número de relacionamentos entre *peers* seja gerenciado com pouco esforço.

Nesta seção, são apresentados os três diferentes *momentos de configuração* através dos quais um *peer* passa durante sua vida: (*i*) configuração inicial que precede a ativação de um *peer*, (*ii*) negociação quando um *peer* ingressa na rede, e (*iii*) ajustes de configuração que ocorrem em resposta às alterações na rede P2P. Cada uma é descrita a seguir.

O primeiro momento se refere à configuração inicial. Antes de um *peer* com P2PSL ser iniciado, deve-se criar um par de chaves e publicar a chave pública. Isto é necessário para obter-se autenticidade e integridade nas mensagens de *controle* trocadas pelos *peers*. A criação de um par de chaves PGP pode ser feita usando-se uma ferramente externa à P2PSL, como GnuPG ou Kgpg. Para a publicação, existem duas alternativas: uma é usar o PGP de forma descentralizada, enquanto que a outra é empregar uma AC (Autoridade Certificadora), que corresponde a um servidor centralizado. Sempre que um *peer* receber uma mensagem assinada com uma chave desconhecida, ele solicita à AC a chave pública correspondente ao *peer* remoto e armazena-a localmente para uso futuro.

Quando um *peer* ingressa em uma rede P2P, ele precisa descobrir os outros *peers* ativos. No que tange à P2PSL, um *peer* precisa determinar o conjunto de mecanismos de segurança exigido por cada um dos outros *peers* com os quais ele deseja se comunicar. Isto corresponde ao segundo momento de configuração. Para tal, é empregado um protocolo de negociação entre *peers*, definido com base em primitivas *send*, *receive* e *broadcast*. Assume-se que as mesmas sejam implementadas pelo *middleware* subjacente, que no caso atual, é JXTA.

O protocolo é descrito a seguir através de um exemplo (vide Figura 3). Assume-se, por enquanto, que não ocorrem falhas na rede P2P; posteriormente são discutidas implicações decorrentes de falhas. Seja Peer1 o *peer* que ingressa na rede, e Peer2 e Peer3 *peers* que já estão ativos; assuma-se ainda que Peer2 saiba a respeito do Peer1 e tenha uma cópia de sua chave pública, enquanto Peer3 e Peer1 não se conhecem previamente. O protocolo funciona

em três passos:

1. Peer1, ingressando na rede, envia por *broadcast* uma mensagem REQUIREMENTS_REQUEST assinada.
2. Peer2 recebe a mensagem, verifica a assinatura, e imediatamente responde ao Peer1 com uma mensagem REQUIREMENTS_REQUEST_AND_REPLY especificando seus requisitos para com Peer1 e ao mesmo tempo solicitando ao Peer1 seus requisitos para com Peer2. Peer3 também recebe a mensagem REQUIREMENTS_REQUEST, mas é incapaz de verificá-la. Peer3 busca junto à AC a chave pública do Peer1, verifica a mensagem recebida do mesmo, e assumindo que ela esteja correta, envia uma mensagem REQUIREMENTS_REQUEST_AND_REPLY para o Peer1 (de forma semelhante ao Peer2).
3. Peer1 verifica a assinatura da mensagem REQUIREMENTS_REQUEST_AND_REPLY recebida do Peer2. Assumindo que ela esteja correta, o Peer1 envia uma mensagem REQUIREMENTS_REPLY individual ao Peer2 contendo seus requisitos para com Peer2. Peer1 realiza um processo similar para o Peer3, mas antes busca a chave pública do Peer3 junto à AC.

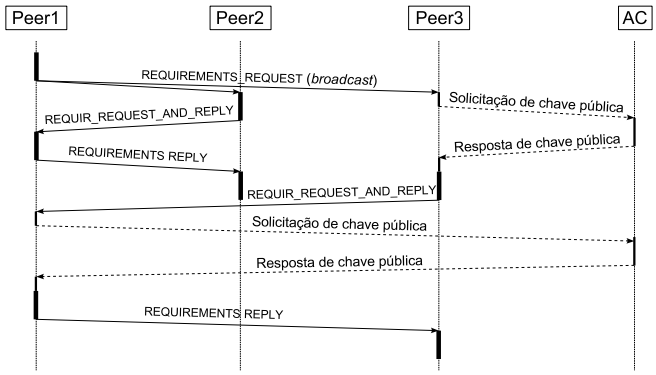


Figura 3. Diagrama de tempo representando a descoberta de requisitos de outros *peers*.

Como mencionado anteriormente, a autenticidade de mensagens é assegurada através de assinaturas digitais. Se um *peer*, a qualquer momento, recebe uma mensagem assinada com uma chave desconhecida, ele envia uma requisição à AC para obter a chave pública correspondente. No exemplo, Peer3 não conhecia Peer1 e portanto precisou buscar a chave pública do Peer1 na AC. De forma similar, Peer1 não sabia sobre Peer3, e consultou a AC também. Se uma mensagem assinada recebida não tem sua assinatura consistente com a assinatura digital esperada, o *peer* a ignorará. Além disso, como resposta, ele coloca *peers* que (consistentemente) enviam mensagens inválidas em um perfil associado a *peers* banidos e, portanto, evita processar quaisquer outras comunicações com aquele *peer*.

Peers previamente desconhecidos são considerados primeiramente não-confiáveis (*untrusted*) e automaticamente colocados em um dos perfis pré-definidos, como descrito a seguir. O perfil legado (*legacy*) se refere aos *peers* que não tem a camada de segurança implementada ou apropriadamente configurada (por exemplo, com uma assinatura digital inválida). O perfil *default* se refere a *peers* que implementam a camada de segurança e possuem uma assinatura digital válida, mas que ainda não haviam sido descobertos por este *peer* (como quando um *peer* encontra um identificador de *peer* pela primeira vez). Relembrando, na Figura 3 foi assumido que Peer1 não sabia sobre o Peer3. Neste caso, Peer3 é associado ao perfil *default*.

Portanto, ao fim do segundo momento de configuração, um *peer* terá estabelecido o conjunto de requisitos (mapeados no conjunto de módulos) a serem aplicados localmente ao enviar e receber de cada outro *peer* com o qual ele deseja se comunicar, e terá informado a estes *peers* quais são seus próprios requisitos.

O terceiro e último momento se refere às mudanças na configuração de segurança de um *peer*, que podem acontecer a qualquer momento de sua vida. Isto é mais provável em aplicações de vida longa, quando há tempo suficiente para novos *peers* entrarem na rede ou *peers* existentes alterarem seus requisitos. Caso assim seja, em reciprocidade, um *peer* pode desejar atualizar seus próprios requisitos em relação àquele *peer*. Isto seria tipicamente obtido migrando-se um *peer* de um perfil para outro. Também é possível mudar os requisitos associados a um perfil, aquele ao qual o *peer* pertence, mas naturalmente afetando os demais *peers* naquele perfil.

A introdução de novos requisitos afeta outros *peers* e precisa ser comunicada aos mesmos. Quando um *peer*, como por exemplo Peer1, altera o conjunto de requisitos em relação a outro *peer*, como Peer2, Peer1 envia uma mensagem a Peer2 informando os novos requisitos. Se o conjunto de requisitos de Peer2 é aumentado, as mensagens enviadas por Peer1 a Peer2 seriam afetadas imediatamente. Para permitir um aumento gradual e suave de requisitos, um intervalo de transição pode ser especificado, atrasando temporariamente as medidas de restrição pela aplicação.

Na descrição acima, foi assumido que não haveria falhas na rede ou nos *peers*. Agora são considerados alguns dos tipos mais comuns de falhas que podem acontecer em um sistema P2P, e como a P2PSL é afetada pelos mesmos. Primeiro, um *peer* pode não receber uma resposta à mensagem REQUIREMENTS_REQUEST de um *peer* ativo devido a uma falha relacionada à rede (como partição). Para lidar com este problema, um *peer* precisa empregar temporizadores para evitar uma espera indefinida. Um *peer* que falha em responder não terá seus requisitos registrados e, portanto, permanecerá nos perfis *default* ou legado (considerados não seguros para comunicação). Assim, o temporizador de espera precisa ser suficientemente longo, considerando que no caso de contenção em um *peer* ou na rede, a chegada de uma mensagem pode ser arbitrariamente atrasada. Adicionalmente, um *peer* pode sofrer uma falha de colapso, no caso em que outros *peers* podem ter que limitar a espera por mensagens, e detectar a falha através de uma entidade externa à aplicação (um *dectetor de defeitos*). Por fim, a P2PSL não tem suporte a falhas Bizantinas, que ocorrem quando um *peer* (potencialmente confiável) começa a se comportar de forma arbitrária, maliciosamente ou não.

5. Implementação

A P2PSL foi implementada em Java, utilizando JXTA ([4, 10]) como *middleware* de comunicação subjacente. JXTA é um projeto que busca estabelecer um conjunto de protocolos independentes de implementação que permitam a criação de uma estrutura P2P de propósito geral, empregável por diferentes aplicações. Mais especificamente, a implementação foi baseada na JXTA Abstraction Layer (JAL - [11]), uma biblioteca cujo principal objetivo é facilitar o desenvolvimento de aplicações sobre JXTA. JAL abstrai vários aspectos da arquitetura JXTA, oferecendo ao programador uma interface mais simples para acessar funcionalidades comuns em sistemas P2P, como envio de mensagens (*unicast* ou *broadcast*), criação de grupos ou busca de recursos. JXTA é amplamente utilizada em uma grande variedade de trabalhos de pesquisa, fazendo com que nossa implementação possa ser empregada em vários projetos já existentes.

A Figura 4 apresenta a estrutura geral de implementação da P2PSL. A principal

peça de implementação é a classe `SecurePeer`, que atua como o invólucro que intercepta mensagens sendo enviadas ou recebidas pelo *peer*. Os módulos são especializações da classe `SecurityModule`, e oferecem uma interface externa genérica que, por sua vez, é acessada pela classe `SecurePeer`. Os métodos em `SecurityModule` representam a análise de cada mensagem recebida e em processo de envio (`verifyIncomingMessage` e `adjustOutgoingMessage`, respectivamente) para verificar se estas satisfazem os requisitos especificados. O acesso aos módulos se dá exclusivamente através desta interface genérica. Em conjunto com a carga dinâmica de classes na instanciação de módulos, isto torna a camada de segurança *extensível*: novos módulos podem ser adicionados sem a necesidade de recompilação da mesma.

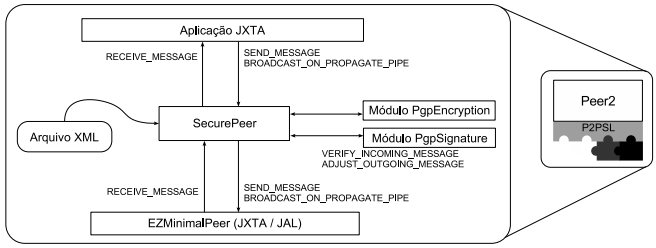


Figura 4. Implementação da P2PSL.

5.1. Módulos Disponíveis

Quatro módulos foram implementados para prover as seguintes as seguintes funcionalidades de segurança: autenticação, confidencialidade, integridade, autorização e auditoria. Novos módulos podem ser adicionados sem alterações ou recompilação de código, como no caso da incorporação um novo mecanismo de segurança ou escolha de uma técnica de segurança alternativa mais adequada para algum determinado cenário. Em linha com a filosofia da P2PSL, os módulos foram implementados de forma que *peers* que não possuem a camada de segurança também possam participar do sistema. Isto permite uma adoção gradual da camada de segurança em sistemas P2P operacionais, e possibilita ao usuário de cada *peer* a escolha de uso ou não da camada. Entretanto, *peers* que implementam a P2PSL podem bloquear mensagens enviadas por *peers* que não a implementam exigindo, por exemplo, autenticação de mensagens desses *peers*. A seguir, são descritos os módulos que foram implementados.

Assinatura PGP O módulo de assinatura PGP visa assegurar autenticidade e integridade da troca de mensagens entre *peers*. Neste modelo, chaves públicas são armazenadas em servidores arbitrários, ou mesmo repassadas diretamente entre as partes interessadas. No momento do envio, é adicionada uma assinatura à cada mensagem. No recebimento, esta assinatura é verificada, resultando em descarte da mensagem em caso de divergência.

Criptografia PGP Como no módulo anterior, este módulo utiliza as funcionalidades providas pelo PGP para garantir confidencialidade de mensagens. Durante o envio, o módulo de criptografia PGP substiui os campos da mensagem por um vetor de bytes cifrado. No recebimento, este vetor de bytes é decifrado, restaurando os campos originais da mensagem.

Verificação de políticas de acesso Visando prover controle de acesso aos recursos (autorização), o módulo para verificação de políticas usa informações genéricas da mensagem (como data, tamanho e identificação de remetente) para definir se a mensagem pode ou

não ser entregue. A especificação e a verificação de políticas são baseadas em XACML [12], um padrão criado pelo OASIS para a definição de políticas de controle de acesso através de XML. As políticas são definidas classificando os peers em papéis, seguindo o mecanismo de *Role-Based Access Control* (RBAC). A referência [13] oferece detalhes adicionais sobre este módulo.

Geração de *Logs* Quando usado, o módulo de geração de *logs* cria, de acordo com um nível pré-estabelecido, um registro textual que apresenta informações sobre as mensagens trocadas. Através deste módulo, é possível auditar a troca de mensagens, identificando problemas no funcionamento ou emprego da aplicação.

5.2. Assistente de Configuração

Para facilitar a tarefa de ajustar o repositório de configuração, descrito na Seção 3.3., uma interface gráfica (*Graphical User Interface* - GUI) é disponibilizada. Esta GUI é ativada durante o processo de configuração do *peer*, descrito na Seção 4.. Para cada perfil, o usuário especifica os aspectos a serem atendidos e então determina a combinação de técnicas disponíveis para alcançar o objetivo desejado.

A Figura 5 ilustra a ferramenta apresentando as opções de configuração de perfil de segurança, onde parâmetros podem ser especificados para um módulo (no caso ilustrado, o módulo de assinatura PGP). Uma vez finalizada, a configuração estabelecida pode ser armazenada no repositório de configuração e interpretada pela camada de segurança.

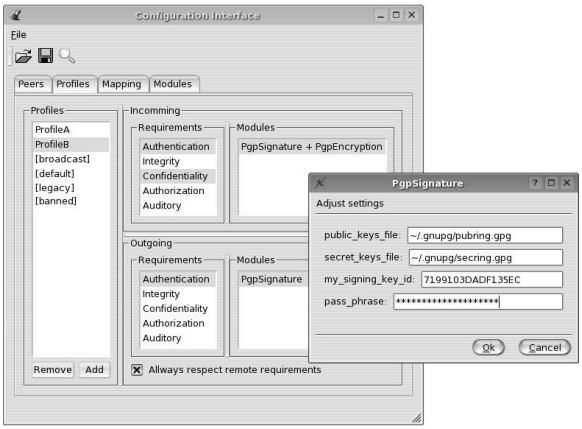


Figura 5. Interface gráfica exibida pelo assistente de configuração.

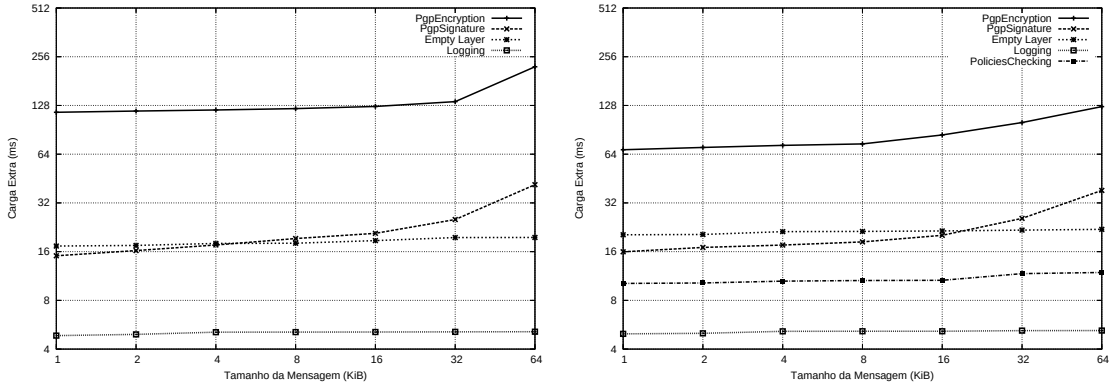
6. Avaliação Experimental

Apesar de ser clara a necessidade de segurança, particularmente em aplicações corporativas, a utilização de mecanismos de segurança introduz uma indesejável sobrecarga. Esta seção apresenta uma avaliação experimental com a implementação da P2PSL. O objetivo foi empregar esta implementação como uma prova de conceito da P2PSL, bem como realizar medidas da sobrecarga de latência induzida pela camada de segurança.

Experimentos foram conduzidos utilizando uma carga sintética de modo a isolar e medir a sobrecarga da P2PSL sem módulos e de cada módulo isoladamente. Eles foram realizados em uma estação Intel Pentium4 de 2.4 GHz com 1 GB de RAM. Apesar de terem sido investigadas diferentes opções de algoritmos de segurança e seus parâmetros chave, são reportados

aqui apenas os resultados principais. Para obter resultados estatisticamente significativos, cada experimento foi repetido 400 vezes. Adotando um nível de significância de 99%, o intervalo de confiança observado para qualquer experimento não ultrapassou 0,38 (milisegundos).

Uma vez que o tamanho das mensagens exerce um papel importante no desempenho da camada de segurança, os experimentos foram conduzidos usando diferentes tamanhos de mensagem: 1, 2, 4, 8, 16, 32 e 64 KB de dados, com conteúdo aleatório. A Figura 6 contém os resultados de latência média para (a) transmissão e (b) recepção de mensagens. Os valores apresentados nos gráficos se referem à P2PSL configurada com nenhum módulo de segurança (*Camada Sem Módulos*), e com cada módulo isoladamente[1] (*PgpSignature*, *PgpEncryption*, *Logging* e *PoliciesChecking*).



(a) transmissão (b) recepção

Figura 6. Sobrecargas introduzidas pela camada sem módulos e por cada módulo individualmente (sem camada).

Examinando a figura, observa-se que a camada sem módulos consome aproximadamente 20 ms tanto na operação de envio quanto na de recebimento, com praticamente nenhuma variação no que tange ao tamanho de mensagem. Conforme esperado, o módulo de criptografia PGP induz uma significativa sobrecarga por mensagem, cuja média chega a quase 250 ms quando criptografando mensagens de 64 KB. A sobrecarga depende do tamanho da chave; nos experimentos apresentados, optamos por usar 1024 bits. O módulo de assinatura PGP também induz uma sobrecarga substancial, alcançando 40 ms para a geração e verificação de assinaturas em mensagens com 64 KB. O módulo de verificação de políticas (aplicável apenas no recebimento de mensagens) apresenta uma baixa sobrecarga, entre 10 e 15 ms para uma única regra, mas a latência torna-se maior de acordo com o número de políticas a serem verificadas. O módulo de geração de *logs* induz uma baixa sobrecarga, abaixo de 6 ms, independente do tamanho da mensagem, porque nos experimentos o nível de detalhamento foi ajustado para *médio*, de modo que o conteúdo das mensagens não fosse escrito para o *log*.

De forma geral, as latências individuais foram grandes, tipicamente em torno de dezenas a centenas de milisegundos por mensagem. Esses valores não devem ser considerados isoladamente, mas combinados, uma vez que uma aplicação P2P tende a empregar múltiplas funcionalidades de segurança. Em um cenário que representa o pior caso, se todos os módulos fossem empregados e as mensagens fossem de 64 KB, a latência total por mensagem atingiria 491 ms, ou seja, aproximadamente 288 ms para o envio e 203 ms para o recebimento.

Para algumas aplicações, tais latências seriam inaceitáveis, como no caso de aplicações

[1] não incluíndo a sobrecarga da camada em si.

P2P interativas. Entretanto, este tipo de aplicação tende a usar mensagens pequenas (menores que 1 KB), caso em que a latência do pior caso é reduzida praticamente à metade, por volta de 272 ms (153 ms para o envio e 119 ms para o recebimento). Mesmo assim, considerando esta latência, a capacidade máxima de transmissão ficaria limitada pelo transmissor a seis mensagens de 1 KB por segundo.

Desta forma, a escolha dos módulos para um *peer* específico fica limitada não apenas pela existência de implementações para os módulos, mas também pela capacidade de processamento disponível para a aplicação P2P naquele nodo. Por outro lado, a latência de processamento introduz custos intrínsecos que têm sido há muito associados à implementação de segurança. Note-se que o objetivo deste artigo não é propor algoritmos de cifragem ou *hashing* mais eficientes. O estudo acima reforça a importância de flexibilidade e autonomia na escolha de quais módulos um *peer* irá empregar.

7. Estudo de Caso: Computação em Grade Baseada em P2P

Nesta seção introduzimos um estudo de caso, que foi feito para demonstrar tanto o conceito quanto à viabilidade técnica de nossa proposta. A P2PSL foi incorporada ao OurGrid [1], uma infra-estrutura de grade computacional baseada em redes P2P.

A motivação para a escolha do OurGrid neste estudo de caso é dupla. Primeiro, embora ele possua uma implementação madura de software e esteja sendo implantado em ambientes de produção [14], ele não possui uma infra-estrutura de segurança robusta, permitindo vários tipos de ataques ou mal uso do sistema. Segundo, infra-estruturas de computação em grade demandam diversos requisitos de segurança, o que nos permite sobrecarregar a camada de segurança.

O restante da seção está organizado como segue. Primeiro, apresenta-se uma visão geral do OurGrid. A seguir, descreve-se como a P2PSL foi incorporada no OurGrid e ilustra-se a instanciação de uma infra-estrutura de grade protegida. Ao implantar tal configuração almeja-se avaliar o comportamento da P2PSL com relação à *integração*, *isolamento*, *assimetria* e *implantação gradual*. Por fim, avalia-se a carga extra imposta pela P2PSL ao executar uma aplicação real de grade.

7.1. OurGrid e seu Protocolo de Operação

O OurGrid é um *middleware* P2P que permite a criação de um ambiente multi-institucional de computação em grade para a execução de aplicações *bag-of-tasks* [1]. Cada instituição em uma rede OurGrid possui um *peer representante* bem como um *escalonador de tarefas*, que gerencia os recursos locais. O *peer* age como um *broker* na rede P2P, tentando reunir recursos remotos sempre que os locais são insuficientes para atender a um pedido.

As principais interações do protocolo de operação do OurGrid estão ilustradas na Figura 7. Quando a demanda por recursos pelos usuários em uma organização (por exemplo, Peer1) excede a capacidade de computação disponível localmente, seu *peer* envia por *broadcast* mensagens com requisições CONSUMER_QUERY para os demais *peers* (mensagem 1 na figura). Se alguma instituição possuir recursos ociosos que satisfazem os requisitos do conjunto de tarefas a serem executadas, a requisição é respondida com uma mensagem PROVIDER_WORK_REQUEST (2). No exemplo mostrado na Figura 7, a instituição 2 responde à solicitação, enquanto que a instituição 3, não. Ao receber uma ou mais respostas, o Peer1 escolhe em quais instituições cada tarefa será executada, enviando uma mensagem PROVIDER_WORK_REQUEST_ACK (3). Por questões de simplicidade, o exemplo mostra um caso onde

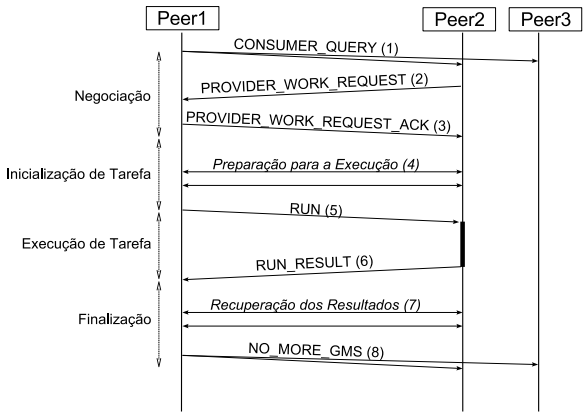


Figura 7. Protocolo usado pelos *peers* OurGrid para executar tarefas remotamente.

existe uma única tarefa. O Peer1 começa então a trocar mensagens com o *peer* que representa a instituição escolhida (Peer2 no exemplo), com o objetivo de preparar a execução da tarefa (4). Essa fase do protocolo compreende tanto a alocação de espaço temporário para executar a tarefa no recurso recebido quanto a transferência dos arquivos de dados e executáveis necessários. A tarefa é, então, executada (5). Ao encerrar (6), uma nova fase inicia, em que os dados resultantes são recuperados (7). Finalmente, após concluir todas as tarefas, o Peer1 envia uma mensagem *broadcast* para os outros *peers* informando que os recursos ociosos não são mais necessários para satisfazer a requisição inicialmente enviada (8).

7.2. Instanciação de uma Infra-Estrutura de Grade Protegida

Como a comunicação do OurGrid se apóia nas classes JXTA/JAL, sua adaptação para a P2PSL precisou apenas da substituição da classe `EZMinimalPeer` (provida pelo JAL) pela classe `SecurePeer` (disponibilizada pela P2PSL). Pelo fato de a classe `SecurePeer` estender a `EZMinimalPeer`, todos os métodos disponíveis nesta última continuaram disponíveis na `SecurePeer`. Portanto, não foi necessário modificar nem a aplicação nem o *middleware* subjacente de comunicação, tornando evidente a propriedade de *isolamento* da P2PSL.

Tendo incorporado a P2PSL no OurGrid, instanciou-se um ambiente real composto por doze *peers*. Por questões de simplicidade, caracterizamos um subconjunto do cenário completo constituído por quatro *peers* – a saber Peer1, Peer2, Peer3, e Peer4. A Figura 8 ilustra o cenário simplificado. O Peer1 representa uma instituição com tarefas computacionais pendentes e sem recursos disponíveis para executá-las. Peer1, Peer2, e Peer4 executam usando a camada de segurança. Enquanto Peer1 e Peer2 foram configurados para permitir comunicação mútua, o Peer4 foi preparado para bloquear mensagens do Peer1. A Tabela 1 sumariza a configuração dos *peers* que usam P2PSL. Neste cenário, executou-se diversos lotes de tarefas relacionadas à bioinformática.

Pôde-se observar a P2PSL fazer cumprir as políticas definidas pelo usuário de um *peer*. Por exemplo, a P2PSL executando no Peer4 descartou mensagens provenientes do Peer1, devido à política definida no primeiro. Também foram observados os conceitos de *integração*, empregando diversos aspectos de segurança usando um único sistema, e *assimetria*, especificando um conjunto de perfis para cada *peer* e confirmando que eles eram corretamente aplicados em cada canal de comunicação entre os *peers*.

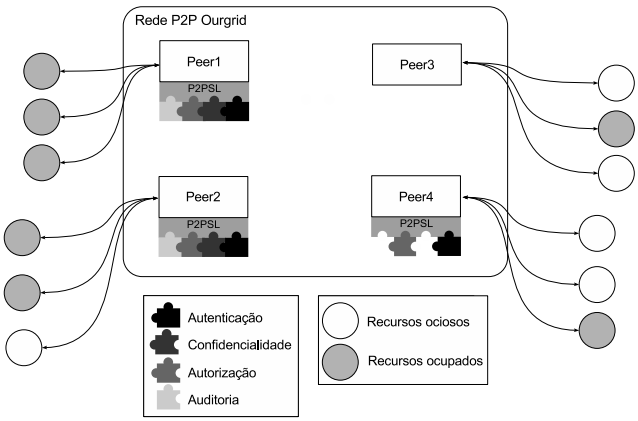


Figura 8. Visão simplificada do cenário do estudo de caso.

7.3. Avaliação de Sobrecarga

Usando a configuração recém descrita avaliou-se também a carga extra de comunicação induzida pela P2PSL na execução de uma aplicação de grade. A aplicação usada realiza computações sobre um modelo determinístico de dinâmica intracelular de um vírus [15]. Todos os *peers* executaram em nas mesmas estações citadas anteriormente. Os perfis de comunicação empregados pelos *peers* foram os mesmos enumerados na Tabela 1.

Na comunicação entre os Peer1 e Peer2 os módulos de segurança foram configurados como segue: assinaturas PGP com chaves DSA de 1024 bits, criptografia PGP com chaves El Gamal de 1024 bits, uma única política RBAC (*Role-Based Access Control*) para controle de acesso, e geração de *log* com nível médio de detalhamento (armazenando, para cada mensagem, identificadores de origem e destino, nomes dos módulos de segurança aplicados, tamanho da mensagem, bem como registros de tempo de envio e recebimento). Para a comunicação entre Peer1 e Peer3 (que não executa a P2PSL), o primeiro foi configurado para empregar a geração de *log* com alto nível de detalhe.

A Tabela 2 apresenta a carga extra (em milisegundos) induzida pela P2PSL na execução da aplicação recém mencionada. Os resultados correspondem à média de valores obtidos pela diferença entre o registro de tempo de uma mensagem chegando em um *peer* e o registro de tempo da próxima mensagem enviada por ele em reação à mensagem recebida. Os valores apresentados são (*i*) o custo computacional da P2PSL sozinha e (*ii*) o custo agregado para a P2PSL e o OurGrid processarem uma mensagem recebida e reagir a ela. Esses valores são organizados por fase do protocolo, a saber *Negociação*, *Inicialização de Tarefa*, *Execução de Tarefa* e *Finalização*, cujas mensagens foram ilustradas na Figura 7.

Pode-se observar a partir da tabela que o custo extra introduzido pela camada de segurança na comunicação entre Peer1 e Peer2 varia de 35% a 55%. Em contrapartida, nas mensagens trocadas entre Peer1 e Peer3 a carga extra oscila de 8% a 11% do custo total de comunicação. Essas diferenças são explicadas pelos requisitos de segurança empregados em cada canal de comunicação (restritivo no primeiro e relaxado no último). Os atrasos gerais induzidos não afetam substancialmente o desempenho da aplicação executando sobre o OurGrid, considerando o paralelismo atingido em sua execução e a predominância de processamento da tarefa em relação aos tempos envolvidos na troca de mensagens.

Tabela 1. Configuração dos *peers*.

Peer1	**Perfil A** *Envio*: Autenticação (Assinatura PGP) + Confidencialidade (Criptografia PGP) + Auditoria (geração de *log*) *Recepção*: Autenticação (Assinatura PGP) + Confidencialidade (Criptografia PGP) + Auditoria (geração de *log*) + Autorização (*Role-Based Access Control*) **Perfil B** *Envio*: Auditoria (geração de *log*) *Recepção*: Auditoria (geração de *log*) Perfil A: **Peer2**, **Peer4** Perfil B: **Peer3**
Peer2	**Perfil C** *Envio*: Autenticação (Assinatura PGP) + Confidencialidade (Criptografia PGP) + Auditoria (geração de *log*) *Recepção*: Autenticação (Assinatura PGP) + Confidencialidade (Criptografia PGP) + Auditoria (geração de *log*) + Autorização (*Role-Based Access Control*) **Perfil D** *Envio*: Auditoria (geração de *log*) *Recepção*: Auditoria (geração de *log*) Perfil C: **Peer1**, **Peer4** Perfil D: **Peer3**
Peer4	**Perfil E** *Envio*: - *Recepção*: Autenticação (Assinatura PGP) + Autorização (*Role-Based Access Control*) Perfil E: **Peer1**, **Peer2**, **Peer3**

Tabela 2. Carga extra de comunicação introduzida pela P2PSL em cada fase do protocolo, em milisegundos.

	Peer 1 - Peer 2		Peer 2 - Peer 1		Peer 1 - Peer 3		Peer 3 - Peer 1	
	P2PSL	Total	P2PSL	Total	P2PSL	Total	P2PSL	Total
Negociação	653	1182	753	1401	84	765	-	1590
Inicialização de Tarefa	502	991	555	1467	104	983	-	684
Execução de Tarefa	-	-	625	182864	-	-	-	222287
Finalização	685	1574	455	2721	47	568	-	2324

8. Considerações Finais

A diversificação e a disseminação de aplicações P2P, especialmente em cenários onde requisitos estritos de segurança devem ser atendidos (por exemplo, compartilhamento de conteúdo em ambientes corporativos e computação distribuída), dependem da disponibilidade de abordagens flexíveis para configurar e implantar mecanismos de segurança. Como mencionado ao longo do artigo, as abordagens existentes carecem de flexibilidade, visto que elas não provêem uma gama de requisitos de forma integrada. Além disso, elas exigem que o usuário/aplicação manipule uma interface de programação complexa e lide com um processo de configuração trabalhoso.

Para tratar as questões mencionadas acima propomos a P2PSL (P2P *Security Layer*). Ela permite a inclusão de funcionalidades de segurança em aplicações P2P, respeitando as questões de: (*i*) integração de aspectos de segurança em uma única aplicação; (*ii*) isolamento entre os mecanismos de segurança tanto da aplicação quanto do *middleware* subjacente; (*iii*) assimetria de segurança permitindo a cada *peer* escolher, independentemente dos demais, quais

requisitos devem ser respeitados; e (*iv*) implantação gradual da camada na rede P2P. Adicionalmente, P2PSL foi usada com sucesso em uma infra-estrutura de computação em grade baseada em redes P2P [1].

Futuramente a P2PSL será incorporada a outras aplicações *peer-to-peer*. Isto nos permitirá avaliar melhor a camada de segurança desenvolvida, especialmente sua generalidade e aderência a outras aplicações. Também pretende-se desenvolver módulos adicionais de segurança que abranjam outros requisitos de segurança, ampliando a aplicabilidade da P2PSL.

Referências

[1] N. Andrade, W. Cirne, F. V. Brasileiro, and P. Roisenberg, "OurGrid: An Approach to Easily Assemble Grids with Equitable Resource Sharing," in *Job Scheduling Strategies for Parallel Processing, 9th International, Workshop, JSSPP 2003*, pp. 61–86, June 2003.

[2] "Groove Virtual Office," Aug. 2005. http://www.groove.net/.

[3] G. Lawton, "Is Peer-to-Peer Secure Enough for Corporate Use?," *IEEE Computer*, vol. 37, pp. 22–25, Jan. 2004.

[4] L. Gong, "JXTA: A Network Programming Environment," *IEEE Internet Computing*, vol. 5, pp. 88–95, June 2001.

[5] "The Peer-to-Peer Trusted Library," Aug. 2005. http://sourceforge.net/ projects/ptptl.

[6] Y. Kim, D. Mazzocchi, and G. Tsudik, "Admission Control in Peer Groups," in *Second IEEE International Symposium on Network Computing and Applications*, p. 131, Apr. 2003.

[7] J. S. Park and J. Hwang, "Role-based Access Control for Collaborative Enterprise In Peer-to-Peer Computing Environments," in *Proceedings of the eighth ACM symposium on Access control models and technologies*, pp. 93–99, 2003.

[8] "Web Services Security (WS-Security) Specification," June 2004. http://www-106.ibm.com/developerworks/webservices/library/ws-secure.

[9] V. Welch and et al., "Security for Grid Services," *IEEE Twelfth International Symposium on High Performance Distributed Computing (HPDC-12)*, June 2003.

[10] S. R. Waterhouse, D. M. Doolin, G. Kan, and Y. Faybishenko, "JXTA Search: A Distributed Search Framework for Peer-to-Peer Networks," *IEEE Internet Computing*, vol. 6, pp. 68–73, Feb. 2002.

[11] "JAL - JXTA Abstraction Layer," Aug. 2005. http://ezel.jxta.org/jal.html.

[12] S. G. et al., "eXtensible Access Control Markup Language (XACML) Version 1.1. Committe Specification," Aug. 2003.

[13] J. F. da Silva, L. P. Gaspary, A. M. P. Barcellos, and A. Detsch, "Policy-based Access Control in Peer-to-Peer Grid Systems," in *6th IEEE/ACM International Workshop on Grid Computing*, Nov. 2005 (to appear).

[14] "Ourgrid Project," Aug. 2005. http://www.ourgrid.org/.

[15] R. Srivastava and L. and J. Yin, "Stochastic vs. Deteministic Modeling of Intracellular Viral Kinetics," *Theory Biology*, vol. 218, pp. 309–321, 2002.