BRUNO PONTES SOARES ROCHA

MIDDLEWARE DE SEGURANÇA ADAPTATIVO PARA COMPUTAÇÃO MÓVEL

Belo Horizonte
17 de dezembro de 2007

Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Programa de Pós-Graduação em Ciência da Computação

MIDDLEWARE DE SEGURANÇA ADAPTATIVO PARA COMPUTAÇÃO MÓVEL

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

BRUNO PONTES SOARES ROCHA

Belo Horizonte
17 de dezembro de 2007

UNIVERSIDADE FEDERAL DE MINAS GERAIS

FOLHA DE APROVAÇÃO

Middleware de Segurança Adaptativo para Computação Móvel

BRUNO PONTES SOARES ROCHA

Dissertação defendida e aprovada pela banca examinadora constituída por:

Ph.D. Antonio Alfredo Ferreira Loureiro – Orientador
Universidade Federal de Minas Gerais

Doutora Raquel Aparecida Freitas Mini
Pontifícia Universidade Católica de Minas Gerais

Ph.D. Gilles J. Barthe
INRIA Sophia-Antipolis, França

Belo Horizonte, 17 de dezembro de 2007

Resumo

O paradigma da computação móvel introduziu novos problemas para desenvolvedores e projetistas de aplicações. Desafios incluem a heterogeneidade de hardware, software e protocolos de comunicação, limitações de recursos e qualidade de canal sem fio variável. Nesse cenário, a segurança se torna uma preocupação fundamental para aplicações e usuários móveis. Requisitos de segurança para cada aplicação são diferentes, assim como as capacidades de hardware de cada dispositivo. Para tornar a situação ainda pior, as condições do meio sem fio podem mudar drasticamente com o tempo, causando grande impacto no desempenho e em garantias de qualidade de serviço da aplicação. Atualmente, a maioria das soluções de segurança para dispositivos móveis utiliza um conjunto estático de algoritmos e protocolos para serviços como criptografia e integridade de dados.

Neste trabalho propomos um serviço de segurança, que funciona como um middleware, com a habilidade de alterar dinamicamente os protocolos de segurança usados entre duas entidades computacionais. Mudanças são feitas de acordo com variações nos parâmetros do meio sem fio e do uso de recursos do sistema, capacidades de hardware, métricas de qualidade de serviço e níveis desejados de segurança definidos pela aplicação. Também apresentamos uma extensão que permite ao middleware considerar o valor semântico de pacotes sendo transmitidos com o objetivo de escolher melhor o protocolo a ser usado. Comparamos nossa solução com alguns protocolos estáticos de segurança bem difundidos e mostramos como nosso middleware é capaz de se adaptar em diferentes condições de meio e sistema.

Abstract

The mobile computing paradigm has introduced new problems for application developers. Challenges include heterogeneity of hardware, software, and communication protocols, resource limitations and varying wireless channel quality. In this scenario, security becomes a major concern for mobile users and applications. Security requirements for each application are different, as well as the hardware capabilities of each device. To make things worse, wireless medium conditions may change dramatically with time, incurring great impact on performance and QoS guarantees for the application. Currently, most of the security solutions for mobile devices use a static set of algorithms and protocols for services such as cryptography and hashes.

In this work we propose a security service, which works as a middleware, with the ability to dynamically change the security protocols used between two peers. Changes are made according to variations on the wireless medium parameters and system resource usage, hardware resources, application-defined QoS metrics, and desired "security levels". We also present an extension that allows our middleware to consider the semantic value of packets being transmitted in order to choose the best protocols to be used. We compare our solution to some widespread static security protocols and show how our middleware is able to adapt itself over different conditions of medium and system.

Aos meus pais, pela ajuda e compreensão incondicionais, por começarem a me guiar mesmo antes de eu saber que iria para algum lugar...

Agradecimentos

Primeiramente, aos meus familiares. Mãe e Pai, fundamentais e indispensáveis desde o começo de tudo, orgulhosos, "corujas", perfeitas estrelas-guias em minha vida. Vovó Darcy, da casa cheia de alegria onde aprendi a andar, a escrever, a nadar e a ser feliz, sempre vibrando e cheia de orgulho. Vovó Jane, pelo carinho, amor e exemplo, de bem com a vida sempre. Tati e Cacá, sempre presentes, sempre torcendo. Nando, Lí, Lela e Beto, sempre companheiros e orgulhosos. Miguel, de longe mas sempre aqui, me "forçando" a entrar de férias todo final de ano.

A todos que me ajudaram dentro do DCC. Ao professor Loureiro, não somente pela orientação acadêmica, mas pelos exemplos, conselhos e pela sincera amizade. A alguns outros professores que foram exemplos pessoais, notadamente os professores Virgilio, pelo companheirismo e ajuda, e Mário, pelo exemplo.

Aos amigos de mestrado e graduação, pela caminhada conjunta na "lama", farras, bobagens, almoços diários, churrascos. Marcus e Lília, pelo apoio fundamental bem no começo dessa caminhada. Aos amigos da Smart Price, campeões do bom humor em meio às crises. E, finalmente, mas não menos importante, aos amigos dos "esparros", sempre presentes nas farras de "rango, golo e esbórnia".

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Capítulo 1

Introdução

A computação móvel permite a criação de um novo paradigma computacional. Uma enorme gama de diferentes dispositivos computacionais podem se conectar uns aos outros com o objetivo de executar um grande conjunto de possíveis aplicações distribuídas. Transmissão de fluxos de vídeo ou voz, transferência de arquivos, notificação, localização e mapas e sistemas interativos e multimídias são meros exemplos de aplicações desenvolvidas para este ambiente. Além disso, a comunicação pode ser feita por diferentes protocolos de comunicação, como o IEEE 802.11 (também conhecido como *WiFi*), Bluetooth, IEEE 802.16 (*WiMAX*), GPRS e outros. O meio sem fio, no qual a comunicação é feita, é também suscetível a variações dramáticas e imprevisíveis na qualidade do canal de transmissão de dados.

Com toda essa heterogeneidade, é difícil criar especificações estáticas para qualquer tipo de hardware e software de computação móvel. Projetistas e desenvolvedores devem considerar limitações no hardware, demandas de segurança e QoS do software e propriedades do protocolo de comunicação. QoS (*Quality of Service*), tradução de "qualidade de serviço", significa métricas que devem ser respeitadas para que uma aplicação provenha suas funcionalidades mínimas. Como toda essa informação nem sempre está disponível, soluções integradas e adaptativas para dispositivos móveis são desejáveis.

Neste contexto, a especificação de protocolos de segurança se torna uma preocupação importante. Do ponto de vista do sistema, cada aplicação pode não apenas ter diferentes necessidades de segurança, como também diferentes demandas de QoS. Já do ponto de vista da aplicação, ela pode estar sendo executada em *hardware* com diferentes capacidades, bem como sobre diferentes protocolos de comunicação. As atuais soluções nativas dos protocolos de comunicação, como as presentes no IEEE 802.11 e no Bluetooth, propoem soluções conhecidamente incompletas ou falhas (Karygiannis e Owens, 2002), além de serem focadas apenas na camada de enlace da pilha de protocolos.

Uma solução comumente aplicada a este problema é a adoção de mecanismos de segurança na camada de aplicação, originalmente desenvolvidos para computadores de mesa e aplicações na Internet. Essa abordagem nem sempre é bem sucedida, à medida que os desafios postados pelos dispositivos móveis são diferentes daqueles de aplicações convencionais da Internet e, segundo Ravi et al. (2002), frequentemente criam um "buraco" entre os requisitos e as capa-

cidades de hardware. Mecanismos de segurança desenvolvidos para aplicações convencionais de Internet normalmente não consideram as propriedades distintas do meio sem fio, que são importantes fontes de informação que dizem respeito ao ambiente local (Li et al., 2006).

Para superar os problemas relacionados à heterogeneidade de hardware, software e comunicação, o desenvolvimento de middleware tem se tornado prática comum. Um middleware é uma camada de programação que media a comunicação entre aplicações e diretivas de sistema de mais baixos níveis, como acesso à rede. Seu papel é prover à aplicação diretivas para interação transparente com o sistema distribuído subjacente (Rocha et al., 2007; Capra et al., 2001).

1.1 Objetivo

Neste trabalho é proposto um middleware de segurança adaptativo que altera dinamicamente os protocolos de segurança usados entre dois pares comunicantes, de acordo com variações em um conjunto de parâmetros. O serviço em questão, nomeado **ASecMid**, sigla para o termo em inglês ***A**daptive **Sec**urity **Mid**dleware*, monitora parâmetros relacionados às condições do meio sem fio, uso de recursos do sistema, capacidades de hardware e definições da aplicação quanto a métricas de QoS e níveis de segurança desejados. É mantida uma grande coleção de protocolos de segurança que podem ser usados, e o mais viável para cada transmissão é escolhido de acordo com os parâmetros monitorados. Aplicações acessam o middleware de uma forma transparente, como um soquete de rede tradicional, não ciente dos protocolos de segurança aplicados. A idéia deste trabalho não é desenvolver novos algoritmos de segurança, mas sim usar os existentes da maneira mais eficiente possível, com base no "Princípio de Proteção Adequada" (Pfleeger e Pfleeger, 2006), que diz que se deve aplicar a segurança adequada para cada tipo de dado e contexto. Os resultados obtidos mostram a eficácia da solução.

É também proposta e testada uma integração do middleware com um mecanismo desenvolvido pelo aluno de mestrado do DCC/UFMG Daniel N. O. Costa (Costa, 2008) que tem como objetivo promover uma análise semântica de tipos de dados estruturados, de forma a determinar diferentes requisitos de segurança para diferentes partes do dado. São comparados os resultados com e sem a análise semântica dos dados.

A seguir são apresentadas a motivação e contribuição do trabalho, os fundamentos sobre segurança em redes usados no texto e os trabalhos relacionados.

1.2 Motivação

As principais motivações para este trabalho surgem da heterogeneidade de dispositivos, aplicações e protocolos de comunicação existentes no meio móvel e sem fio, como explicitado no começo deste capítulo. Diante disso, o desenvolvimento do middleware já é benéfico no sentido de prover uma única camada de segurança para diferentes contextos de hardware, software e comunicação.

Além disso, o trabalho toca num ponto importante ao se prover segurança para aplicações móveis: a relação custo/benefício entre prover segurança e manter a qualidade de comunicação e desempenho do sistema. O middleware tem o objetivo de trabalhar sobre essa relação, mantendo o equilíbrio entre segurança e desempenho de acordo com o configurado pela aplicação sobrejacente.

As características do meio em que um dispositivo está inserido muitas vezes são poderosas fontes de informação (Li et al., 2006). Com isso, a utilização de parâmetros de diversas fontes (aplicação, meio sem fio, recursos do sistema) é uma abordagem interessante para se criar um mecanismo adaptativo e que seja sensível às mudanças no contexto.

1.3 Contribuição

As contribuições deste trabalho são:

- Definição do problema de selecionar protocolos de segurança com base em métricas de segurança (força criptográfica) e uso de recursos (processamento, memória e rede);
- Proposição de uma metodologia para tratamento do problema, dividida em estágios e com objetivo de não proporcionar um *overhead* considerável ao sistema;
- Desenvolvimento de uma implementação de referência do mecanismo proposto, na forma de um middleware, pronto para ser executado no ambiente Linux e de código facilmente portável para outros sistemas;
- Avaliação da solução quanto a sua adaptabilidade em relação a mudanças no contexto de execução;
- Integração da solução proposta com um software de análise semântica de dados;
- Avaliação da solução com análise semântica dos dados, em comparação com o mecanismo sem a mesma análise.

1.4 Organização do Texto

Este documento é organizado da seguinte forma. Fundamentos teóricos e trabalhos relacionados são apresentados no Capítulo 2. No Capítulo 3 são discutidos os modelos considerados para rede, algoritmos e protocolos de segurança e parâmetros e métricas considerados, bem como um detalhamento do problema tratado pelo *middleware*. O *middleware* em si é apresentado no Capítulo 4, com detalhamento de suas funcionalidades, projeto e arquitetura e lógica de tomada de decisões. Os resultados experimentais são discutidos no Capítulo 5. No Capítulo 6 são apresentadas as conclusões e possíveis trabalhos futuros.

Capítulo 2

Fundamentos Teóricos

Neste capítulo são apresentados alguns fundamentos teóricos deste trabalho. Serão apresentados alguns tópicos fundamentais com o objetivo de esclarecer suposições do trabalho e como ele se encaixa nas áreas da Ciência da Computação estudadas. Para estudos mais aprofundados sobre os fundamentos, é indicada em cada subseção uma bibliografia didática.

2.1 Pilha de Protocolos de Rede

Na área de Redes de Computadores, a especificação e desenvolvimento de protocolos de rede é normalmente feita a partir de uma arquitetura em camadas. Desta forma, diferentes camadas de software têm funções específicas, com o objetivo de alcançar modularidade, facilidade de implementação e modificação de código e separação de funções.

No final da década de 70 surgiu então o modelo OSI, sigla do termo em inglês *Open Systems Interconnection Basic Reference Model* ou simplesmente *OSI Model*, que serviu de base para grande parte dos sistemas de redes de computadores desde então.

Hoje em dia, a pilha de protocolos TCP/IP se tornou muito difundida, devido ao fato de ser a arquitetura de rede da Internet. O padrão TCP/IP não segue a especificação OSI à risca, mas se baseia nela. A Figura 2.1 mostra um esquema da pilha de protocolos TCP/IP, comparada ao modelo OSI. A Figura 2.2 mostra alguns dos protocolos usados em cada camada do modelo.

A lógica proposta neste trabalho pode ser adaptada e implementada em diferentes modelos de rede, e usamos aqui o modelo TCP/IP como exemplificação do modelo mais difundido. A implementação de referência, usada para os experimentos e discutida na seção 4.5, coloca o middleware entre as camadas de aplicação e transporte do modelo TCP/IP. Além disso, ele interage com a sub-camada de acesso ao meio (MAC), da camada de enlace (ou de interface de rede), caracterizando-o como um software que utiliza "cruzamento de camadas", do inglês *cross-layer design* (Shakkottai et al., 2003). A Figura 2.3 ilustra como o middleware proposto neste trabalho se encaixa na pilha de protocolos do modelo TCP/IP.

Para um estudo aprofundado sobre protocolos de rede, modelo OSI e modelos de camadas, TCP/IP e estudo sobre protocolos reais utilizados em cada camada, recomenda-se a leitura

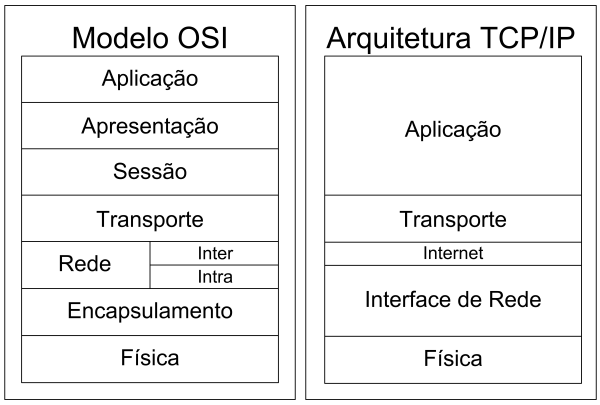


Figura 2.1: Camadas do modelo TCP/IP comparadas ao modelo OSI

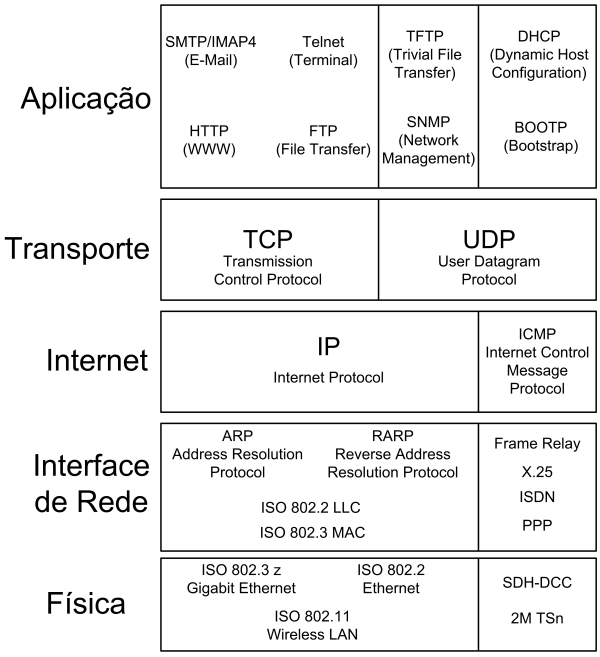


Figura 2.2: Protocolos mais famosos usados no modelo TCP/IP

da obra de Tanenbaum (2002).

2.2 Serviços de Segurança

O universo de segurança em computação conta uma grande quantidade de diferentes abordagens e técnicas. Segurança pode ser tratada em diferentes partes de um sistema computacional, como por exemplo:

- **Software:** técnicas de desenvolvimento e engenharia de software para prevenir ataques e erros que causam brechas de segurança, bem como tratamento de software malicioso (vírus, *worms*, etc...).

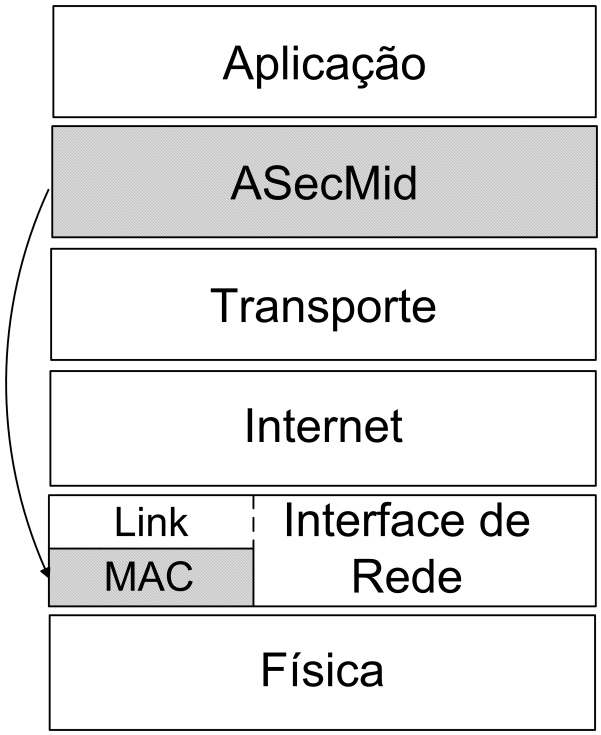


Figura 2.3: Localização do middleware proposto na pilha de protocolos TCP/IP

- **Sistemas Operacionais:** desenvolvimento de sistemas que fazem de forma segura tarefas como execução de múltiplos programas, gerenciamento de memória, autenticação de usuários e outros.
- **Bancos de Dados:** manter dados sensíveis de forma segura, impedindo que violações de controle de acesso ocorram.
- **Redes:** garantir que trocas de dados entre elementos distintos sejam seguras e não causem danos, perdas ou vazamento de informações, tanto do ponto de vista dos dados quanto dos dispositivos.

Este trabalho é direcionado para a **segurança em redes**, tratando, portanto, da comunicação entre diferentes dispositivos computacionais. Apesar da implementação de referência que iremos propor considerar vários aspectos de segurança de software, o propósito do trabalho é o estudo e avaliação da técnica de segurança em rede, podendo ela ser implementada de diversas maneiras.

Um mecanismo de segurança pode ter qualquer um dos seguintes objetivos: prevenção, detecção e ação. O primeiro indica que o sistema em questão tem o propósito de evitar que falhas de segurança ocorram. O segundo acontece quando o sistema tenta detectar brechas, falhas e ataques a um dispositivo, rede ou sistema de software. Por fim, o último objetivo consiste no sistema que executa ações específicas contra ataques e brechas de segurança, no sentido de tentar amenizar danos, parar o ataque e descobrir a identidade do infrator. Este trabalho tem o primeiro objetivo, o de **prevenção**.

Ao abordar segurança preventiva em redes de computadores, podem ser providos os seguintes serviços de segurança:

- **Autenticação:** asserção da identidade de uma entidade que envia dados pela rede.
- **Controle de Acesso:** garantia de que diferentes partes de um sistema (dados, diretivas, comandos, etc) são acessadas somente por entidades que possuem nível de acesso para tal e que, reciprocamente, uma entidade só possua acesso para as partes que lhes são permitidas.
- **Confidencialidade:** garantia de que dados serão lidos somente pela(s) entidade(s) a quem são destinados.
- **Integridade:** certificação de que dados alcançarão seu destino sem sofrerem modificações.
- **Não-repudiação:** certeza que nenhuma das partes envolvidas em uma comunicação irá negar seus serviços (aceitação de conexões, envio de dados, acesso a recursos, etc.) caso tenha recursos disponíveis para tal.

Este trabalho propõe um sistema de segurança que pode ser usado por virtualmente qualquer tipo de aplicação. Dado sua natureza genérica, este trabalho provê os serviços de **autenticação**, **confidencialidade** e **integridade**. Os serviços de controle de acesso e não-repudiação geralmente necessitam de maior conhecimento sobre a aplicação específica em que estão inseridos, para que possam ser providos. No intuito de manter a simplicidade da interface do middleware proposto com as aplicações, estes dois serviços não são tratados neste trabalho.

Por fim, podemos listar as categorias de possíveis ataques a um sistema de rede de computadores:

- **Ataques Passivos**

  - *Espionagem* [1]: a comunicação é "ouvida" por uma entidade não autorizada e dados sensíveis são revelados.

  - *Análise de Tráfego*: o padrão de comunicação (número de mensagens, tamanho, duração, etc) é analisado por uma entidade maliciosa que infere informações confidenciais a partir dessa análise.

- **Ataques Ativos**

  - *Mascaragem* [2]: uma entidade maliciosa assume uma falsa identidade, se passando por uma entidade confiável, e tendo acesso a dados e diretivas não autorizadas a ela.

  - *Reenvio*: o infrator capta mensagens enviadas na rede e as reenvia em um momento posterior, causando algum tipo de resposta das entidades que, por sua vez, irá causar uma falha de segurança.

[1] Em inglês, é usado o termo *eavesdropping*.
[2] Em inglês, é usado o termo *masquerade*.

- *Modificação*: entidade maliciosa captura mensagens enviadas, impede que elas cheguem ao seu destino, modifica seus conteúdos e então as reenvia, podendo assim causar perda de informação e induzir falhas de segurança.

- *DoS* [3]: acontece quando o infrator exaure os recursos de uma dada entidade, até que ela não consiga mais atender a requisitos de entidades legítimas.

Este trabalho não foca em nenhum tipo de ataque específico, e sim nos serviços abordados anteriormente. Ao prover os serviços de segurança de forma eficaz, o sistema acaba por tratar cada um dos ataques listados acima. Por exemplo, o serviço de confidencialidade está diretamente ligado ao ataque de espionagem, enquanto que o de integridade está ligado ao ataque de modificação. A combinação dos três serviços fundamentais (confidencialidade, integridade e autenticação), que são os providos pelo middleware proposto neste trabalho, permite, assim, tratar de cada um dos tipos de ataques acima.

Para um estudo detalhado sobre serviços de segurança, tipos de segurança em computação e debates sobre sua administração, questões éticas, privacidade e outros tópicos da área, recomenda-se o trabalho didático de Pfleeger e Pfleeger (2006).

2.3 Primitivas Criptográficas

A criptografia é a base da maioria dos serviços de segurança. Ela consiste em técnicas para codificar um dado de forma que entidades não-autorizadas não consigam decifrá-lo, mas ao mesmo tempo garantindo que o receptor pretendido consiga. A criptografia tem uma extensa história que data desde a antiguidade. Um estudo sobre a história da criptografia pode ser encontrada no trabalho de Kahn (1996). Não está no escopo deste trabalho a discussão de algoritmos individuais de segurança, incluindo possíveis falhas ou qualidades dos mesmos. Como será explicado adiante, o middleware proposto pode ser configurado para operar com o conjunto desejado de algoritmos, incluindo somente os confiáveis pelo administrador do sistema.

Os principais tipos de diretivas criptográficas são mostrados na Tabela 2.1. A seguir, cada tipo de diretiva é descrita em mais detalhe. Para um estudo mais aprofundado de diretivas criptográficas, bem como de todas as técnicas e algoritmos citamos nessa seção, recomendam-se os trabalhos de Stallings (2005) e Pfleeger e Pfleeger (2006).

Algoritmos de criptografia

Os algoritmos de criptografia consistem no tipo mais básico de diretivas criptográficas. Têm o objetivo de criptografar e decriptografar dados. Os algoritmos modernos são todos baseados no uso de chaves. Uma chave de criptografia é um bloco de dado de tamanho pré-definido usado no processo de criptação e decriptação do dado. Dessa forma, para poder decodificar um dado criptografado, é necessária a chave correspondente a usada no processo de codificação. O uso de chaves é proveitoso pois o mecanismo usado na criptografia pode ser de domínio público,

[3] Do inglês, *Denial of Service*, ou negação de serviço.

Diretiva	**Serviço provido**	**Exemplos de algoritmos**	**Usado pelo middleware**
Criptografia de chave simétrica	Confidencialidade	DES, 3DES, AES, RC4, RC2	Sim
Criptografia de chave pública	Confidencialidade	RSA, ECC	Sim
Distribuição de chaves	Necessário para qualquer algoritmo que use chaves	Diffie-Hellman, troca decentralizada, publicação de chave pública	Sim
Medidas contra análise de tráfego [4]	Confidencialidade	*padding* e *salting*	Não
Funções de *hash*	Integridade	MD5, SHA-1, Whirlpool	Sim
Algoritmos MAC	Integridade	HMAC-SHA-1, HMAC-MD5, CMAC-DES, CMAC-AES	Sim
Assinaturas digitais	Autenticação	DSA, ECC-sign, RSA-sign, RSA-priv	Sim
Controle de meio sem fio [4]	Confidencialidade	Saltos de freqüência, controle de potência de rádio	Não

Tabela 2.1: Tipos de diretivas criptográficas

encorajando assim o desenvolvimento e divulgação de mecanismos mais fortes, enquanto que a parte secreta do esquema fica sendo a chave usada.

Os algoritmos de criptografia modernos podem ser classificados quanto ao tipo de chaves utilizadas, em: algoritmos de chaves simétricas, algoritmos de chaves assimétricas. Os primeiros utilizam uma mesma chave tanto para criptografar quanto para decriptografar os dados. Esses algoritmos geralmente são rápidos e eficientes, mas criam o problema de se encontrar uma maneira segura para distribuir as chaves secretas. Os do segundo tipo, também conhecidos como algoritmos de chave pública, utilizam um par de chaves. Um dado é criptografado com uma chave do par e decriptografado com a outra. Com isso, uma das chaves de uma certa entidade normalmente é a "chave privada", sendo ela secreta. A "chave pública", por sua vez, pode ser divulgada para todos. Se uma entidade A deseja enviar um dado para a entidade B, ela criptografa o dado com a chave pública de B. A entidade B, por sua vez, é a única detentora da chave privada que irá decriptografar o dado recebido. Esses algoritmos são geralmente computacionalmente pesados e utilizados em operações de distribuição de chaves

[4] Não são diretivas criptográficas propriamente ditas, mas as consideramos nessa tabela por serem técnicas que trabalham sobre os dados com o objetivo de prover algum serviço de segurança.

simétricas ou autenticação de pacotes, descritas a seguir.

Outra classificação de algoritmos de criptografia é quanto ao tamanho de dados em que eles operam. A grande maioria dos algoritmos atuais são "cifras de bloco" pois trabalham em blocos de dados de tamanhos pré-determinados. Mensagens longas devem ser quebradas e (de)criptografadas bloco a bloco. Um outro tipo, mais comum em algoritmos de chave simétrica, são as cifras de fluxo, ou *stream ciphers*, que trabalham no dado bit a bit. O RC4 é um exemplo de algoritmo de fluxo.

Distribuição de chaves

Algoritmos que usam chaves criam o problema de como distribuir essas chaves. Se uma chave secreta é enviada por um canal aberto, um intruso pode facilmente interceptar essa comunicação e ter conhecimento do valor da chave. Dessa forma, é preciso que se tenham métodos seguros para a distribuição de chaves. Primeiramente, é preciso considerar que chaves assimétricas e simétricas propõem diferentes desafios de distribuição, devido às suas diferentes naturezas. Em segundo lugar, pode existir uma "chave mestra" que é um valor qualquer, possivelmente na forma de uma senha em texto, que os dois lados de uma comunicação têm conhecimento *a priori*. A existência de uma chave mestra facilita o processo de distribuição.

Se tratando de chaves assimétricas, somente chaves públicas devem ser divulgadas. Isso propõe uma maior facilidade, pois pode-se simplesmente publicar as chaves públicas. No entanto, dependendo do cenário em que uma entidade se encontra, pode haver problema de falsificação de identidade, com entidades publicando chaves enquanto afirmam serem outras entidades. Esse risco é muitas vezes resolvido com técnicas que supõem a existência de entidades centrais, reguladoras, que atestam autenticidade. Esse tipo de técnica não é considerada nesse trabalho, uma vez que ele é focado em redes *ad hoc*. Caso exista uma chave mestra entre entidades comunicantes, pode-se usar um esquema de distribuição decentralizada, usando a chave mestra com algum algoritmo de chave simétrica.

Chaves simétricas apresentam um problema maior de distribuição. No entanto, ao contrário das assimétricas, as chaves simétricas podem ser geradas mais facilmente, podendo ter qualquer valor dentro do tamanho de chave especificado. Com isso, na presença de uma chave mestra, além de se poder usar o esquema de distribuição decentralizada previamente mencionado, pode-se também derivar uma chave simétrica a partir da chave mestra, de modo que todas as partes envolvidas gerem a mesma chave sem a necessidade de comunicação. Caso não exista uma chave mestra, o mecanismo mais usado consiste em alguma aplicação de algoritmos de chaves assimétricas (previamente distribuídas) para transmitirem as chaves simétricas. Além disso, existem esquemas como o Diffie-Hellman, que usam técnicas parecidas com as de algoritmos de chave assimétrica para gerar chaves simétricas idênticas em duas entidades distintas, trocando apenas informações insuficientes para um intruso deduzir a chave gerada.

Medidas contra análise de tráfego

Além das medidas básicas de criptografia, *hashes* e códigos MAC (discutidos adiante), pode-se também adotar medidas específicas contra análise de tráfego. Essas consistem principalmente na técnica de *traffic padding* ("enchimento de tráfego"), que consiste em adicionar bits entre mensagens ou campos para tornar difícil a análise baseada em volume de comunicação, e também o *message salting* ("salgamento de mensagens"), que consiste em concatenar bits (*salt*, ou sal) ao final de mensagens antes do processo de criptação, com o objetivo de dificultar análises de mensagens de tamanho fixo. Por exemplo, duas mensagens idênticas produziriam o mesmo dado criptografado. Com a adição do "sal" antes de criptografar, elas produzem resultados diferentes, enganando a entidade maliciosa.

Como este trabalho apresenta um mecanismo que procura o equilíbrio entre segurança e uso de recursos (CPU, memória, rede), não consideramos essas duas técnicas de medidas contra análise de tráfego. O *overhead* causado por essas técnicas pode ser prejudicial para determinados contextos de computação móvel, como em condições de má qualidade de canal de transmissão.

Funções *hash*

São funções que trabalham sobre um conjunto de dados e produzem um bloco de tamanho fixo, geralmente menor que a mensagem em si. Não possuem chave e produzem sempre a mesma saída para um mesmo dado. Dessa forma, são úteis para verificar a integridade de um certo dado. O código *hash* de uma mensagem pode ser concatenado ao final da mesma, e o seu valor verificado pelo receptor. É uma das diretivas criptográficas mais leves computacionalmente.

Algoritmos MAC

São semelhantes aos códigos *hash*, mas a diferença prática é que os códigos MAC utilizam chaves. Os HMAC trabalham em conjunto com alguma função de *hash*, enquanto que os CMAC trabalham em conjunto com algum algoritmo de criptografia. São mais difíceis de serem "quebrados" que os códigos *hash*.

Assinaturas digitais

São técnicas para garantir a autenticidade de uma entidade transmissora. A maioria das técnicas consiste em criptografar mensagens, *hashes* ou códigos MAC usando a chave privada do transmissor. O receptor decriptografa o bloco em questão com a chave pública do transmissor, e com isso assegura sua identidade, sabendo que só ele pode ter feito a criptografia daquele dado. Variações dessa técnica usando diretivas dos algoritmos de chaves assimétricas RSA e ECC são comuns. Outra técnica comum são algoritmos como o DSA, que produzem um bloco de assinatura digital usando par de chaves geradas exclusivamente para este propósito.

Controle de meio sem fio

Em redes sem fio, pode-se usar controle relacionado ao meio de transmissão para garantir maior confidencialidade. O salto contínuo de freqüências ou canais de comunicação (*frequency/channel hopping*) é útil pois dificulta o trabalho de um intruso tentar espionar no canal de comunicação. Outra técnica comum é o controle da potência do rádio de transmissão, de forma que mensagens tenham potência para alcançar distâncias não maiores que as do receptor da mensagem. Como este trabalho propõe um mecanismo adequado a diferentes tecnologias de transmissão, foi optado por não trabalhar com controle de meio sem fio, dependente da tecnologia específica para a qual é aplicado.

2.4 Trabalhos Relacionados

Mecanismos de segurança padrões para redes sem fio têm sido amplamente estudados, com suas falhas discutidas na literatura. Patiyoot e Shepherd (1999) discutem as principais técnicas criptográficas para redes sem fio, incluindo o IEEE 802.11, ATM sem fio, GSM, UMTS e outras. Um estudo detalhado das falhas e vulnerabilidades dos serviços de segurança nativos providos pelo IEEE 802.11 (conhecido como WiFi) e pelo Bluetooth pode ser encontrado no trabalho de Karygiannis e Owens (2002). Bhagyavati et al. (2004) analisam técnicas de segurança para o protocolo IEEE 802.11 e suas variações (IEEE 802.11i e a família 802.1X) e Patiyoot (2002) estuda problemas de segurança em redes ATM sem fio.

Alguns estudos encontrados na literatura identificaram abordagens especiais para promover segurança em ambientes sem fio, apontando desafios para alcançá-los. Alguns desses desafios incluem o problema da capacidade de processamento restrita dos dispositivos móveis, como apontado por Ravi et al. (2002). Desafios específicos em relação ao protocolo IEEE 802.11 são discutidos por Arbaugh (2003) e Potter (2006), com o último focando nos pontos de acesso, ou *hotspots*.

Propostas têm sido feitas na tentativa de promover segurança considerando propriedades da computação móvel. Soliman e Omari (2005) propõem um sistema criptográfico focado no domínio móvel, usando algoritmos de cifra de fluxo. Mais recentemente, Li et al. (2006) propuseram um sistema que usa as propriedades do sinal de rádio para autenticação e confidenciabilidade na camada física da pilha de protocolos, delineando a necessidade de se trabalhar com projeto em múltiplas camadas nesse tipo de cenário.

Tripathi (2002) discute requisitos e desafios de projetar um sistema do tipo *middleware*. Em (Capra et al., 2001; Mascolo et al., 2002a) os autores identificam novos requisitos para um *middleware* que suporta mobilidade, discutem as principais diferenças entre os meios sem e com fio e categorizam e analisam algumas possíveis soluções para um *middleware* móvel. Eles também propõem o uso de um *middleware* reflexivo para computação móvel (Capra et al., 2002) e descrevem um exemplo para redes *ad hoc* (Mascolo et al., 2002b). Em (Capra et al., 2003) é apresentado um *middleware* que possibilita trocas de contexto baseadas em um conjunto de regras lógicas.

Alguns exemplos de propostas concretas de *middleware* de segurança também são apresentados na literatura. Focado em redes cabeadas, Foley et al. (2004) propõem um arcabouço para inter-operação com *middleware* de serviços Web padrões, como .NET, CORBA e EJB. Focado no meio sem fio, Corradi et al. (2005) descrevem um *middleware* que lida com problemas de controle de acesso usando agentes móveis para acessar redes cabeadas em nome de usuários móveis que se encontram fora de alcance ou temporariamente desconectados.

Até a data de confecção deste documento, não foi encontrado na literatura um trabalho existente que propõe a escolha dinâmica de protocolos de segurança de acordo com condições do meio sem fio e uso de recursos do sistema.

Capítulo 3

Modelos e Definições

Neste capítulo apresentamos alguns modelos considerados para este trabalho. São também definidos termos e conceitos utilizados ao longo do texto, bem como uma definição sucinta do problema a ser resolvido.

3.1 Modelo de Rede

O sistema proposto neste trabalho considera uma rede *ad hoc*, sem pressupor nenhum tipo de infra-estrutura. Essa rede possui uma distribuição heterogênea de dispositivos, aplicações de software sendo executadas e protocolos de comunicação sendo usados. Dispositivos podem ser móveis e o meio sem fio pode sofrer interferência e variações na latência de comunicação. Abaixo são apresentados exemplos de elementos dessa rede:

- **Dispositivos de hardware:** aparelhos celulares com poucos KBytes de memória e processadores limitados, PDAs (*Personal Digital Assistant*, também conhecidos como palmtops) com capacidades de processamento e memória variáveis, laptops com Gigabytes de memória e processadores de maior desempenho.
- **Aplicações em software:** aplicação Web com rajadas de comunicação intensa seguidas de momentos de pausa, fluxos de vídeo ou áudio com intensa comunicação bidirecional, transferência de arquivos com pacotes de tamanho fixo.
- **Protocolos de comunicação:** WiFi (IEEE 802.11, com ou sem mecanismos de segurança nativos, como WEP ou WPA), Bluetooth, WiMAX, GPRS.

3.2 Algoritmos e Protocolos de Segurança

Existe uma certa ambigüidade na literatura em relação aos termos "algoritmo de segurança" e "protocolo de segurança". Primeiramente, vamos analisar definições dos termos algoritmo e protocolo (Wikcionário, 2007).

`algoritmo` [1]: substantivo, **al.go.rit.mo** *masculino* (*plural:* algoritmos)

1. (*Matemática*) Conjunto de regras necessárias para resolução de um problema ou cálculo.
2. (*Informática*) Processo computacional bem definido, baseado num conjunto de regras, que executa uma determinada tarefa.

`protocolo`: substantivo próprio, **pro.to.co.lo** *masculino*

1. Conjunto de regras a observar em matéria de etiqueta, como as seguidas em cerimônias oficiais.
2. Processo verbal reunindo as resoluções de uma assembléia, de uma conferência.
3. (*Informática*) Código (linguagem) utilizado entre dois sistemas (computadores) para comunicarem entre si. A identificação de um documento.
4. (*Ciência*) Conjunto de regras, de condições relativas ao desenrolar de uma experiência.

Analisamos então definições mais específicas dentro da literatura científica. Segundo Savage (1987) e Gurevich (2000), "um algoritmo é um processo computacional definido por uma máquina de Turing". Já Thayer Fabrega et al. (1998) dizem que "um protocolo de segurança é uma seqüência de mensagens entre duas ou mais entidades na qual criptografia é usada para prover autenticação ou para distribuir chaves criptográficas para novas sessões".

Muitas das diretivas criptográficas usadas pelo middleware proposto neste trabalho podem ser classificas tanto como algoritmos quanto como protocolos. Podemos tomar como exemplo o caso do esquema Diffie-Hellman de distribuição de chaves. O esquema, proposto por Diffie e Hellman (1976), é considerado um algoritmo pois é perfeitamente descrito como um processo computacional seqüencial, definido nos moldes clássicos da computação, seguindo o modelo da máquina de Turing. Ao mesmo tempo, ele pressupõe troca de mensagens de natureza criptográfica entre dois pares, caracterizando-o como um protocolo de segurança.

Com isto, usaremos ao longo deste texto as seguintes definições:

Definição 3.1. *Chamamos de* ***algoritmo de segurança*** *um processo computacional que representa uma operação de segurança atômica sobre um bloco de dados, com um único objetivo criptográfico. Por exemplo, codificar/decodificar o dado, gerar um código hash, trocar uma ou mais chaves criptográficas.*

Definição 3.2. *Chamamos de* ***protocolo de segurança*** *um conjunto de um ou mais algoritmos, de acordo com a definição 3.1, que serão aplicados sobre blocos de dados em comum acordo por duas ou mais entidades, e que juntos podem prover um ou mais serviços de segurança.*

Dessa forma, o esquema Diffie-Hellman é definido neste trabalho como um **algoritmo de segurança**. Outros exemplos de algoritmos incluem o DES para criptografia, SHA-1 de códigos *hash*, qualquer algoritmo de códigos MAC ou qualquer outro esquema de trocas de

[1] Do árabe *al-khwarizmi*, alcunha de um famoso matemático árabe.

chaves. O acordo entre duas entidades de publicar suas chaves públicas de RSA, usar Diffie-Hellman para gerar chaves simétricas de 128 bits e a cada envio de dados gerar um código *hash* MD5, encriptá-lo com a chave privada, anexar essa informação ao pacote de dados e então criptografar tudo com um triplo DES usando os primeiros 112 bits da chave simétrica gerada, caracteriza um **protocolo de segurança**.

3.3 Parâmetros e Métricas

O middleware proposto trabalha com diferentes valores para poder escolher os melhores protocolos de segurança. Esses valores são divididos entre **parâmetros** e **métricas** e possuem diferentes propósitos. As definições 3.3 e 3.4 ilustram como esses termos são usados no contexto deste trabalho.

Definição 3.3. *Um **parâmetro** é um valor que carrega informações sobre o contexto em que o middleware está inserido, e que é usado como auxílio no processo de escolha de protocolos de segurança a serem utilizados.*

Definição 3.4. *Uma **métrica** é um valor associado a um algoritmo de segurança que define uma propriedade do mesmo. Métricas podem definir tanto propriedades funcionais dos algoritmos, quanto propriedades relacionadas a serviços de segurança. Uma métrica de um protocolo de segurança representa a agregação dos valores daquela métrica para cada algoritmo contido no protocolo.*

O middleware monitora parâmetros de diferentes fontes: meio sem fio, capacidades de hardware, uso de recursos do sistema e diretivas de QoS e níveis de segurança vindas da aplicação. Os parâmetros usados no sistema proposto são listados na Tabela 3.1. As unidades usadas têm como finalidade facilitar no processo de decisão de protocolos, detalhado no Capítulo 4. Dessa forma, são privilegiadas unidades percentuais, por consistirem de valores com mínimos e máximos conhecidos.

Cada algoritmo de segurança é associado a seis propriedades definidas por métricas. Dessas, três são métricas de força de segurança (ou força criptográfica): **confidencialidade**, **integridade** e **autenticação**, e outras três são métricas relacionadas a uso de recursos: uso de **memória**, tabelas de tempo de **processamento** e de *overhead* de **rede** (em quanto o pacote de dados original aumenta de tamanho). As métricas de força de segurança são representadas por números inteiros, de 0 a 100, cuja utilidade é comparar diferentes algoritmos. Como discutido no Capítulo 4, esses valores são definidos pelo administrador do sistema, através de um arquivo de configuração. Por exemplo, a criptografia AES deve ter um valor de confidencialidade maior que a DES, por ser um algoritmo conceitualmente mais forte e seguro. A diferença entre uma mesma métrica de dois algoritmos deve representar o quão mais forte um algoritmo é em relação ao outro. As métricas de uso de recursos são medidas, e possuem valores em bytes, para uso de memória e *overhead* de rede, e microssegundos para tempo de processamento. O *overhead* de rede e tempo de processamento são representados

Fonte	**Parâmetro**	**Unidade**
Meio sem fio	Mobilidade Qualidade do canal Latência Roteamento	porcentagem porcentagem microssegundos direto ou roteado
Capacidades de hardware	Memória Informação do rádio	bytes rádio usado (WiFi, Bluetooth, etc)
Uso de recursos	Consumo de memória Uso de CPU Bateria (se presente)	bytes porcentagem porcentagem
Aplicação (QoS)	Latência máxima Máximo uso de memória Máximo *overhead* de rede Máximo *overhead* de negociação	microssegundos bytes bytes bytes
Aplicação (níveis de segurança)	Confidencialidade Integridade Autenticação	níveis: desligado, opcional, desejável, mandatório ou crítico

Tabela 3.1: Parâmetros usados pelo middleware

por tabelas, com entradas para diferentes tamanhos de dados. A geração dos valores dessas métricas é discutida em detalhe no Capítulo 4.

Das listas apresentadas pode-se perceber facilmente que certos parâmetros se relacionam a certas métricas. Definimos isso como um mapeamento de cada métrica para um ou mais parâmetros de contexto. A definição formal desse mapeamento é apresentada na Seção 3.4, que trata da caracterização do problema. Na Tabela 3.2 apresentamos quais métricas são afetadas por cada parâmetro considerado para este trabalho. Por exemplo, um alto índice de mobilidade implica em maior número de possíveis outros dispositivos na região, relacionando-se à necessidades de confidencialidade e autenticação. No entanto, a mobilidade não se relaciona ao uso de memória, por exemplo.

Parâmetro	**Métricas**
Mobilidade	Confidencialidade e Autenticação
Qualidade do canal	Confidencialidade, Integridade e Rede
Latência	Processamento
Roteamento	Confidencialidade e Autenticação
Capacidade de memória	Memória
Informação do rádio	Confidencialidade, Integridade e Autenticação
Consumo de memória	Memória
Uso de CPU	Processamento
Bateria	Processamento, Rede e Memória
Latência máxima (QoS)	Processamento
Memória máxima (QoS)	Memória
Máximo *overhead* de rede (QoS)	Rede
Máximo *overhead* em negociações (QoS)	Rede
Nível de confidencialidade	Confidencialidade
Nível de integridade	Integridade
Nível de autenticação	Autenticação

Tabela 3.2: Mapeamento de parâmetros e métricas afetadas

3.4 Caracterização do Problema

Com as definições apresentadas nas seções anteriores podemos caracterizar o problema a ser tratado pelo middleware proposto. Primeiramente, vamos listar todos os elementos envolvidos no problema:

- Um conjunto de protocolos de segurança, de acordo com a definição 3.2.
- Cada protocolo possui um conjunto de métricas, com seis elementos, discutidos na Seção 3.3.
- Um conjunto de parâmetros relacionados ao contexto de execução, discutidos na Seção 3.3.
- Sabemos que cada parâmetro está diretamente ligado a pelo menos uma métrica, de acordo com a Tabela 3.2.

Com isso, podemos formalizar os elementos do problema e suas relações na definição 3.5.

Definição 3.5 (Elementos do problema)**.** *Seja S o conjunto de protocolos de segurança disponíveis. Denotamos M_i o conjunto de métricas do protocolo S_i, sendo $M_{i,j}$ a métrica j do referido protocolo. Denotamos também P como o conjunto de parâmetros relacionados ao contexto de execução. Sabemos que cada métrica $M_{i,j}$ é mapeada a um ou mais parâmetros, para todo protocolo S_i, ou seja:*

$$\forall j \exists k \forall i \; M_{i,j} \mapsto P_k$$

A existência ou não de mapeamento entre uma métrica e um parâmetro é sabida a priori.

Sabemos que o conjunto de métricas de cada protocolo é formado pelas métricas: confidencialidade, integridade, autenticação, memória, processamento e rede. Por conveniência, chamaremos essas métricas c, i, a, m, p e r. Com isso, estendemos o conceito de protocolo de segurança, aplicando-o mais ao problema tratado, na definição 3.6.

Definição 3.6 (Protocolo de segurança, no contexto do problema)**.** *Sobre o conjunto M_i de métricas do protocolo S_i, sabemos que:*

$$\forall i \; M_i = \{c_i, i_i, a_i, m_i, p_i, r_i\}$$

Simplificando, podemos denotar cada protocolo S como uma 6-tupla:

$$S = (c, i, a, m, p, r)$$

Por fim, definimos que cada uma das seis métricas tem um valor normalizado, com todas no mesmo intervalo, consistindo de números inteiros. As métricas c, i e a representam a força criptográfica do protocolo, com valores mais altos sendo mais desejáveis, enquanto que as métricas m, p e r representam uso de recursos de sistema, portanto sendo os valores mais baixos os mais desejáveis. Se especificarmos Q como a "qualidade" de um protocolo, podemos dizer que:

$$Q \sim \left(\frac{c + i + a}{m + p + r} \right)$$

No entanto, é sabido que protocolos criptograficamente mais fortes tendem a usar mais recursos de sistema, criando a relação:

$$c + i + a \sim m + p + r$$

Fazendo Q tender a 1 para a maioria dos casos.

Da definição acima podemos identificar o problema a ser resolvido. O objetivo do middleware proposto passa a ser o de selecionar, em tempo real, o "melhor" protocolo de segurança para um dado contexto. No entanto, segue da definição que os níveis de "qualidade" dos protocolos, seguindo as métricas usadas, tende a ser semelhante para todo o universo de protocolos. Com isso, a informação da definição 3.5 sobre o mapeamento de métricas a parâmetros de contexto passa a ter importância significativa na tentativa de resolver o problema. Assim, formalizamos o problema a ser tratado neste trabalho na definição 3.7.

Definição 3.7 (Problema da escolha de protocolos de segurança em tempo real, no modelo de força criptográfica e uso de recursos)**.** *Tendo um conjunto S de protocolos, de acordo com a definição 3.6, escolher o protocolo mais adequado para um determinado contexto. É também conhecido um conjunto P de parâmetros de contexto, e relações de mapeamento entre métricas dos protocolos e parâmetros de P, de acordo com a definição 3.5.*

A abordagem tomada, bem como a geração de protocolos e métricas, obtenção de parâmetros e outros detalhes são apresentadas na Seção 4.3.

Capítulo 4

O Middleware Proposto

Como discutido anteriormente, o principal objetivo do middleware proposto neste trabalho é o de escolher dinamicamente o protocolo de segurança mais adequado para comunicação entre dois pares. O protocolo escolhido pode ser trocado durante a execução, caso as condições de contexto variem. Em termos gerais, o middleware busca esse objetivo mantendo uma biblioteca de algoritmos de segurança existentes, avalia-os de acordo com as seis métricas definidas e monitora parâmetros de contexto para serem usados no processo de tomada de decisões.

Nesse capítulo discutimos a solução proposta. Primeiramente apresentamos como o middleware pode ser configurado por usuários, como administradores de sistema ou aplicações. Depois apresentamos a arquitetura, e então os algoritmos da lógica de seleção de protocolos. Mostramos ainda a extensão de análise semântica dos dados e finalmente discutimos nossa implementação de referência.

4.1 Configuração do Middleware

Um arquivo de configuração é lido pelo middleware toda vez que ele é inicializado. Esse arquivo contém a biblioteca de possíveis algoritmos de segurança a serem usados. Cada entrada no arquivo representa um algoritmo e informações sobre ele, incluindo aí os valores das seis métricas. Os valores das métricas de uso de recursos são automaticamente calculados e gerados durante a instalação do middleware (Seção 4.3.1). O comportamento do middleware pode ser configurado das seguintes formas:

1. Editando o arquivo de configuração e nele excluir quaisquer algoritmos que não devem ser usados.

2. Ainda no arquivo de configuração, editar os valores das três métricas de força criptográfica de cada algoritmo, definindo o quanto um algoritmo é mais forte que outro.

3. Em tempo de execução, a aplicação pode definir os parâmetros relacionados a níveis de segurança desejados (confidencialidade, integridade e autenticação), bem como os

limites de QoS (latência, uso de memória, *overheads* de rede por pacote e tempo de negociação).

Dessa forma, a aplicação e/ou administrador do sistema podem garantir que o middleware irá executar apenas algoritmos que eles confiam, e ainda definir quais deles são mais fortes. A aplicação irá também definir níveis de segurança mínimos de forma independente para os três serviços de segurança, além das limitações de QoS. Se, por exemplo, a aplicação define o nível de confidencialidade para *mandatório*, isso significa que o middleware sempre utilizará um protocolo que inclui criptação de dados. Opcionalmente, a aplicação (ambos os seus lados na rede) pode também suprir uma senha, que será usada para a geração de uma chave mestra criptográfica.

4.2 Arquitetura

O middleware é dividido em três partes fundamentais: conexão segura, máquina de segurança e controle de parâmetros. A conexão segura fornece a API para a aplicação acessar o middleware. Sua interface é muito similar a uma biblioteca de rede padrão (*sockets*) e múltiplas instâncias podem ser executadas ao mesmo tempo. A máquina de segurança é responsável por executar a lógica de tomada de decisões, bem como aplicar os protocolos de segurança em pacotes enviados ou recebidos pelas conexões seguras. O controle de parâmetros monitora todos os parâmetros e os torna disponíveis para a máquina de segurança. A Figura 4.1 ilustra a arquitetura do sistema.

O módulo de controle de parâmetros usa métodos diferentes para adquirir cada tipo de parâmetro. Parâmetros do meio sem fio são monitorados por um software separado que acessa *drivers* de interfaces de rede sem fio. A informação sobre o meio é guardada em um *buffer* compartilhado com o middleware. Capacidades de hardware e uso de recursos de sistema são obtidos através de primitivas do sistema operacional, enquanto que limitações de QoS e níveis desejados de segurança são passados pela aplicação, usando a API da conexão segura. É importante observar que os parâmetros podem ser divididos entre aqueles que possuem um valor fixo e os que possuem valor variável em tempo real. Capacidades de hardware, restrições de QoS e níveis de segurança têm valores fixos e só precisam ser adquiridos uma única vez. Por outro lado, informação do meio sem fio e uso de recursos do sistema são necessários em tempo real e precisam ser continuamente monitorados.

4.3 Algoritmos para Seleção de Protocolos de Segurança

Em função de tornar a seleção de protocolos mais fácil, o middleware divide o processo em estágios. Cálculos e testes computacionalmente pesados são feitos durante o processo de instalação no dispositivo alvo. Durante a execução, algum *overhead* de processamento é causado na inicialização do middleware, bem como ao estabelecimento de cada nova conexão, enquanto que o *overhead* de tomada de decisões para cada transmissão é o mínimo possível. O Algoritmo 4.1 apresenta uma visão geral do fluxo de execução (portanto desconsiderando

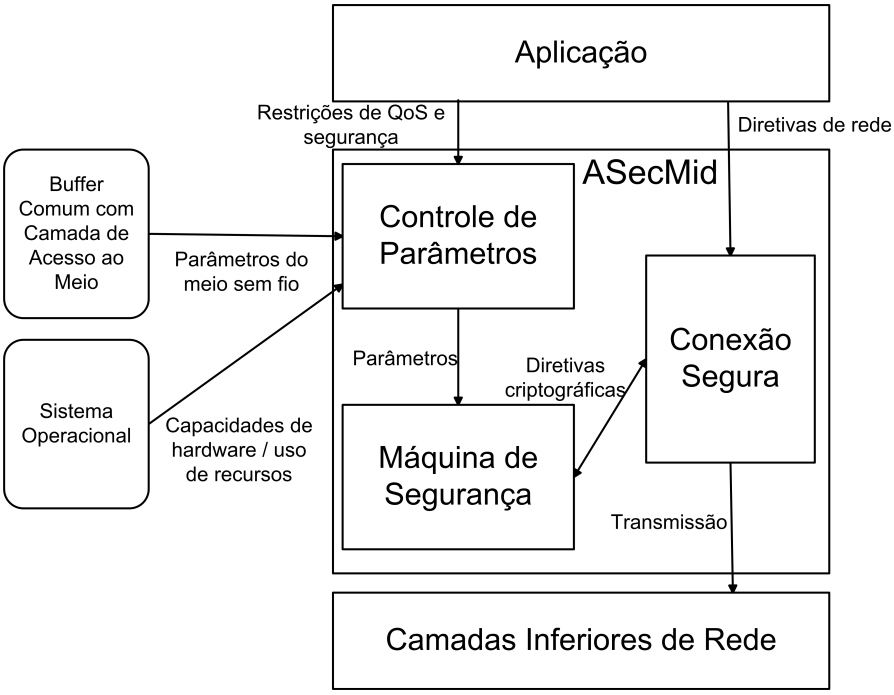


Figura 4.1: Arquitetura do middleware proposto

o processo de instalação) da máquina de segurança. Nas próximas subseções discutimos os passos individualmente.

Algoritmo 4.1 Fluxo de execução básico da máquina de segurança

```
1: Inicialização de algoritmos e protocolos de segurança
2: Aquisição de parâmetros permanentes
3: para todo conexões ativas faça
4:    Fusão de requisitos permanentes locais e remotos
5:    Seleção de protocolos válidos
6:    Escolha do conjunto de protocolos a serem usados
7:    Escolha dos algoritmos de troca de chaves
8:    Transmissão do conjunto de protocolos e algoritmos de troca de chaves a serem usados
9:    Geração e trocas de chaves necessárias
10:   enquanto Conectado faça
11:      se Envio de mensagem então
12:         Seleciona melhor protocolo, de acordo com parâmetros de tempo real
13:         Aplica protocolo e concatena seu código no começo da mensagem
14:         Envia mensagem para camadas de rede inferiores
15:      senão se Recepção de mensagem então
16:         Lê código do protocolo usado no começo da mensagem
17:         Aplica protocolo reverso
18:         Envia mensagem para camada da aplicação
19:      fim se
20:   fim enquanto
21: fim para
```

4.3.1 Instalação

Durante o processo de instalação o middleware lê o arquivo de configuração, que nesse ponto ainda não possui as métricas de uso de recursos de cada algoritmo, e calcula os valores dessas métricas. Para isso, cada algoritmo é testado com diferentes combinações de dados e chaves gerados aleatoriamente e de diferentes tamanhos. Os tamanhos de blocos e o número de execuções para cada tamanho podem ser configurados. Durante esses testes o middleware mede o uso de memória, tempo de processamento e *overhead* em bytes de cada algoritmo. Para o uso de memória é considerada a média de todas as execuções, enquanto que para as tabelas de processamento e *overhead* são consideradas as médias de execuções para cada tamanho de dados. Os resultados das medições são então escritos no arquivo de configuração, de forma que esse processo não tenha que ser repetido (a não ser que o sistema sofra uma modificação de hardware, o que invalidaria os resultados anteriores).

4.3.2 Inicialização

Quando o middleware é inicializado pela aplicação, sua principal tarefa é ler e processar o arquivo de configuração. Com isso, o middleware lê as definições de algoritmos do aquivo e então gera um conjunto com todos os possíveis protocolos de segurança (veja as definições 3.1 e 3.2, pág. 15). O procedimento de geração dos protocolos respeita regras básicas de consistência, como não gerar um protocolo que inclua dois algoritmos do mesmo tipo (como uma função *hash*). Além disso, são disponibilizados protocolos que não contêm algoritmos de todos os três serviços criptográficos tratados (confidencialidade, integridade e autenticação), incluindo protocolos com somente um algoritmo. Os algoritmos de troca de chaves são mantidos como algoritmos simples e não são adicionados a nenhum protocolo. Eles são tratados de forma um pouco diferente, como mostrado nas próximas seções. O sistema também gera um caso especial que representa a ausência de algoritmos de segurança, como um protocolo "vazio", para casos em que necessitará de usar transmissão aberta.

As métricas de cada protocolo são geradas através da fusão das métricas dos algoritmos individuais. Para cada métrica de segurança é considerado o valor mais alto entre os algoritmos presentes. Para as métricas de uso de recursos é feita a soma dos custos dos algoritmos individuais.

Finalmente, durante a inicialização, o controle de parâmetros adquire os parâmetros fixos. As capacidades de hardware são obtidas do sistema operacional e nesta fase espera-se que a aplicação chame as diretivas para definição de restrições de QoS e níveis de segurança.

4.3.3 Conexão

Para cada conexão estabelecida, o middleware executa quatro passos básicos: (1) combinar os parâmetros fixos, (2) selecionar o conjunto de protocolos a serem usados durante aquela conexão, (3) selecionar algoritmos para troca de chaves e (4) gerar e trocar as chaves. O primeiro passo, representado pela linha 4 do Algoritmo 4.1, é um processo simples que consiste

em os dois pares transmitirem seus parâmetros fixos e manterem, para cada parâmetro, o valor mais restritivo.

Depois disso, o middleware segue para selecionar o conjunto de protocolos a serem usados na conexão. O objetivo desse passo é diminuir o número de protocolos a serem considerados durante a transmissão, para que o processo de decisão em tempo real não seja muito custoso. Nos nossos experimentos, discutidos no Capítulo 5, o conjunto inicial consiste em mais de 7.000 protocolos, devido às diferentes possibilidades de combinações.

A primeira parte dessa escolha (linha 5 do algoritmo) consiste em eliminar protocolos cujas métricas não satisfazem os parâmetros fixos. Esse processo elimina protocolos que satisfaçam quaisquer uma das relações abaixo (lembre-se da nomenclatura das definições 3.5 e 3.6, pág. 19):

$$m > P_{capac.mem.} \tag{4.1}$$

$$m > P_{max.mem.} \tag{4.2}$$

$$\forall i \, (p_i > P_{max.latência}) \tag{4.3}$$

$$\forall i \, (r_i > P_{max.ovhd}) \tag{4.4}$$

$$(P_{nível.conf.} \geq mandatório) \wedge (c = 0) \tag{4.5}$$

$$(P_{nível.int.} \geq mandatório) \wedge (i = 0) \tag{4.6}$$

$$(P_{nível.aut.} \geq mandatório) \wedge (a = 0) \tag{4.7}$$

$$(P_{nível.conf.} = desligado) \wedge (c > 0) \tag{4.8}$$

$$(P_{nível.int.} = desligado) \wedge (i > 0) \tag{4.9}$$

$$(P_{nível.aut.} = desligado) \wedge (a > 0) \tag{4.10}$$

Depois dessa eliminação inicial de protocolos, o sistema adota uma estratégia gulosa para escolher uma pequena porção de protocolos a serem usados durante a conexão (linha 6). Neste ponto, o sistema escolhe da lista os seguintes protocolos:

- Os n_1 que têm maiores valores de cada métrica de segurança;
- Os n_2 que têm menores gastos de memória;
- Os n_3 que têm menores valores de processamento e os n_3 com menor *overhead*, para blocos de dados pequenos (10 bytes);
- Os n_4 que têm menores valores de processamento e os n_4 com menor *overhead*, para blocos de dados grandes (128 KBytes);
- Os n_5 que têm as maiores somas das três métricas de segurança; e
- Os n_6 que têm as menores somas das três métricas de segurança.

As constantes n_i são configuráveis e os valores usados são discutidos na Seção 4.5. Dessa forma, um pequeno grupo contendo protocolos diferenciados em termos dos valores das métricas é criado. Essa escolha é feita por somente um dos lados da comunicação (o que aceita a comunicação, por padrão), que então transmite o conjunto escolhido para o outro lado. Cada protocolo é numerado e uma assinatura é gerada para a transmissão.

A última decisão a ser tomada na fase de conexão é a escolha de algoritmos de troca de chaves a serem utilizados (linha 7). Primeiramente, o conjunto de protocolos selecionados no passo anterior é analisado para determinar o número e tipos de chaves necessárias. Para as chaves assimétricas, um único mecanismo é escolhido, priorizando aquele que entra dentro da restrição de bytes máximos de negociação e possui maior força criptográfica (soma das métricas). Caso a aplicação tenha fornecido uma chave mestra, será priorizada e escolha de um algoritmo que a use. Para as chaves simétricas, são escolhidos um número de algoritmos entre um e o número de chaves a serem escolhidas, priorizando algoritmos de acordo com a seguinte ordem de requisitos: (1) usa uma chave mestra existente *e* faz autenticação dos pares, (2) usa uma chave mestra existente, (3) autentica os pares, (4) usa criptografia assimétrica e (5) o restante. O Algoritmo 4.2 ilustra esse processo. A informação de bytes de negociação usados é uma propriedade típica dos algoritmos de distribuição de chaves, fazendo parte das informações do algoritmo lidas do arquivo de configuração.

Algoritmo 4.2 Seleção de algoritmos de distribuição de chaves

```
1:  // Distribuição de chaves assimétricas
2:  para todo Algoritmos de distribuição de chaves assimétricas faça
3:    se Algoritmo possui overhead de negociação dentro da restrição de QoS então
4:      se Existe chave mestra e o algoritmo a usa ou não existe chave mestra e algoritmo
        não a necessita então
5:        se Algoritmo possui maior força criptográfica que o atual então
6:          Seleciona algoritmo como atual para distribuição de chaves assimétricas
7:        fim se
8:      fim se
9:    fim se
10: fim para
11: // Distribuição de chaves simétricas
12: enquanto Número de algoritmos adicionados ≤ número de chaves necessárias faça
13:   // condição é checada a cada algoritmo adicionado
14:   se Existe chave mestra então
15:     Adiciona algoritmos que usam chave mestra e autenticam pares
16:     Adiciona algoritmos que usam chave mestra mas não autenticam pares
17:   fim se
18:   Adiciona algoritmos que autenticam pares
19:   Adiciona algoritmos que usam criptografia assimétrica
20:   Adiciona restante dos algoritmos
21: fim enquanto
```

Por fim, o lado que tomou as decisões envia o conjunto de protocolos e algoritmos de distribuição de chaves para o outro lado e então eles fazem a geração e distribuição de chaves. Os algoritmos de distribuição de chaves são executados um a um, na ordem que foram selecionados, para gerar cada um dos tamanhos de chaves necessárias. Se o número de algoritmo é menor que o de chaves necessárias, a lista de algoritmos é iterada de forma cíclica, reiniciando do começo. Se a aplicação tiver suprido uma chave mestra, o middleware a usa para criptografar todo o tráfego de negociação com um algoritmo pré-definido (AES, por padrão). Isso

é feito para evitar intrusos de fingirem ser uma instância do middleware, apesar desse tipo de proteção não ser o foco deste trabalho.

4.3.4 Transmissão

O passo final é executado a cada transmissão de dados. Quando um par decide transmitir um pacote, o middleware analisa os parâmetros de tempo real para escolher o melhor protocolo naquele momento, aplica-o sobre a mensagem e concatena o pacote gerado com o número do protocolo utilizado e os tamanhos de cada segmento, de modo que o outro lado possa aplicar o mesmo protocolo de forma reversa. Nesse ponto, o middleware possui um pequeno conjunto de protocolos (número depende das constantes n_i), cada um com suas seis métricas, e o grupo de parâmetros que pode influenciar em quais métricas devem ser mais importantes naquela situação. O fato dessa escolha ser crítica em termos de tempo caracteriza essa parte do processo como um *algoritmo online* (Borodin e El-Yaniv, 1998), tornando impraticável o uso de técnicas pesadas computacionalmente, como aquelas baseadas em otimização e pesquisa operacional. Para uma computação rápida e eficaz, nosso método obtém uma pontuação para cada protocolo. Dados os valores linearizados de cada métrica, a pontuação X de um protocolo i é calculada como:

$$X_i = (c \times W_c + i \times W_i + a \times W_a) - (p \times W_p + r \times W_r + m \times W_m) \tag{4.11}$$

onde W_j é o peso da métrica j.

Antes de calcular esses pesos, no entanto, o middleware deve checar se cada protocolo é aplicável às condições de tempo real. Por exemplo, o *overhead* causado por um protocolo pode ser abaixo das restrições de QoS para tamanhos de dados de até 10 KBytes, mas acima caso contrário. Desta forma, o middleware descarta protocolos que satisfaçam qualquer uma das seguintes condições:

$$m > min(P_{max.mem.}, P_{mem.livre}) \tag{4.12}$$

$$p[tamanho] + P_{latência} > P_{max.latência} \tag{4.13}$$

$$r[tamanho] > P_{max.ovhd} \tag{4.14}$$

Caso não exista a entrada *tamanho* nas tabelas de tempo de processamento e *overhead*, é feita uma interpolação simples com os valores das duas entradas mais próximas (superior e inferior).

Depois disso cada uma das seis métricas tem seu valor linearizado (definido dentro do intervalo de 0 a 100). As métricas de segurança já possuem valores entre 0 e 100 por definição. Tempo de processamento é definido como a porcentagem do "tempo restante" disponível para o protocolo, ou seja, a razão entre a entrada na tabela de processamento e a diferença entre a latência máxima e a atual latência da rede. *Overhead* de rede é linearizado como porcentagem do tamanho da mensagem atual, e uso de memória como porcentagem da atual memória disponível pelo sistema ou restrição de QoS (o que for menor).

O próximo passo é a determinação de cada um dos pesos das métricas, usados no cálculo da pontuação dos protocolos. Sabendo do mapeamento entre métricas e parâmetros (Tabela 3.2, pág. 18), o valor linearizado de cada parâmetro é multiplicado por uma constante, que representa o peso K daquele parâmetro sobre o peso W daquela métrica na pontuação final X. Desta forma, o peso de uma métrica i é calculado pela seguinte equação, onde K_{P_j,M_i} é a constante que representa o impacto do parâmetro j no peso da métrica i:

$$W_i = \sum_j P_j \times K_{P_j,M_i} \tag{4.15}$$

Com os pesos propriamente calculados, o middleware itera sobre o conjunto de protocolos com o objetivo de encontrar a maior pontuação. Apesar de todo o processo parecer complicado, ele consiste em apenas calcular valores por aritmética simples e então iterar sobre um pequeno conjunto de protocolos, descartando os não aplicáveis e mantendo registro do que possui a maior pontuação. As constantes K, assim como as n, são configuráveis e os valores utilizados são discutidos na Seção 4.5, sobre a implementação.

4.4 Integração com Módulo de Análise Semântica de Dados

Uma segunda parte desse trabalho, proposta como uma extensão ao mecanismo, consiste na integração com o trabalho de Costa (2008). A integração é tratada separadamente na implementação e nos experimentos. As próximas subseções detalham como foi feita a integração.

4.4.1 A Camada de Configuração de Segurança

A camada de configuração tem como objetivo permitir à aplicação configurar a segurança que deseja obter em cada mensagem que transmite. Com isto, a aplicação pode personalizar suas transmissões indicando os níveis de segurança desejados para cada trecho de dados.

A aplicação acessa a camada através de um arquivo de anotação, que contém a descrição semântica dos dados a serem enviados. Essa descrição é feita através de uma linguagem de anotação baseada em XML. A linguagem de anotação permite a definição de *configurações de segurança*, que consistem em conjuntos de valores para os requisitos de segurança e QoS considerados pela camada de configuração. Esses requisitos são:

- **Segurança**
  - Confidencialidade
  - Integridade
  - Autenticação
- **QoS**
  - Latência máxima
  - Taxa de transmissão mínima
  - Máximo *overhead* por pacote
- **Restrições em casos de impossibilidade de cumprir requisitos**

- Transmitir de qualquer forma
- Agendar envio em más condições
- Retransmitir em caso de perda

Dessa forma, o arquivo de anotação possui definições de várias configurações desses requisitos. Além dessas definições, possui um fluxo que define como será o padrão de comunicação daquela aplicação, indicando qual configuração utilizar para cada trecho de dados. Como exemplo, a aplicação pode especificar uma transmissão de dados FTP, configurando que mensagens do tipo `ack`, usados só para confirmação, não necessitem de ser criptografados, aumentando o desempenho do sistema. Um exemplo dessa linguagem é mostrado abaixo.

Exemplo da linguagem de anotação em XML

```
<security-definition>
  <global-configuration>
    <requirements>
      <authenticity level="Disabled" priority="Low" />
      <confidentiality level="Disabled" priority="Low" />
      <integrity level="Disabled" priority="Low" />
      <maximum-latency priority="Low" time="0" />
      <minimum-transmission-rate priority="Low" rate="0" timeout="0" time-unit="Microseconds
          " />
      <maximum-byte-overhead-per-packet priority="Low" overhead="0" />
    </requirements>
  </global-configuration>
  <configuration name="high-sec">
    <requirements>
      <authenticity level="Critical" priority="High" />
      <confidentiality level="Critical" priority="High" />
      <integrity level="Critical" priority="High" />
    </requirements>
  </configuration>
  <configuration name="mid-sec">
    <requirements>
      <authenticity level="Desired" priority="Medium" />
      <confidentiality level="Desired" priority="Medium" />
      <integrity level="Desired" priority="Medium" />
    </requirements>
  </configuration>
  <configuration name="low-sec" />
</security-definition>

<message-definition>
  <message name="special-msg">
    <mask-comparator first-byte="1" mask-size="2" match-mask="0xFFFF" match-value="0x0103" /
        >
  </message>
  <message name="end-loop">
    <size-comparator operator="equal" value="4096" />
  </message>
</message-definition>

<transmission-model repeat="true">
  <send configuration="high-sec" label="first-msg" />
  <receive configuration="low-sec" size="4096" />
  <send configuration="mid-sec" label="second-msg" />
  <receive configuration="low-sec" size="4096" />
  <if>
    <if-condition>
      <compare-message label="second-msg" name="special-msg" />
    </if-condition>
    <if-then>
      <send configuration="high-sec" />
      <receive configuration="low-sec" />
```

```
    </if-then>
    <if-else>
      <loop count="10">
        <stop-condition>
          <compare-message label="sent-msg" name="end-loop" />
        </stop-condition>
        <send configuration="low-sec" label="sent-msg" />
        <receive configuration="low-sec" />
      </loop>
    </if-else>
  </if>
</transmission-model>
```

4.4.2 Adaptação da Lógica de Seleção de Protocolos

Para fazer a integração com a camada de configuração, algumas modificações devem ser feitas na lógica de seleção de protocolos do middleware. Dos requisitos da camada de configuração, o middleware considera os de segurança e os de latência e *overhead* máximos. Os requisitos de configuração de segurança também são definidos em cinco níveis, ficando fácil a conversão para as unidades do middleware.

Com a introdução da camada de configuração, os cinco requisitos considerados deixam de ser parâmetros fixos e passam a ter características de tempo real, podendo mudar de valor a cada transmissão. O middleware passa a ter, em sua API, diretivas especiais para conexão e aceitação de conexão para dados com arquivos de anotação definidos. Com isso, a lógica de seleção de protocolos usando a camada de configuração deve diferir da lógica apresentada nas seções anteriores nos seguintes pontos:

1. Na início da conexão, ambos os lados fornecem o arquivo de anotação ao middleware, que por sua vez o repassa à camada de configuração, onde ele será processado.
2. Ainda na fase de conexão, o middleware deve escolher um conjunto de protocolos que satisfaça todas as possíveis configurações de segurança contidas na anotação. Para isso, a camada de configuração fornece ao middleware o conjunto de todas as configurações especificadas na anotação. Dessa forma, o middleware executa o processo de escolha de conjunto de protocolos uma vez para cada configuração definida, e ao final do processo funde os protocolos selecionados, eliminando redundâncias.
3. Na fase de transmissão, o middleware passa a adquirir toda vez os valores para os parâmetros de níveis de segurança, latência e *overhead* máximos, tratando-os como parâmetros de tempo real. Além disso, no momento de checar a aplicabilidade do protocolo às condições de tempo real, deve considerar esses parâmetros também. Dessa forma, na hora de verificar as condições das equações 4.12 a 4.14, deve-se também re-verificar as das equações 4.3 a 4.10, antes só verificadas em tempo de conexão.

Dessa forma o middleware passa a considerar restrições de QoS e níveis de segurança para cada pacote individual, enquanto continua considerando os parâmetros de meio sem fio e uso de recursos, sem perder assim sua capacidade de adaptação. No Capítulo 5 apresentamos os resultados experimentais, comparando o middleware proposto com a extensão de análise semântica de dados.

4.5 Implementação de Referência

Foi feita uma implementação de referência do middleware proposto, com o objetivo de se fazer simulações e servir como base para possíveis implementações em diferentes dispositivos.

A implementação foi feita na linguagem de programação C++, orientada a objetos. Cada um dos três módulos do middleware foi implementado como uma classe, sendo o controle de parâmetros definido como um *singleton* (Gamma et al., 1995). A aquisição de dados do sistema é feita através de diretivas de sistemas operacionais Linux.

Quanto às constantes K e n, temos um total de 24 combinações. Elas interferem no processo de decisão do middleware, mas não de forma independente, com o valor de uma constante influindo no valor de outra. Além disso, elas são influenciadas pela forma que os parâmetros variam. Com isso, fica impraticável realizar experimentos para determinar valores ótimos para as constantes. O que foi feito, nesse caso, foram execuções manuais e um tratamento empírico para encontrar bons valores para cada constante. Os valores usados na implementação podem ser alterados por diretivas de compilação. A Tabela 4.1 mostra os valores das constantes n da Seção 4.3.3, pág. 24. A Tabela 4.2 mostra as constantes K, sendo que alguns parâmetros não possuem nenhum mapeamento com valor válido, pois são usados somente para restrições (equações 4.1 a 4.14), e os parâmetros que possuem valores discretos (como níveis de segurança), possuem um valor K para cada valor de P possível.

Constante	**Valor**
n_1	1
n_2	1
n_3	3
n_4	3
n_5	3
n_6	3

Tabela 4.1: Constantes de seleção de conjunto de protocolos

A implementação possui dois modos de operação: execução e simulação. O primeiro é o modo normal de funcionamento. No segundo os parâmetros são lidos de um módulo de simulação, que por sua vez lê um arquivo de entrada para determinar como cada parâmetro será simulado. Possui também um módulo separado, que chama o middleware usando um gerador de tráfego que pode ter alguns tipos diferentes de padrão de geração. Outra diferença acontece quando o middleware não consegue selecionar um protocolo que atende às restrições de QoS e níveis de segurança. No modo de execução, o middleware retorna o controle de execução para a aplicação, reportando uma exceção do tipo "*deadlock* de requisitos". Já no modo simulação o middleware simplesmente passa a utilizar transmissão direta, sem protocolos de segurança, e registra a partir de que momento o *deadlock* ocorreu. As simulações são descritas com mais detalhes no Capítulo 5. Foram feitos ainda um conjunto de *scripts* nas linguagens C-Shell e Perl para tratamento de dados.

Parâmetro \ Métrica	c	i	a	p	r	m
Mobilidade	0,75	-	0,75	-	-	-
Qualidade do canal	0,6	0,8	-	-	1	-
Latência	-	-	-	-	-	-
Roteamento	0;0,5;0,75	-	0;0,25;0,5	-	-	-
Memória	-	-	-	-	-	-
Rádio	0,15;0,25;0,5			-	-	-
Uso de memória	-	-	-	-	-	1,25
Uso de CPU	-	-	-	1,25	-	-
Bateria	-	-	-	1	1	0,4
Restrições de QoS	-	-	-	-	-	-
Níveis de segurança	0;0;0,4;0,4;0,7			-	-	-

Tabela 4.2: Constantes K de seleção de protocolos em tempo real

Primitivas criptográficas de baixo nível foram usadas de bibliotecas de terceiros [1]. Nossa implementação possui uma biblioteca que consiste de 90 algoritmos de segurança que durante a inicialização geram 7.755 protocolos. É bom lembrar que não é escopo deste trabalho discutir forças e fraquezas de algoritmos individuais. Como mencionado no começo do Capítulo 4 (pág. 21), a lista de algoritmos a ser utilizada pelo middleware pode ser previamente configurada. Nossa lista inclui algoritmos como:

- **Criptografia**: Variações de DES, 3DES, AES, Camellia, RC4, RC2, CAST5, RSA, ECC, ...
- **Integridade**: MD5, MD4, SHA-1 (e suas variações), modalidades de HMAC e CMAC, ...
- **Autenticação**: DSA/DSS, variações de codificação com chave privada de RSA e ECC, ...
- **Distribuição de chaves simétricas**: Diffie-Hellman, distribuição decentralizada baseada em chave mestra, acordos de criptação de chave pública, ...
- **Distribuição de chaves assimétricas**: Publicação direta da chave pública, distribuição decentralizada baseada em chave mestra, ...

[1] `http://libtom.org`, `http://openssl.org` e `http://dragongate-technologies.com/products.html#borZoi`

Capítulo 5

Resultados Experimentais

O grande número de variáveis envolvidas neste trabalho torna a experimentação uma tarefa difícil. Além disso, o uso de um simulador como o NS-2 apresenta dificuldades, uma vez que seria necessário adquirir muitas informações tanto de um sistema operacional quanto de um meio sem fio simulados, e é difícil encontrar um simulador que tenha todos os parâmetros necessários (o próprio NS-2 não simula uso de recursos do sistema).

Portanto, foi adotada uma estratégia diferente: fazer execuções reais do middleware, mas usando parâmetros simulados. Como explicado na Seção 4.5, o modo simulação de nossa implementação é capaz de criar diferentes perfis de simulação de parâmetros. Como os parâmetros são simulados, eles não causam nenhum dano real à comunicação sendo feita. Dessa forma, de cada experimento analisamos as relações entre a variação dos parâmetros, distribuição de protocolos usados pelo middleware (bem como a força criptográfica dos mesmos) e o *throughput* (quantidade de dados transmitidos) da comunicação. A análise do *throughput* é diretamente relacionada ao quão seguros são os protocolos sendo aplicados (protocolos mais seguros tendem a possuir mais algoritmos, e serem mais pesados computacionalmente) e, portanto, têm dois significados importantes: (1) o middleware deve ter um *throughput* variável no tempo, refletindo mudanças dinâmicas de protocolos, e (2) o *throughput* do middleware deve ser proporcional ao quão estrito é o perfil em que ele executa, isto é, em cenários simulados muito restritos ele deve usar protocolos mais "leves", tendo maior *thoughput*, enquanto que em simulações de muitos recursos disponíveis ele deve usar protocolos mais "fortes" e lentos. Os parâmetros são variados de forma idêntica nos dois lados da comunicação.

Nas próximas seções são apresentados os resultados experimentais. Primeiramente apresentamos execuções do ASecMid em diferentes cenários comparadas a execuções sem aplicação de protocolos e com emulações de quatro mecanismos de segurança estáticos: PGP e S/MIME de segurança de e-mails aplicados às transmissões, IPSec, e o protocolo SET para transações bancárias. Em seguida, comparamos a versão padrão do middleware com o a versão estendida, usando análise semântica de dados da camada de configuração de segurança. Cada experimento foi executado 33 vezes, e as médias tiradas dos valores observados. Nos gráficos que comparam *throughputs* de diferentes mecanismos, barras de erro são para intervalos de confiança de 90%, usando o teste-t estatístico.

5.1 ASecMid comparado a mecanismos estáticos

Nesta seção cada experimento é apresentado com quatro gráficos, todos em função do tempo em segundos no eixo x: o primeiro mostra a variação de *throughput* dos mecanismos considerados no eixo y, o segundo é um *zoom* do primeiro, sem a curva relativa à transmissão sem segurança, o terceiro mostra a variação de alguns parâmetros simulados no eixo $y1$ e da soma de força criptográfica dos protocolos usados pelo middleware no eixo $y2$, e o quarto compara o *throughput* do middleware no eixo $y1$ com a força criptográfica total do mesmo no eixo $y2$. Em caso de que se deseje reproduzir os experimentos, os detalhes numéricos sobre variações de parâmetros e protocolos de segurança escolhidos podem ser encontrados no Apêndice A. Lá também se encontram gráficos de variação de força criptográfica do middleware, incluindo barras de intervalos de confiança de 90%.

Uma execução típica acontece com parâmetros de meio sem fio e uso de recursos sofrendo uma variação pseudo-aleatória, isto é, cada parâmetro assume um valor aleatório por um período de tempo aleatório, e depois recalcula o valor e o próximo período de tempo, tudo dentro de limites aceitáveis. A única exceção fica para o nível de bateria, que tem um valor decrescente ao longo do tempo. A Figura 5.1 mostra esse tipo de experimento, com um tráfego de dados também pseudo-aleatório (cada lado envia uma rajada com número aleatório de pacotes, entre 1 a 15, de tamanhos aleatórios entre 1 a 64K bytes). Percebe-se pela Figura 5.1(a) como que o middleware possui um *throughput* muito mais variável que os métodos estáticos. Isso se deve ao fato de que, ao longo da execução, oito protocolos de segurança diferentes foram empregados. Nesse experimento a aplicação configurou o middleware com restrições de QoS bem estritas, de forma que foram usados protocolos mais "leves" que os dos mecanismos estáticos, daí o *throughput* do middleware ser maior. É também importante observar na Figura 5.1(d) como que a força criptográfica (soma das três métricas de segurança do protocolo sendo usado, varia de 0 a 300) tem uma relação inversa com o *throughput*. Como esperado, isso significa que protocolos mais fortes são computacionalmente mais pesados, implicando menor *throughput*, e vice-versa.

Apresentamos agora um exemplo de execução com variação mais controlada de parâmetros. Na execução representada pela Figura 5.2 as restrições de segurança e QoS foram mantidas em níveis baixos, vários parâmetros tiveram valores fixados e somente o uso de recursos de sistema (uso de CPU e de memória) foram aumentando com o tempo. A Figura 5.2(c) mostra como o middleware foi selecionando protocolos mais leves (baixando a força criptográfica) à medida que o uso de recursos foi aumentando (uso de memória, omitido no gráfico, varia de forma idêntica ao uso de CPU). As Figuras 5.2(a) e 5.2(d) mostram também como o *throughput* sobe com o tempo, refletindo uso de protocolos mais leves. Foram usados quatro protocolos de segurança diferentes durante a execução.

Mostramos agora um exemplo com outro tipo de parâmetro variando com o tempo. No experimento da Figura 5.3 as condições são as mesmas do anterior, exceto que agora os parâmetros de uso de recursos de sistema são fixos (e baixos), enquanto que as condições do meio sem fio (qualidade do canal, mobilidade e latência) vão piorando com o tempo. Do

mapeamento de parâmetros e métricas usados, sabemos que os parâmetros de meio sem fio se relacionam tanto a métricas de segurança como de uso de recursos (Tabela 3.2, pág. 18). Com isso, a queda na qualidade do meio pode causar tanto uma subida da força criptográfica usada (condições de segurança deterioraram) ou uma descida (condições de rede pioraram). Nesse caso, o comportamento do middleware fica determinado principalmente pelas restrições de QoS. Como nesse experimento elas são praticamente nulas, o middleware tende a empregar seus protocolos mais fortes. Isso pode ser observado na Figura 5.3(c), que mostra a força criptográfica aumentando à medida que as condições de meio se deterioram. Nas Figuras 5.3(a) e 5.3(d) pode-se perceber como que o *throughput* cai bastante com uso de protocolos muito fortes. Ao longo da execução três protocolos são usados, incluindo um de força criptográfica igual a 281, o mais forte gerado na biblioteca utilizada, que criptografa pacotes com o algoritmo RSA com chave de 2048 bits, calcula código CMAC usando AES com chave de 256 bits e ainda código de autenticação DSA.

Se aumentarmos as restrições de QoS e níveis de segurança da execução anterior podemos observar um comportamento diferente. No caso da Figura 5.4, o middleware tende a dar mais peso para os custos de recursos à medida que as condições do meio pioram. Com isso, pode-se perceber pela Figura 5.4(c) como que os protocolos usados vão sendo cada vez mais fracos. A partir de aproximadamente 35 segundos de execução, no entanto, o middleware não consegue mais encontrar protocolos que satisfaçam as restrições de QoS e os altos níveis de segurança exigidos, e acontece o "*deadlock* de requisitos", discutidos na Seção 4.5, pág. 31. Com isso, o middleware passa a transmitir sem segurança, e o *throughput* passa a ser como o de uma transmissão direta, como mostrado na Figura 5.4(a). Antes do *deadlock* foram usados quatro diferentes protocolos. Como nesse experimento foi usado uma simulação de tráfego FTP, o lado transmissor (transmite pacotes de 64 KB) usou dois protocolos diferentes dos dois usados pelo receptor (envia mensagens `ack` de 1 byte). Isso acontece devido ao fato de protocolos causarem tempos de processamento e *overhead* diferentes para tamanhos de dados distintos.

Um outro exemplo do acontecimento do *deadlock* pode ser visto na Figura 5.5. Com altas restrições de QoS e níveis de segurança desejados, os parâmetros de uso de recursos vão aumentando com o tempo. O middleware usa somente protocolos fortes (Figura 5.5(d)), até que ao final da execução não consegue mais atender aos altos requisitos de QoS e segurança e ocorre o *deadlock*. Isso mostra como deve haver consistência nas configurações de métricas. Executar um forte protocolo de segurança, com altas restrições de QoS e condições ruins do contexto de execução pode se tornar tarefa impossível. Foram usados cinco protocolos até o *deadlock*.

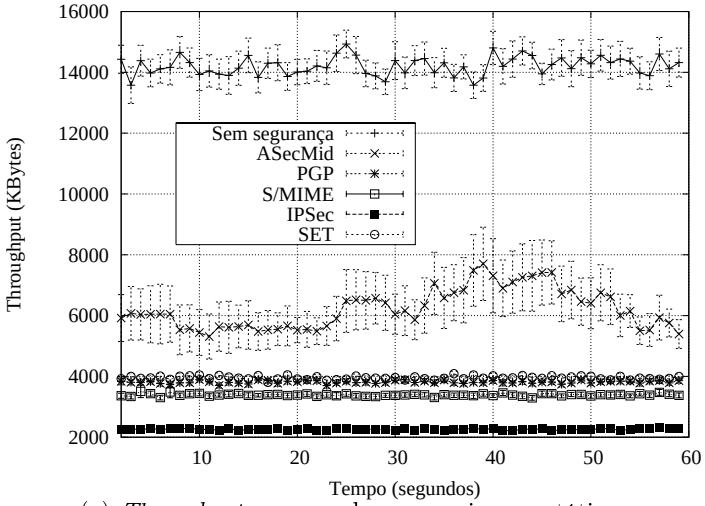
No próximo experimento, mostrado na Figura 5.6, foram aplicadas restrições de QoS, mas não de níveis de segurança (o middleware fica livre para usar a força criptográfica que quiser). Todos os parâmetros possuem variação pseudo-aleatória, com exceção da bateria, decrescente. É possível observar na Figura 5.6(c) como que a queda na bateria faz com que o middleware passe a aplicar protocolos mais fracos, ainda que a variação seja pequena (perceba a escala do eixo $y2$). Também se trata de um experimento com simulação de tráfego de FTP, e somente o lado receptor (que envia `acks` de 1 byte) foi influenciado pela queda de bateria, diminuindo

o seu tamanho de chave utilizada ao longo da execução. As Figuras 5.6(a) e 5.6(d) mostram como o *throughput* sobe com o tempo. Foram usados três protocolos distintos.

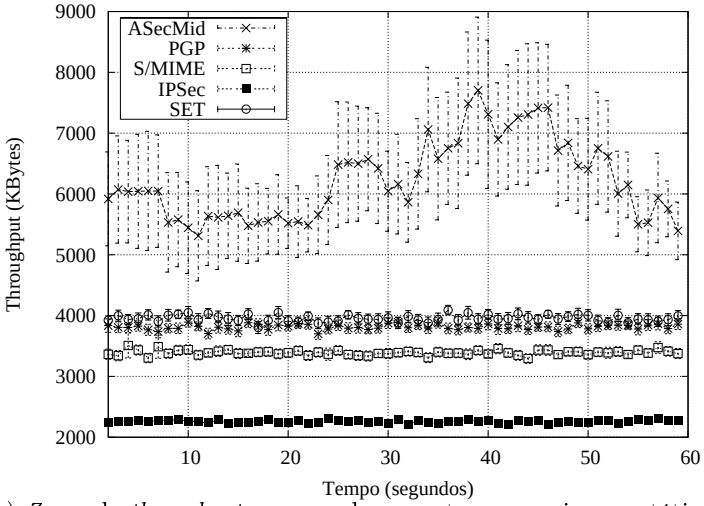
Se invertermos as restrições do experimento anterior, aplicando altas restrições de níveis de segurança mas retirando as de QoS, obtemos o experimento da Figura 5.7. Nesse caso, a queda de bateria não foi suficiente para forçar o middleware a reduzir a força criptográfica, uma vez que as restrições o obrigam a usar protocolos fortes e não impõem restrições de QoS relacionadas ao uso de recursos. Da Figura 5.7 percebe-se que o *throughput* é mais baixo que os mecanismos estáticos, por serem usados protocolos fortes. A variação pseudo-aleatória dos parâmetros (exceto bateria) combinada ao padrão pseudo-aleatório de tráfego faz com o que o middleware tenha alta variação do número de protocolos usados, tendo usado sete ao longo do tempo de execução.

Na Figura 5.8, usamos as condições dos dois últimos experimentos mas fazendo médias das restrições usadas para QoS e níveis de segurança. Dessa forma, para esses dois tipos de restrições foram usados valores médios (não excessivamente restritivos, mas consideráveis). Os resultados são um meio termo dos dois últimos experimentos. A queda de bateria só faz efeito nos 10 segundos finais, quando atinge valores muito baixos (Figura 5.8(c)). No resto do experimento o middleware usa protocolos com força criptográfica em torno de 208, mantendo *throughput* semelhante ao dos mecanismos estáticos (Figura 5.8(a)). Foram usados ao todo quatro protocolos de segurança diferentes.

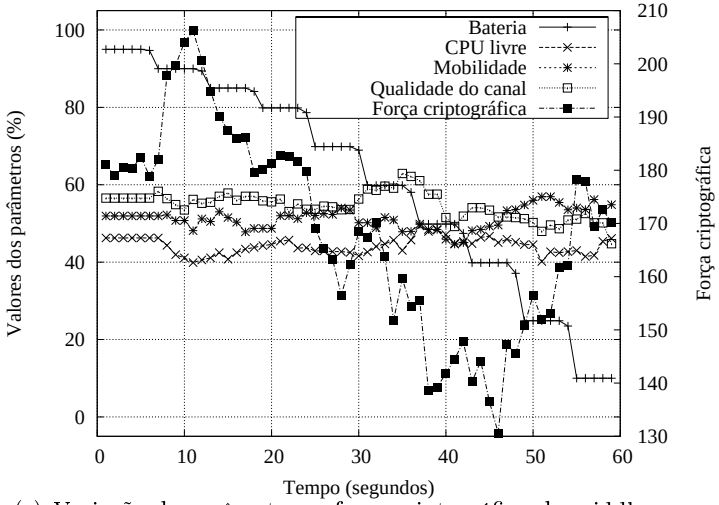
Finalmente, tratamos uma situação estática no experimento da Figura 5.9. É importante observar que mesmo quando o meio sem fio e o uso de recursos não variam, o middleware se adapta às condições fixas, ou seja, restrições de QoS e segurança e capacidades de hardware. Nesse caso ele manteve um mesmo protocolo de segurança durante toda a execução, que não teve variação de parâmetros, com exceção da bateria (Figura 5.9(c)), que não foi suficiente para induzir modificações de protocolos, devido às altas restrições de segurança.



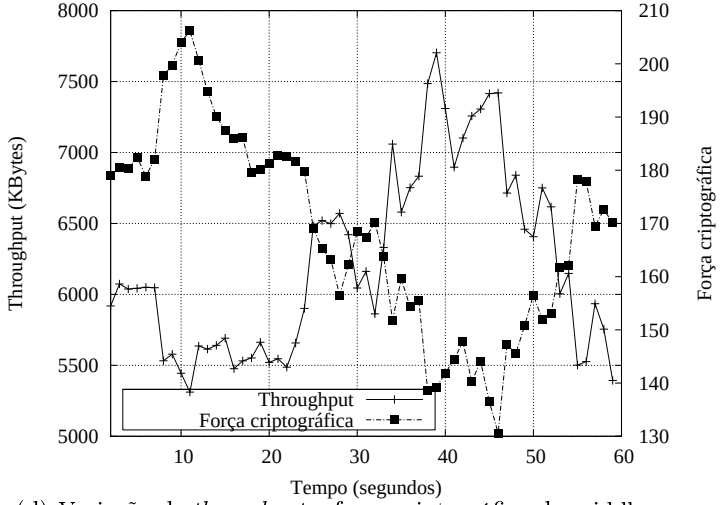
(a) *Throughput* comparado a mecanismos estáticos



(b) *Zoom* do *throughput* comparado somente a mecanismos estáticos

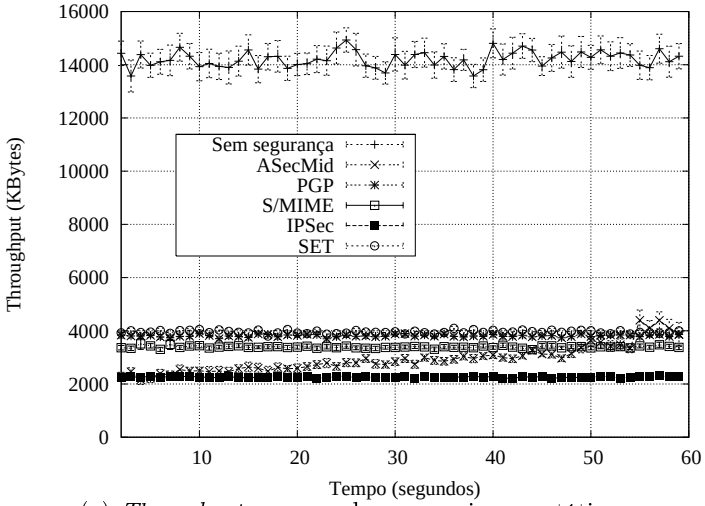


(c) Variação de parâmetros e força criptográfica do middleware

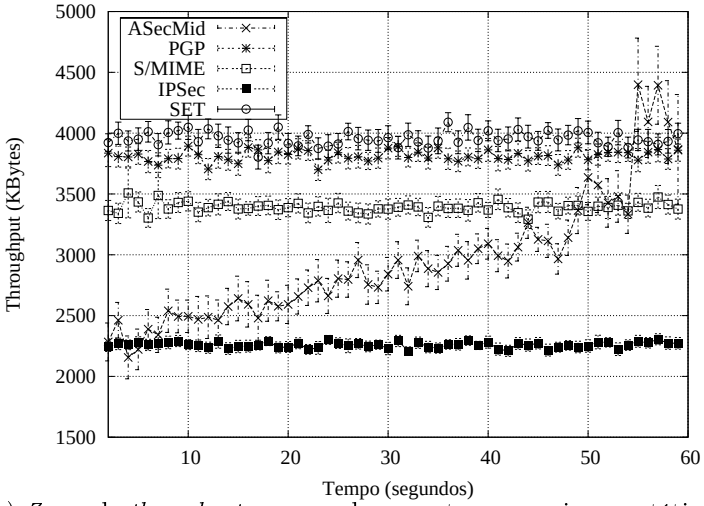


(d) Variação de *throughput* e força criptográfica do middleware

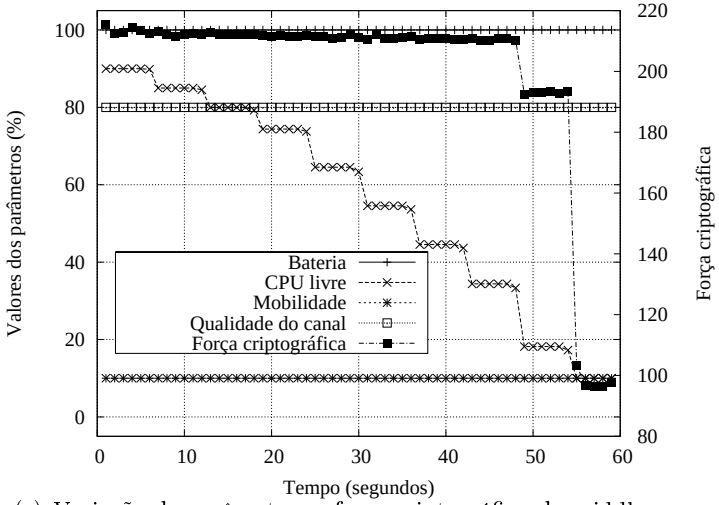
Figura 5.1: Execução típica, com restrições de QoS e níveis de segurança altos, e variação aleatória de parâmetros



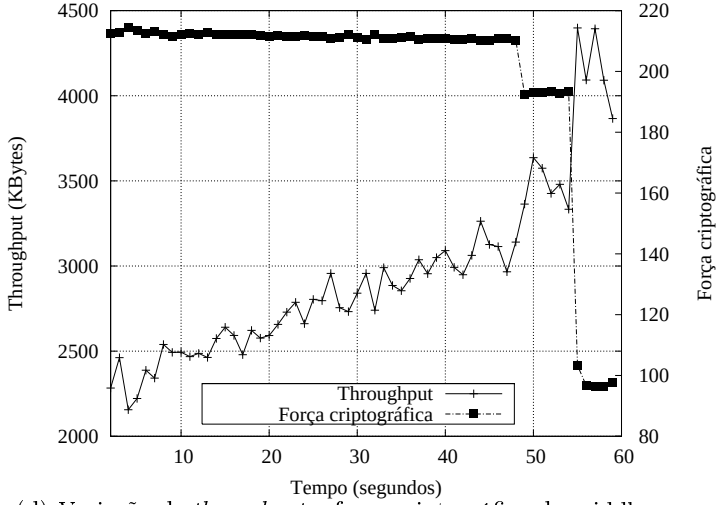
(a) *Throughput* comparado a mecanismos estáticos



(b) *Zoom* do *throughput* comparado somente a mecanismos estáticos

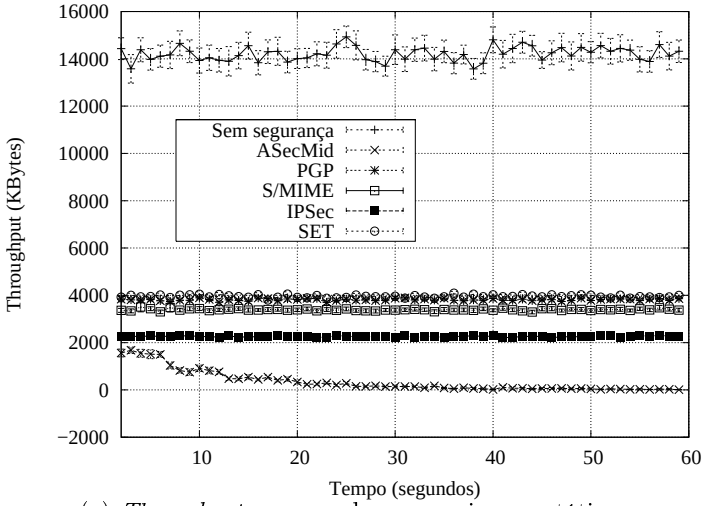


(c) Variação de parâmetros e força criptográfica do middleware

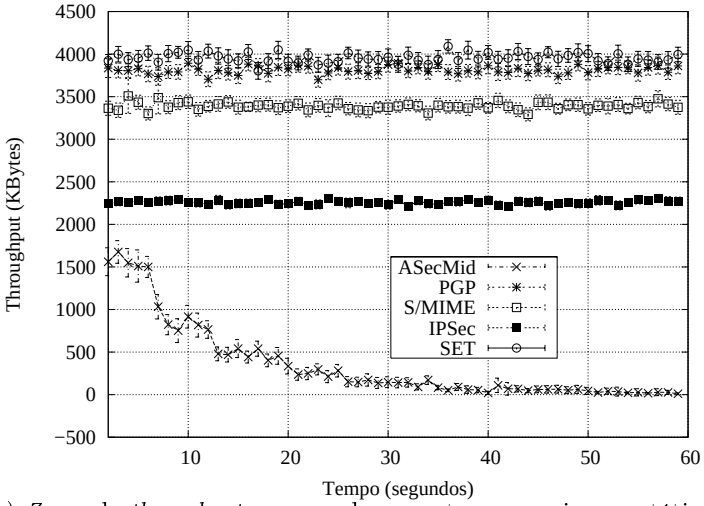


(d) Variação de *throughput* e força criptográfica do middleware

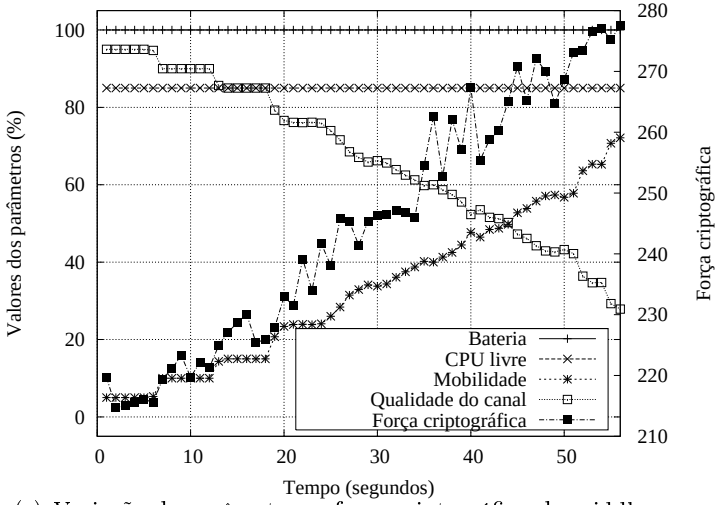
Figura 5.2: Execução com baixas restrições de QoS e segurança, e recursos se exaurindo com o tempo



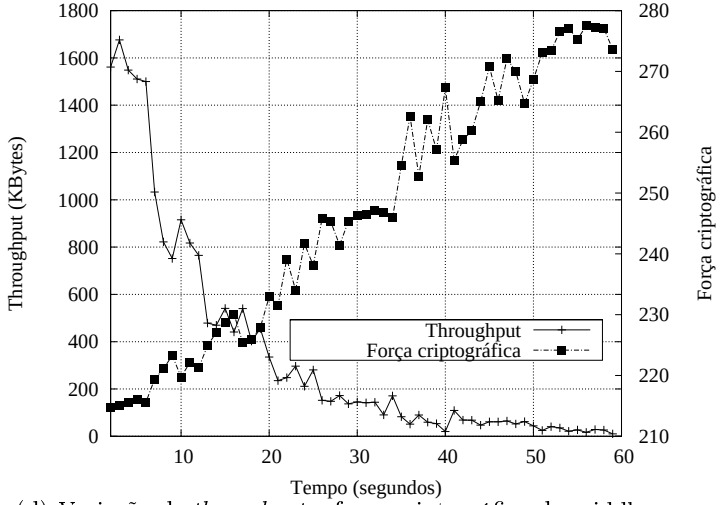
(a) *Throughput* comparado a mecanismos estáticos



(b) *Zoom* do *throughput* comparado somente a mecanismos estáticos

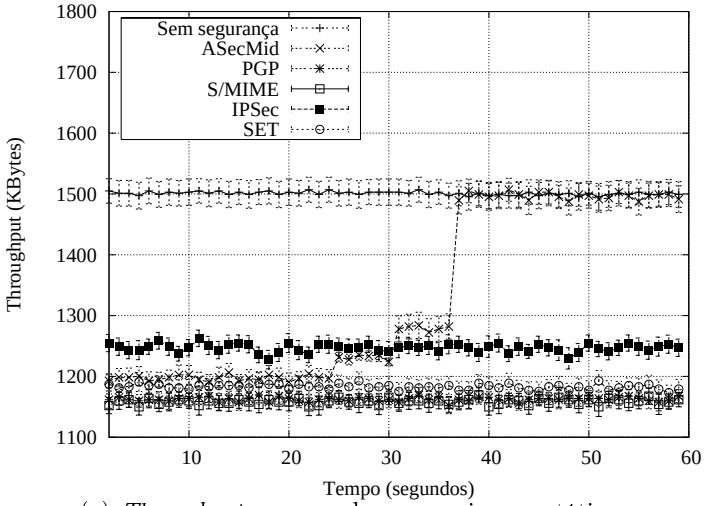


(c) Variação de parâmetros e força criptográfica do middleware

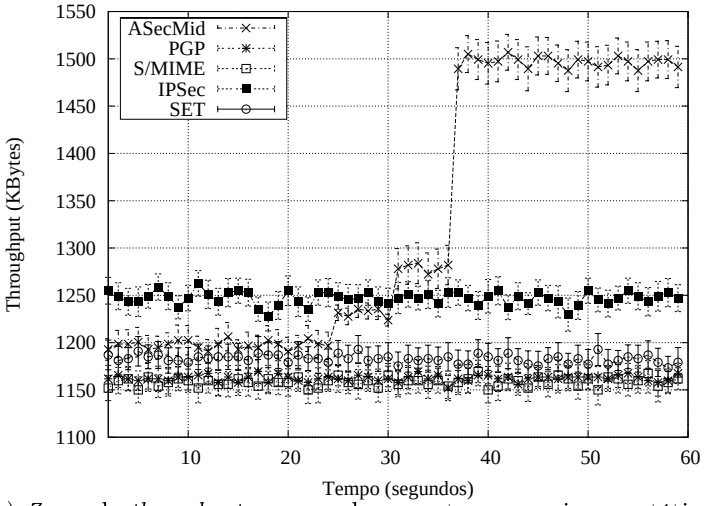


(d) Variação de *throughput* e força criptográfica do middleware

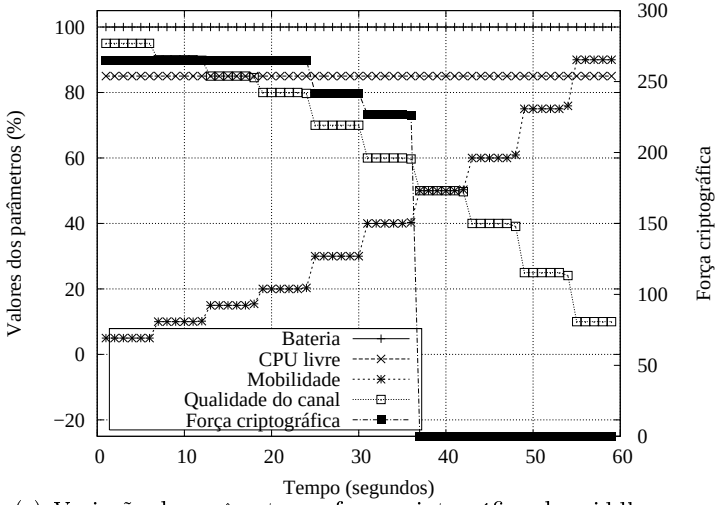
Figura 5.3: Execução com baixas restrições de QoS e segurança, e qualidade do meio sem fio piorando com o tempo



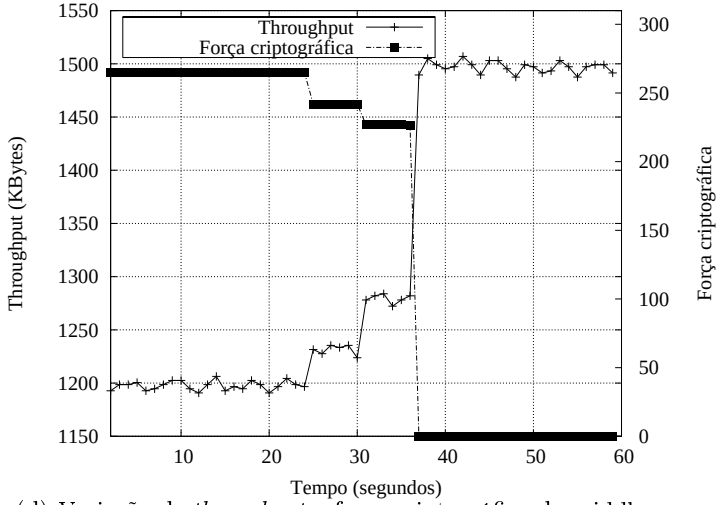
(a) *Throughput* comparado a mecanismos estáticos



(b) *Zoom* do *throughput* comparado somente a mecanismos estáticos

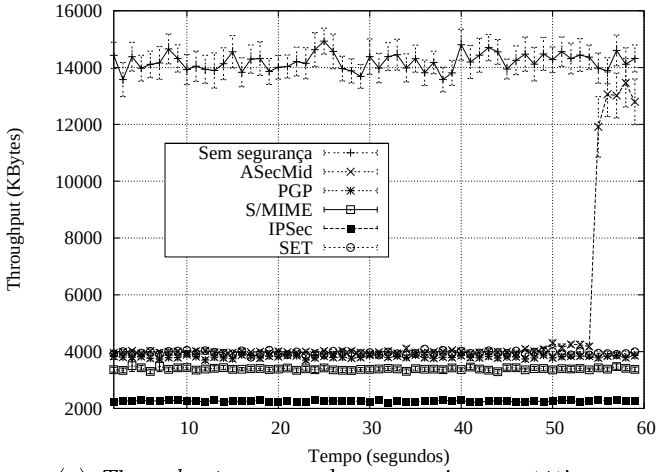


(c) Variação de parâmetros e força criptográfica do middleware

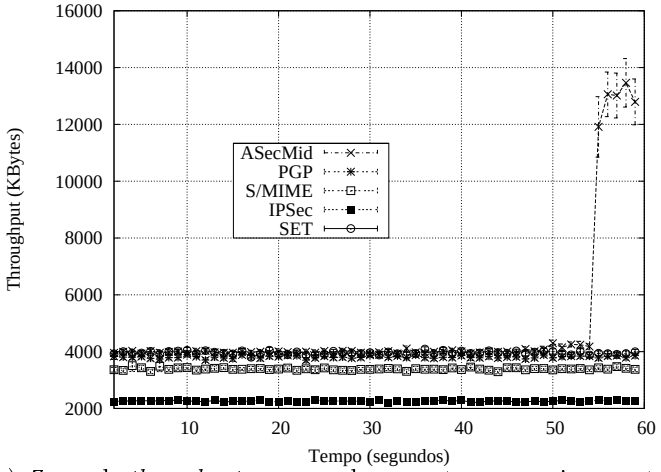


(d) Variação de *throughput* e força criptográfica do middleware

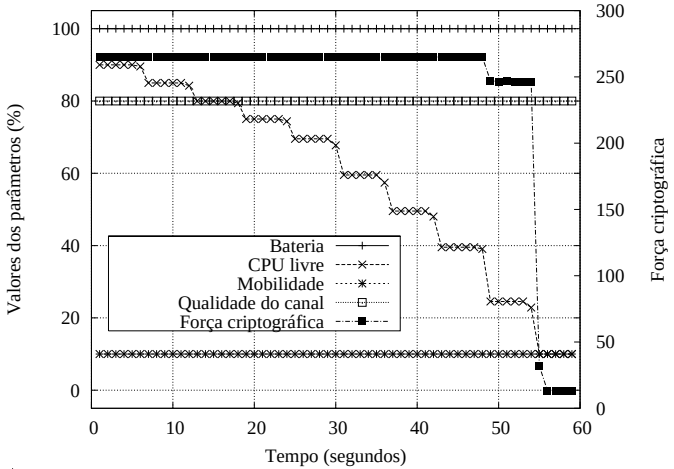
Figura 5.4: Execução com altas restrições de QoS e segurança, e qualidade do meio sem fio piorando com o tempo



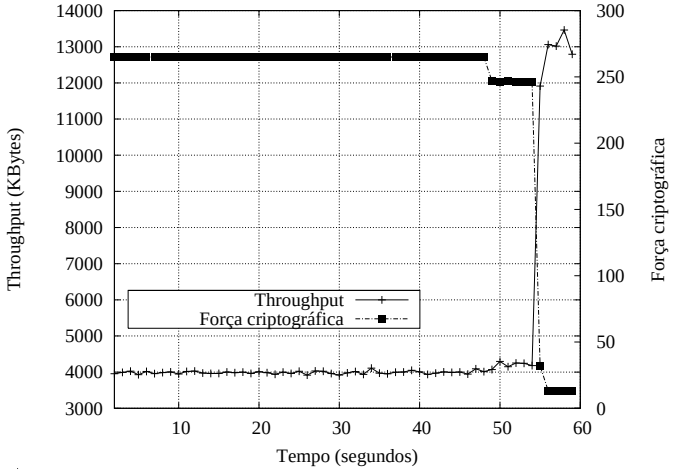
(a) *Throughput* comparado a mecanismos estáticos



(b) *Zoom* do *throughput* comparado somente a mecanismos estáticos

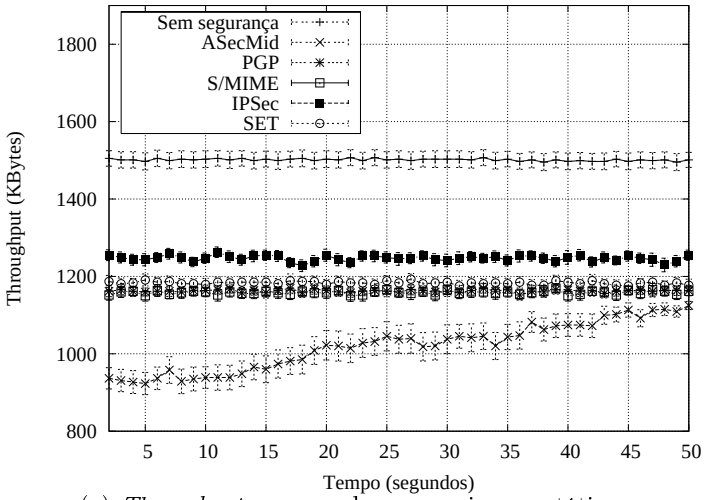


(c) Variação de parâmetros e força criptográfica do middleware

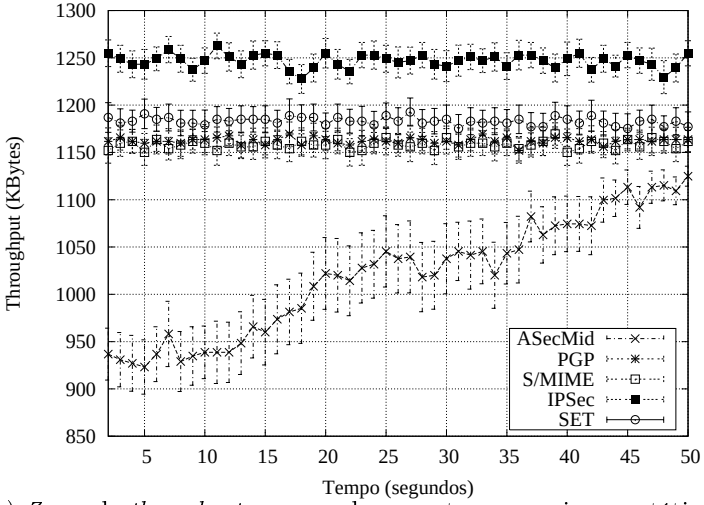


(d) Variação de *throughput* e força criptográfica do middleware

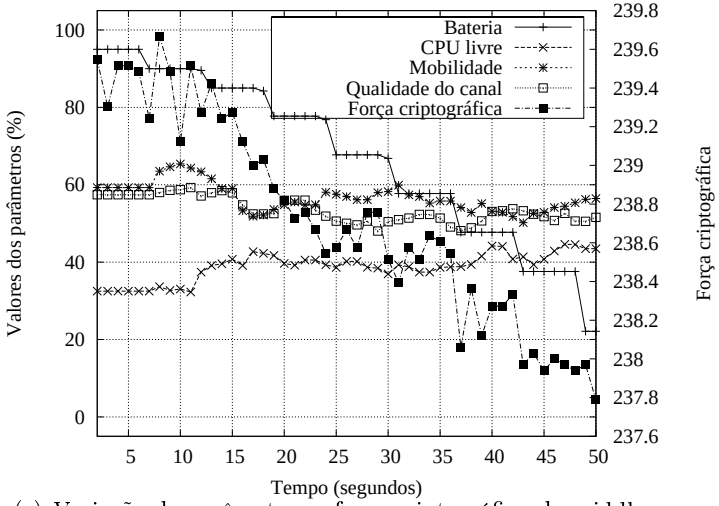
Figura 5.5: Execução com altas restrições de QoS e segurança, recursos se exaurindo com o tempo e exemplo de acontecimento de *deadlock* de requisitos



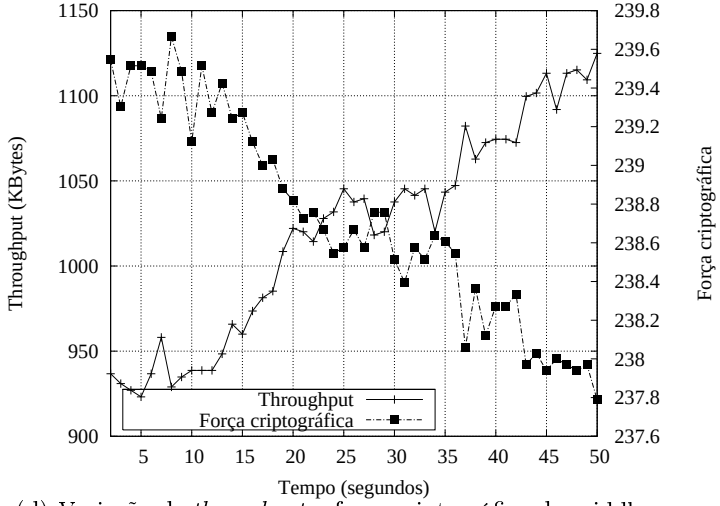
(a) *Throughput* comparado a mecanismos estáticos



(b) *Zoom* do *throughput* comparado somente a mecanismos estáticos

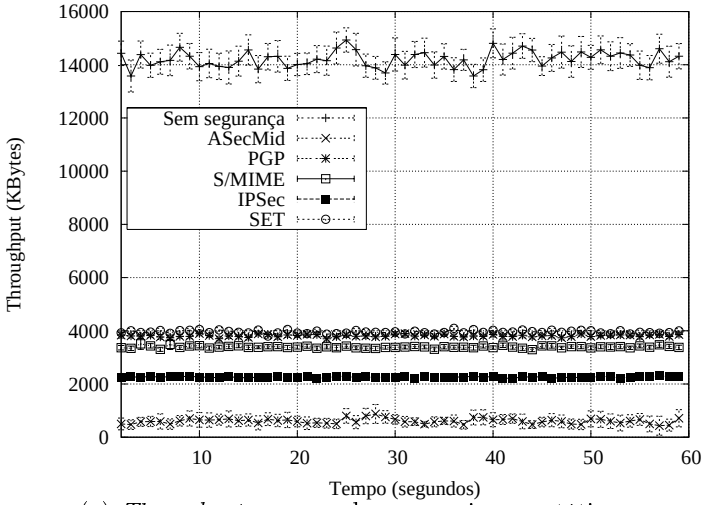


(c) Variação de parâmetros e força criptográfica do middleware

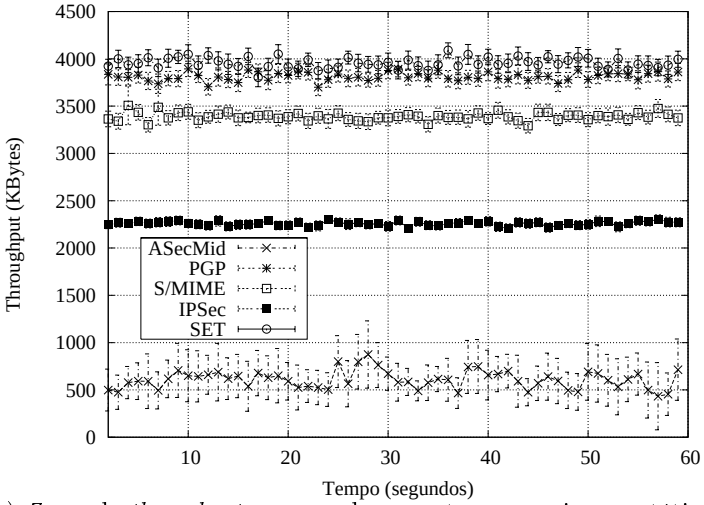


(d) Variação de *throughput* e força criptográfica do middleware

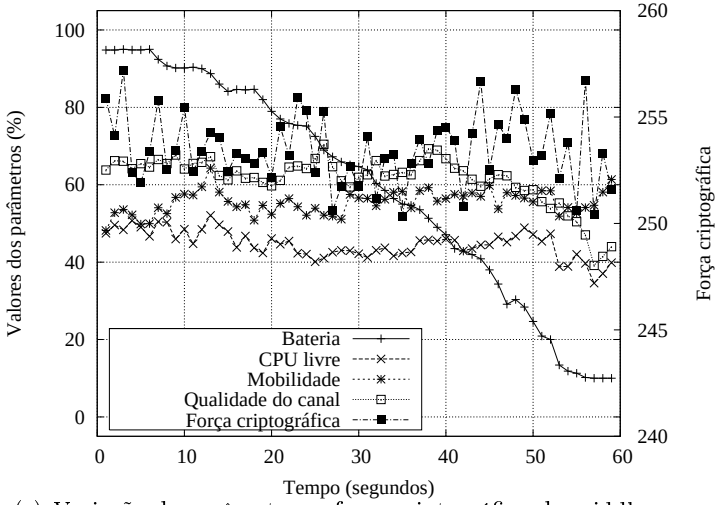
Figura 5.6: Execução com restrições de QoS e não de segurança, parâmetros aleatórios e bateria diminuindo com o tempo



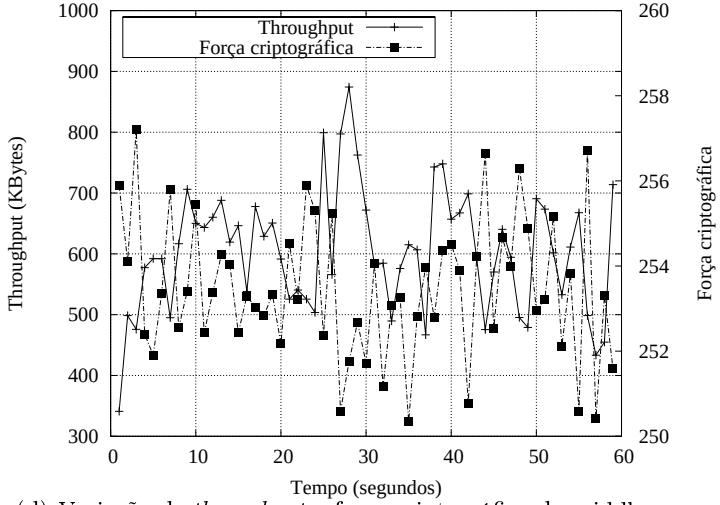
(a) *Throughput* comparado a mecanismos estáticos



(b) *Zoom* do *throughput* comparado somente a mecanismos estáticos

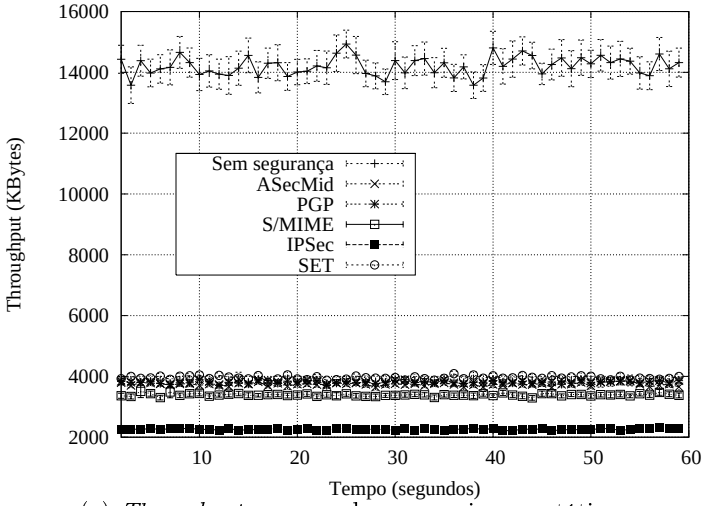


(c) Variação de parâmetros e força criptográfica do middleware

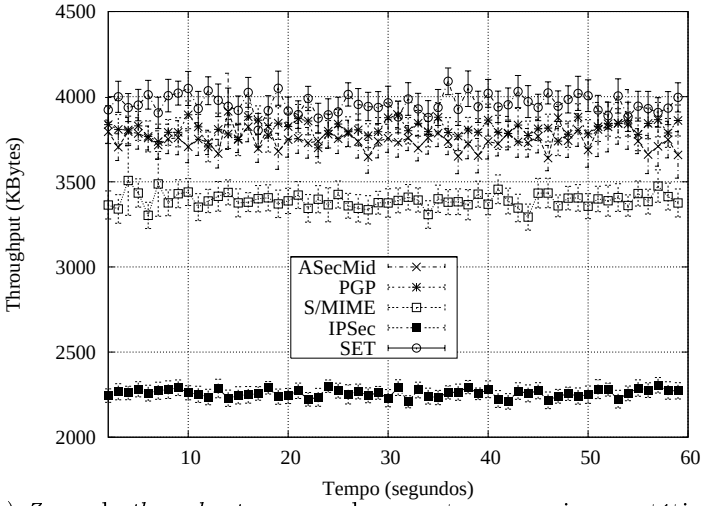


(d) Variação de *throughput* e força criptográfica do middleware

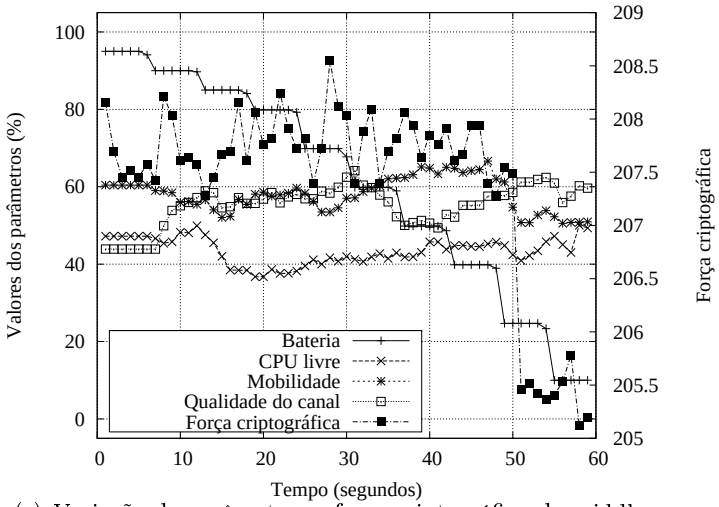
Figura 5.7: Execução com restrições de segurança e não de QoS, parâmetros aleatórios e bateria diminuindo com o tempo



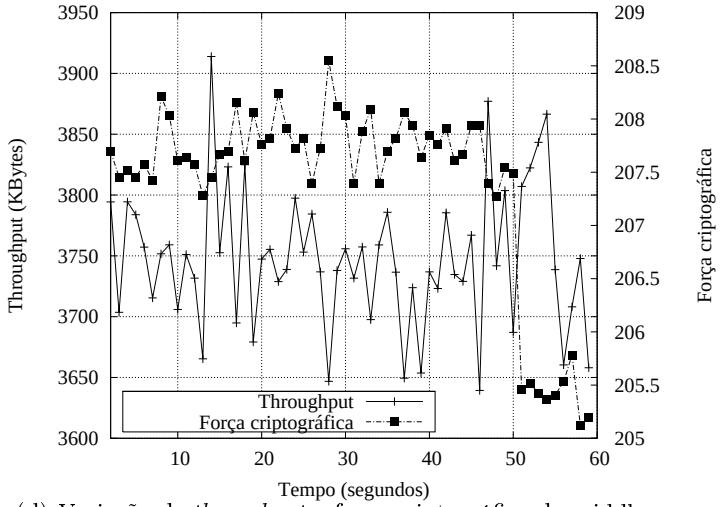
(a) *Throughput* comparado a mecanismos estáticos



(b) *Zoom* do *throughput* comparado somente a mecanismos estáticos

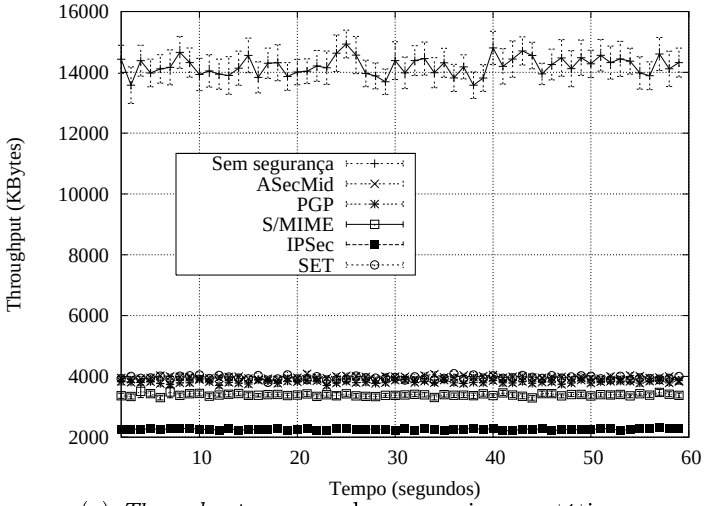


(c) Variação de parâmetros e força criptográfica do middleware

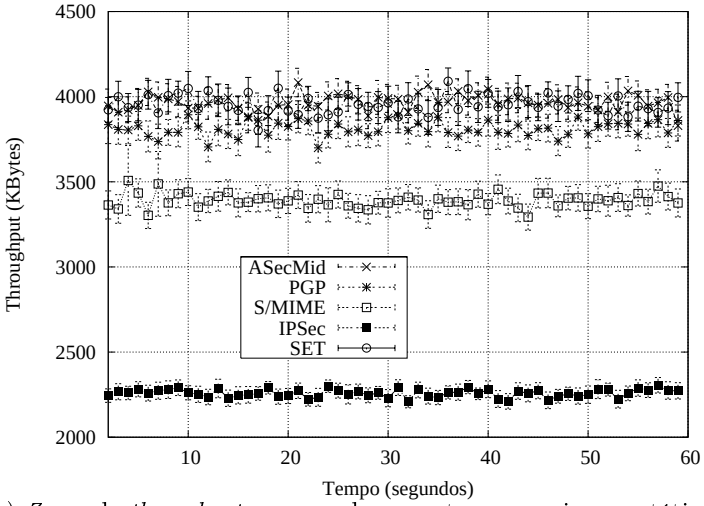


(d) Variação de *throughput* e força criptográfica do middleware

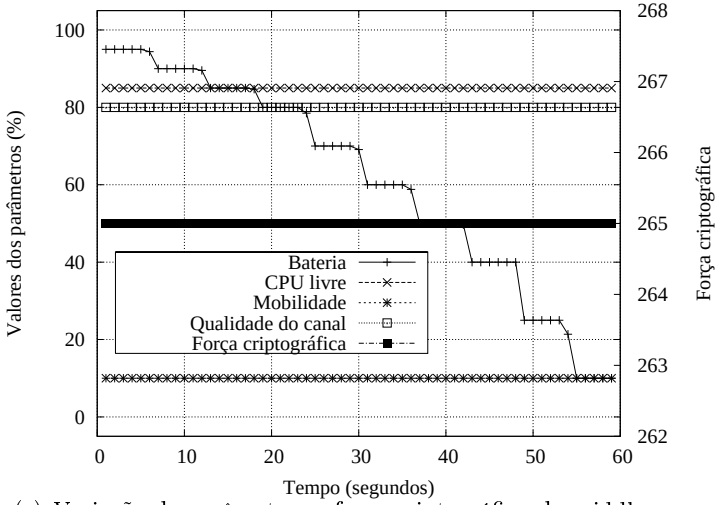
Figura 5.8: Execução com restrições medianas de QoS e segurança, parâmetros aleatórios e bateria diminuindo com o tempo



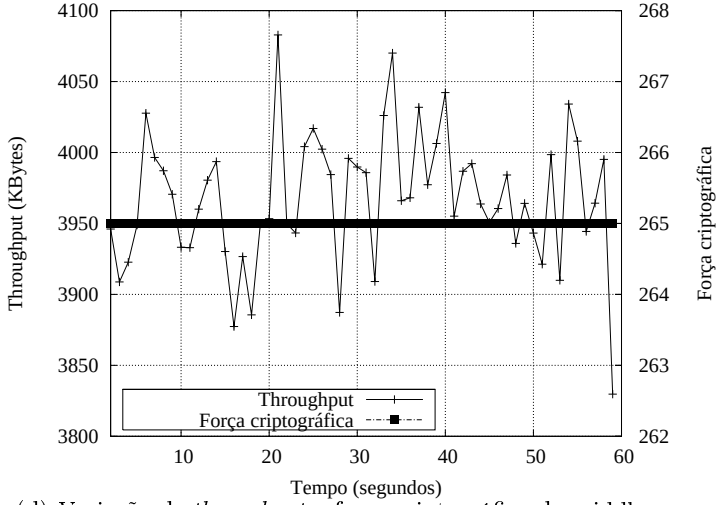
(a) *Throughput* comparado a mecanismos estáticos



(b) *Zoom* do *throughput* comparado somente a mecanismos estáticos



(c) Variação de parâmetros e força criptográfica do middleware



(d) Variação de *throughput* e força criptográfica do middleware

Figura 5.9: Execução com altas restrições de QoS e segurança e parâmetros fixos, com exceção da bateria

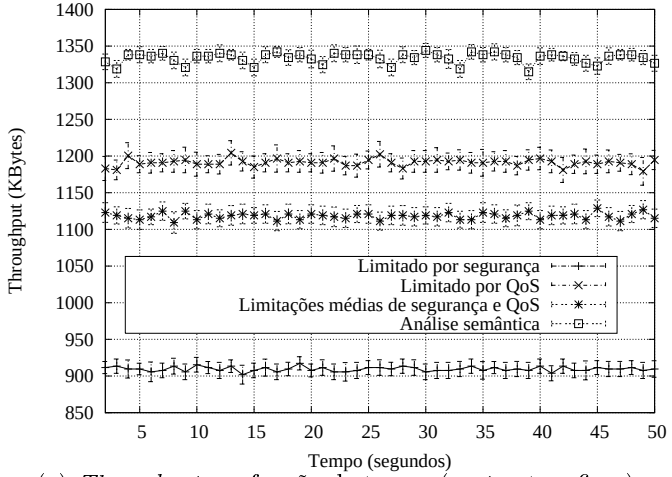
5.2 Análise semântica de dados

Nesta seção apresentamos experimentos com a extensão de análise semântica de dados, da camada de configuração de segurança. Em cada experimento comparamos quatro execuções do middleware: uma com a extensão de análise semântica e outras três sem a extensão, mas com diferentes configurações de restrições de QoS e segurança: restrição forte de segurança, forte de QoS e média em ambos. Cada experimento é apresentado com dois gráficos: o primeiro mostra a variação do *throughput*, enquanto que o segundo mostra a variação da força criptográfica, ambos em função do tempo.

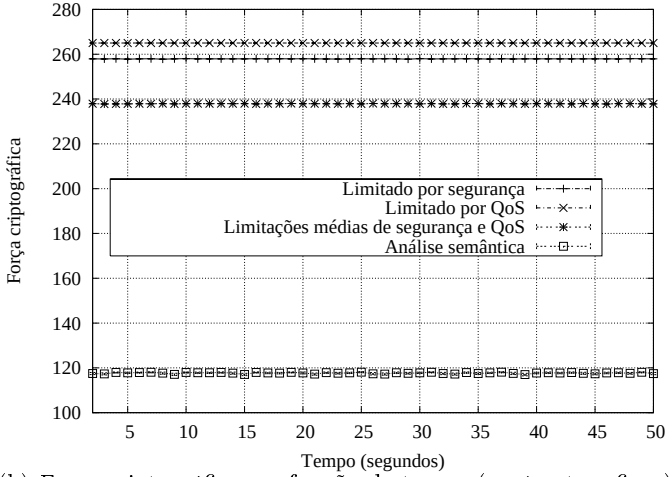
Nos experimentos da Figura 5.10 foi usada transmissão FTP. O arquivo de anotação descreve a transmissão classificando os pacotes de 64 KB como de alta importância de segurança, enquanto que os `ack` de 1 bytes são considerados sem importância. No experimento das Figuras 5.10(a) e 5.10(b) todos os parâmetros são fixos, enquanto que no das figuras 5.10(c) e 5.10(d) os parâmetros variam de forma pseudo-aleatória enquanto que a bateria decresce com o tempo. Como a versão com a extensão usa transmissão direta para os pacotes `ack`, ela aplica, na média, uma força criptográfica menor a cada pacote, obtendo maior desempenho, mas ainda sim aplicando o nível de segurança desejado nos pacotes importantes.

Nos experimentos da Figura 5.11 foi usado um padrão de tráfego que alterna entre pacotes importantes e não importantes, do ponto de vista criptográfico. As Figuras 5.11(a) e 5.11(b) mostram o experimento com parâmetros fixos, enquanto que as Figuras 5.11(c) e 5.11(d) mostram o experimento com parâmetros pseudo-aleatórios e bateria decrescente. É fácil observar como a extensão lida com a semântica dos dados, alternando entre uso de protocolos pesados e transmissão direta.

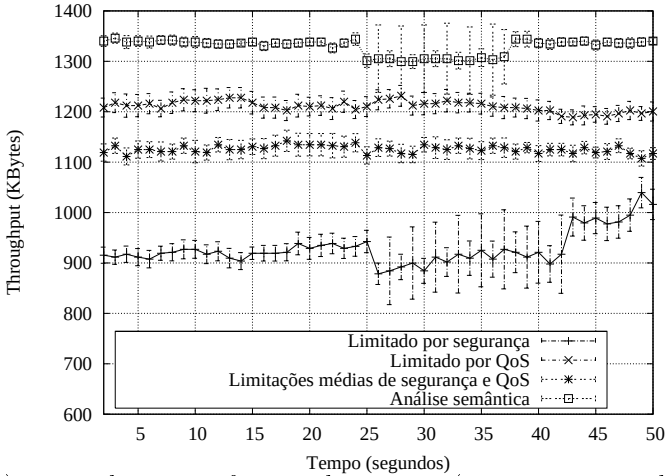
Finalmente, nos experimentos da Figura 5.12, reproduzimos o padrão de tráfego do formato MPEG de vídeo (Gall, 1991). Assim como nos anteriores, as Figuras 5.12(a) e 5.12(b) mostram o experimento com parâmetros fixos, e as Figuras 5.12(c) e 5.12(d) mostram o experimento com parâmetros pseudo-aleatórios e bateria decrescente. Nesses experimentos a anotação semântica identifica quadros MPEG dos tipos I, P e B, determinando níveis de segurança alto, médio e baixo para cada um deles, respectivamente. Mais uma vez é notável como a extensão proporciona maior desempenho sem comprometer a segurança oferecida, nos casos em que é possível determinar previamente a anotação com a semântica dos dados a serem transmitidos.



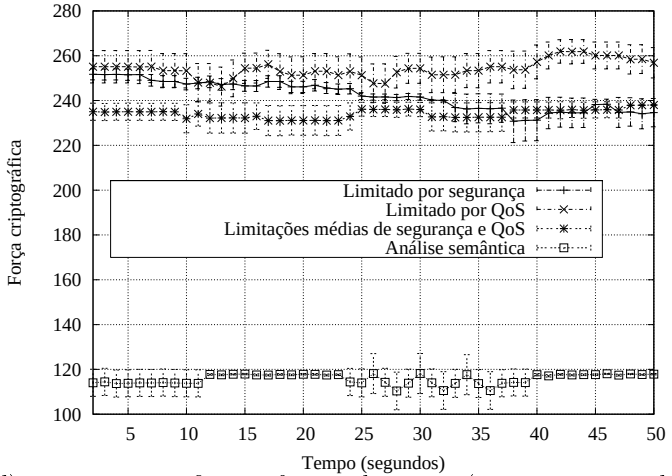
(a) *Throughput* em função do tempo (parâmetros fixos)



(b) Força criptográfica em função do tempo (parâmetros fixos)

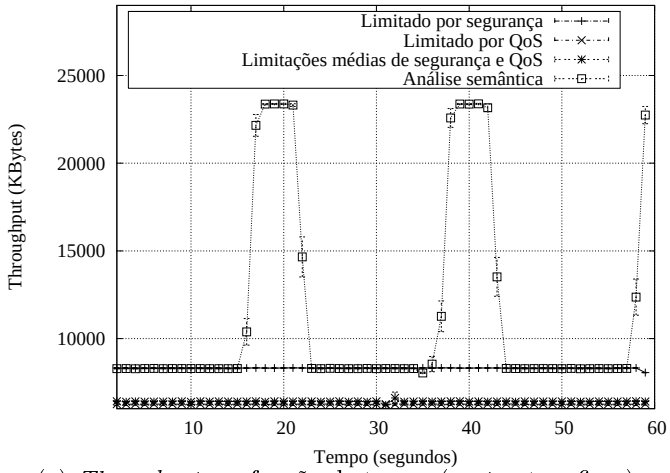


(c) *Throughput* em função do tempo (parâmetros pseudo-aleatórios)

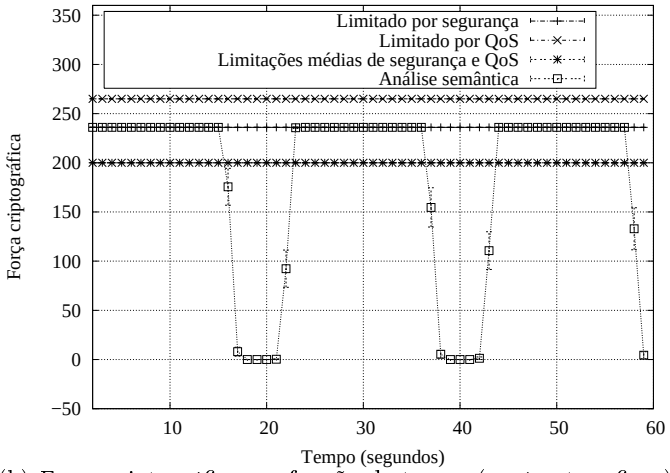


(d) Força criptográfica em função do tempo (parâmetros pseudo-aleatórios)

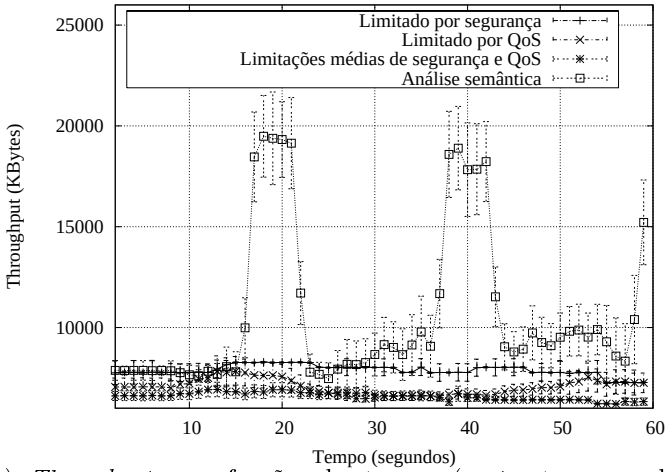
Figura 5.10: Execução de transmissão de dados FTP



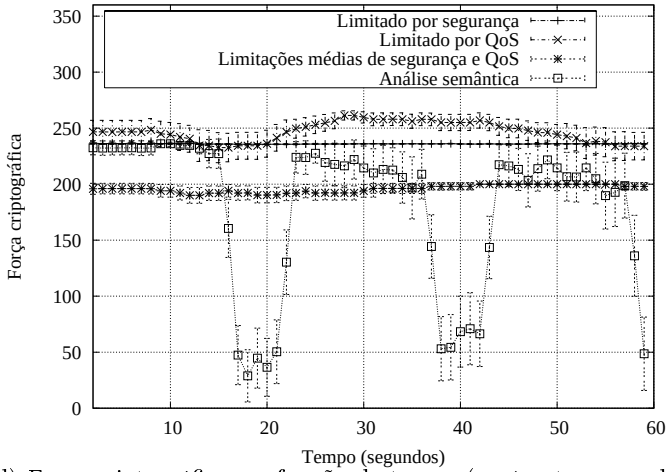
(a) *Throughput* em função do tempo (parâmetros fixos)



(b) Força criptográfica em função do tempo (parâmetros fixos)

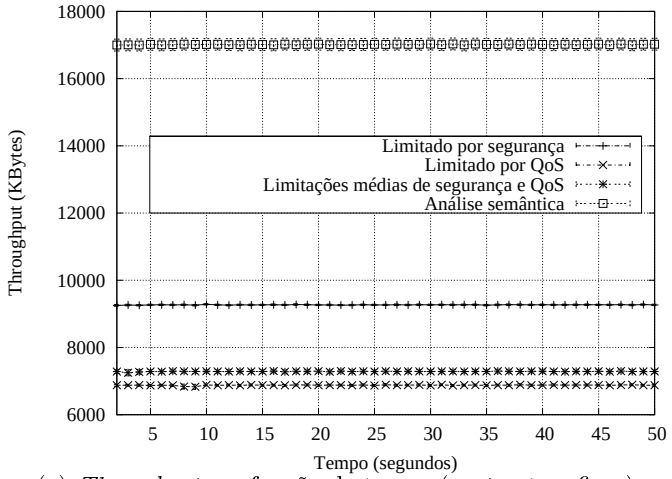


(c) *Throughput* em função do tempo (parâmetros pseudo-aleatórios)

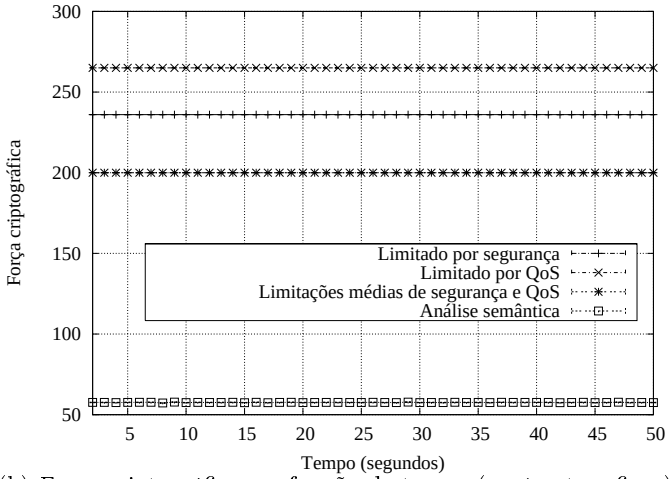


(d) Força criptográfica em função do tempo (parâmetros pseudo-aleatórios)

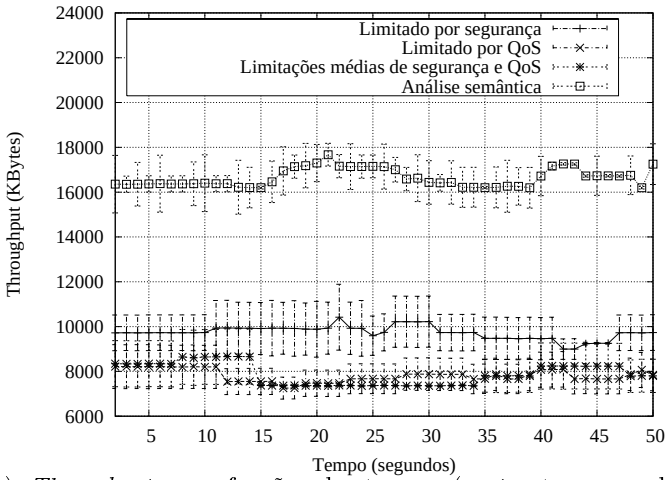
Figura 5.11: Execução de transmissão de dados que alternam entre mensagens com e sem importância de segurança



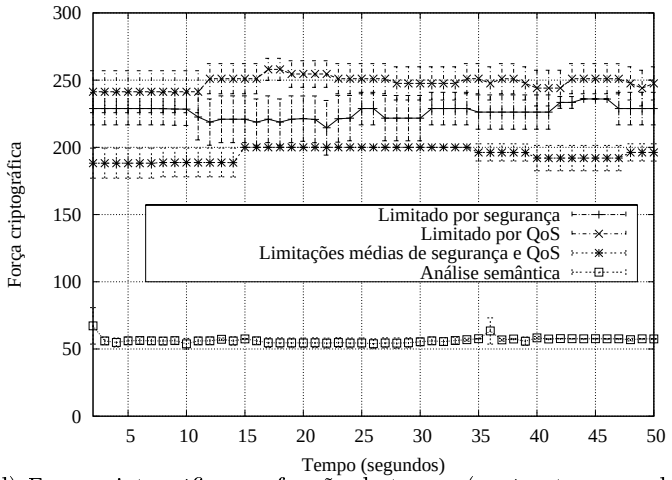
(a) *Throughput* em função do tempo (parâmetros fixos)



(b) Força criptográfica em função do tempo (parâmetros fixos)



(c) *Throughput* em função do tempo (parâmetros pseudo-aleatórios)



(d) Força criptográfica em função do tempo (parâmetros pseudo-aleatórios)

Figura 5.12: Execução de transmissão de dados MPEG

Capítulo 6

Considerações Finais

6.1 Conclusões

Neste trabalhamos apresentamos uma técnica para escolha adaptativa de protocolos de segurança, na forma de um middleware. Apresentamos ainda uma implementação de referência, da qual foram executados experimentos. Nossa implementação é robusta no sentido de poder ser configurada em relação as pesos das decisões tomadas. O middleware proposto é também configurável tanto pela aplicação que o utiliza (restrições de QoS e segurança), quanto pela entidade que mantém o sistema, através das configurações da biblioteca de algoritmos. Até onde sabemos, essa é uma abordagem inovadora para o problema.

Nossos resultados mostraram que a técnica proposta é adaptável a diversos contextos e traz benefícios em desempenho e facilidade, uma vez que a aplicação deixa de se preocupar com aplicações de algoritmos e protocolos de segurança. Os principais fatores que estimulam o uso do middleware são a presença da alta variabilidade de parâmetros de contexto, além de aplicações que permitem maior flexibilidade na escolha de protocolos.

Além disso, testamos nosso middleware com uma extensão que permite à aplicação determinar valores semânticos para cada trecho de dado transmitido. Com isso, o poder de adaptação aos parâmetros de contexto do middleware se soma à possibilidade de aplicar segurança de acordo com o real valor de cada dado. Dessa forma, o middleware consegue um ganho de desempenho considerável quando é possível descrever o formato de dados a ser transmitido.

O middleware proposto nesse trabalho apresenta uma nova abordagem para um problema de segurança cada vez maior na área de computação móvel e sem fio. Além disso, a solução proposta é aplicável para uma grande gama de dispositivos de hardware.

6.2 Trabalhos Futuros

Um dos trabalhos futuros imediatos seria o aprofundamento do estudo dos parâmetros. Novos parâmetros podem ser considerados, como por exemplo a taxa efetiva de transmissão, com o middleware alternando protocolos na tentativa de mantê-la dentro de um determinado nível.

A determinação das constantes n e K pode ser também melhor estudada, com geração de técnicas para determinação de escolhas de valores apropriados para contextos específicos.

Do ponto de vista arquitetural, identificamos alguns pontos a serem explorados. Um é a transferência do middleware para camadas mais baixas na pilha de protocolos, derivando assim uma técnica para chaveamento de protocolos em baixo nível. Outro é a utilização de técnicas de reflexão para que o middleware identifique e aplique novos algoritmos de segurança sem que mudanças no código-fonte sejam necessárias. Por fim, o processo de escolha de protocolos pode ser feito de forma bidirecional, possivelmente adotando um canal de comunicação separado somente para negociação em tempo-real.

Este trabalho também abre espaço para ser utilizado como base de novas abordagens de segurança. Por exemplo, o mecanismo base de chaveamento de protocolos pode ser modificado para suportar um ambiente com múltiplos *hosts* em que ocorre roteamento. Atualmente, o middleware escolhe e aplica protocolos em cada conexão par-a-par. Pode ser feito um esquema para roteamento inteligente de pacotes seguros, havendo re-codificação do pacote somente em áreas da rede cujo contexto é diferente. A Figura 6.1 ilustra essa possibilidade.

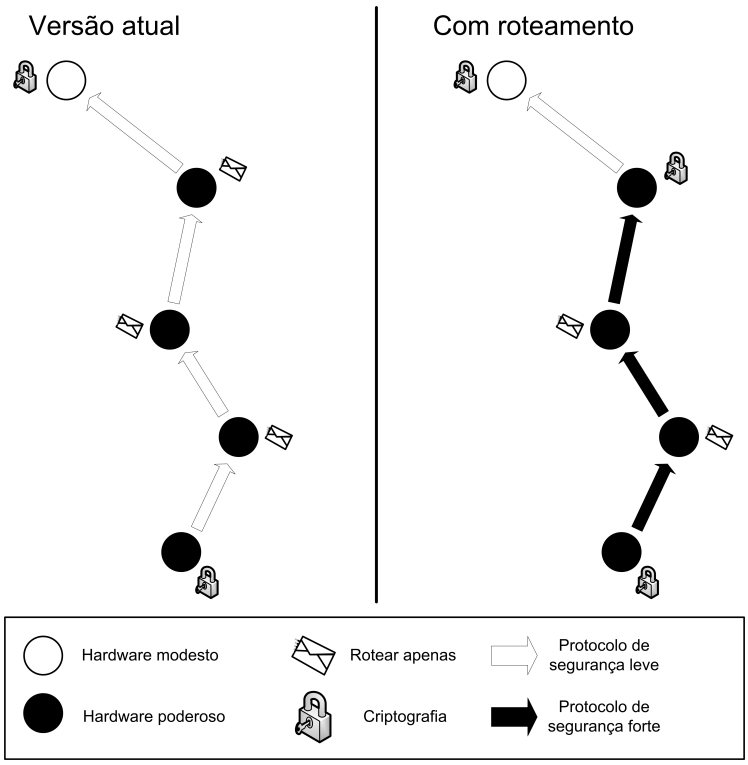


Figura 6.1: Possibilidade de extensão do middleware para tratar roteamento

O trabalho pode ainda ser aplicado em segurança em grupos. Um esquema para comunicação com elementos distantes de uma rede pode ser feito, mantendo a negociação de protocolos somente entre vizinhos, ou então mantendo uma única negociação para um grupo confiável. Questões sobre grupos móveis, aceitação de novos integrantes, confiança e interação entre diferentes grupos também são interessantes.

Apêndice A

Detalhamento de Parâmetros e Protocolos dos Experimentos

A.1 ASecMid comparado a mecanismos estáticos

Parâmetro	**Valor**
Capacidade de memória	8 MB
Bateria	Decrescente, 100 a 0%
Memória livre	Entre 10 KB e 8 MB
Uso de CPU	Entre 0 e 100%
Rádio	WiFi
Roteamento	Não
Mobilidade	Entre 0 e 100%
Qualidade do canal	Entre 0 e 100%
Latência	Entre 0 e 10 ms
Confidencialidade	Mandatório
Integridade	Mandatório
Autenticação	Mandatório
Máx. *overhead*	64 bytes
Máx. latência	11 ms
Máx. memória	20 KB

Tabela A.1: Variação de parâmetros no experimento da Figura 5.1

Protocolo	**Num. médio de usos**		**Força**
	servidor	cliente	
`des cfb, md2, dsa`	2678,09	3441,27	180
`aes ofb 256, cmac aes 256, dsa`	2566,94	2737,15	265
`aes cfb 256, cmac aes 256, dsa`	175,82	210	265
`aes ofb 128, md4, dsa`	161,45	145,51	220
`rc4 128, hmac md4, dsa`	108,24	112,33	216
`rc4 256, md4, dsa`	10,15	4,97	210
`aes cfb 192, cmac aes 256, dsa`	8,51	9,79	260
`rc4 128, md4, dsa`	0,12	0,06	200

Tabela A.2: Protocolos usados no experimento da Figura 5.1

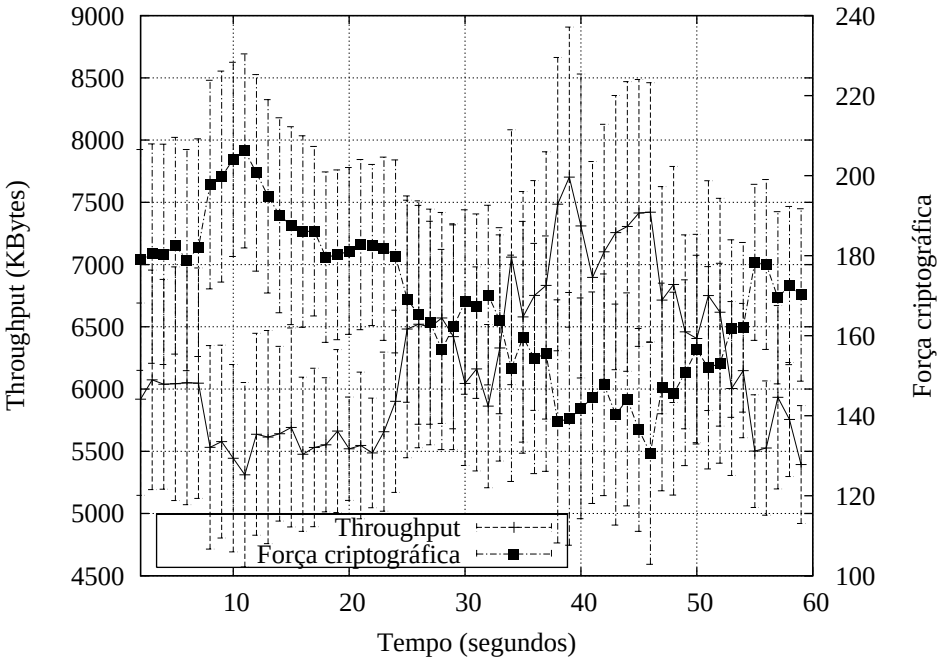


Figura A.1: Variação de força criptográfica e *throughput* do middleware, com intervalos de confiança de 90%, para o experimento da Figura 5.1(d)

Parâmetro	**Valor**
Capacidade de memória	512 KB
Bateria	100%
Memória livre	Decrescente, de 100 a 10 KB
Uso de CPU	Crescente, de 5 a 100%
Rádio	WiFi
Roteamento	Não
Mobilidade	10%
Qualidade do canal	80%
Latência	1 ms
Confidencialidade	Opcional
Integridade	Opcional
Autenticação	Opcional
Máx. *overhead*	-
Máx. latência	-
Máx. memória	-

Tabela A.3: Variação de parâmetros no experimento da Figura 5.2

Protocolo	**Num. médio de usos**		**Força**
	servidor	cliente	
`des ecb, sha1, rsa msg 1024`	1833,67	1605,52	200
`hmac md4`	427,12	266,24	76
`rsa 1024, cmac aes 256, dsa`	338,36	300,79	277
`rsa 2048, sha1, rsa msg 1024`	60,88	37,61	246

Tabela A.4: Protocolos usados no experimento da Figura 5.2

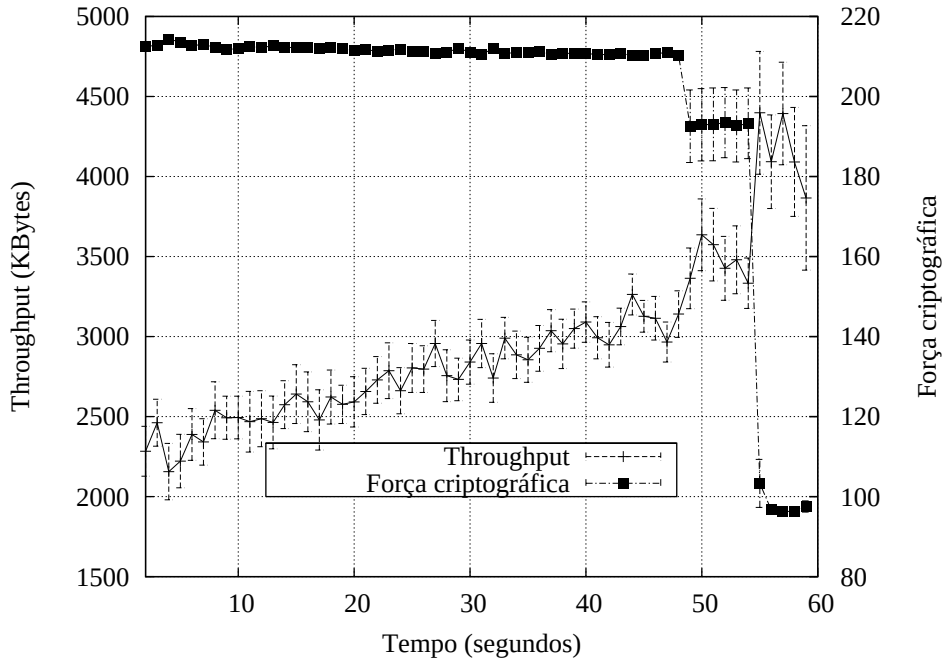


Figura A.2: Variação de força criptográfica e *throughput* do middleware, com intervalos de confiança de 90%, para o experimento da Figura 5.2(d)

Parâmetro	**Valor**
Capacidade de memória	512 MB
Bateria	100%
Memória livre	100 MB
Uso de CPU	15%
Rádio	WiFi
Roteamento	Não
Mobilidade	Crescente, de 0 a 100%
Qualidade do canal	Decrescente, de 100 a 0%
Latência	Crescente, de 1 a 10 ms
Confidencialidade	Opcional
Integridade	Opcional
Autenticação	Opcional
Máx. *overhead*	-
Máx. latência	-
Máx. memória	-

Tabela A.5: Variação de parâmetros no experimento da Figura 5.3

Protocolo	**Num. médio de usos**		**Força**
	servidor	cliente	
`des ecb, sha1, dsa`	241,42	230,82	200
`rsa 1024, cmac aes 256, dsa`	95,03	87,39	277
`rsa 2048, cmac aes 256, dsa`	14,64	13,58	281

Tabela A.6: Protocolos usados no experimento da Figura 5.3

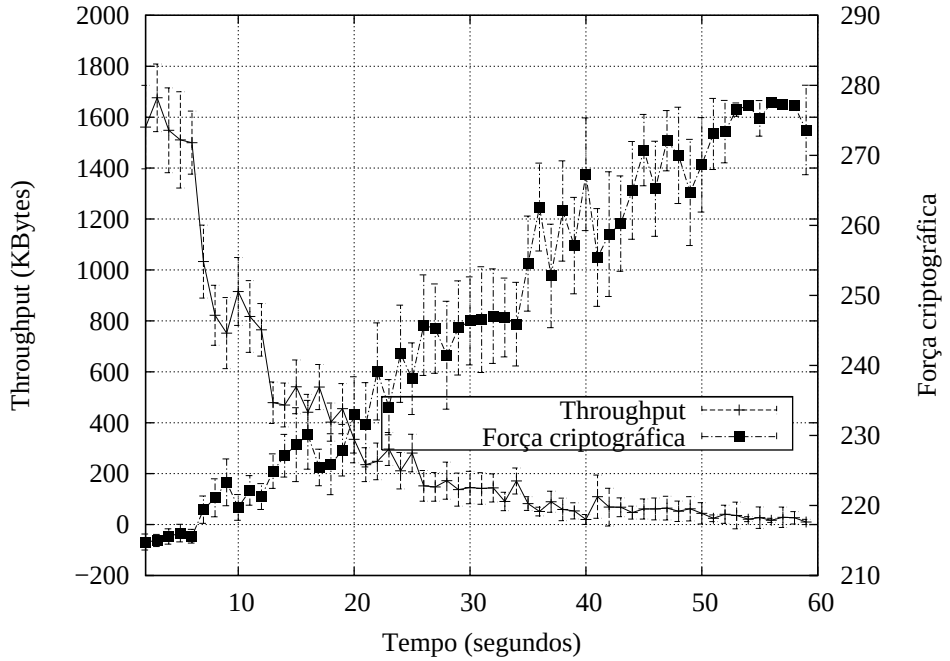


Figura A.3: Variação de força criptográfica e *throughput* do middleware, com intervalos de confiança de 90%, para o experimento da Figura 5.3(d)

Parâmetro	**Valor**
Capacidade de memória	512 MB
Bateria	100%
Memória livre	100 MB
Uso de CPU	15%
Rádio	WiFi
Roteamento	Não
Mobilidade	Crescente, de 0 a 100%
Qualidade do canal	Decrescente, de 100 a 0%
Latência	Crescente, de 1 a 10 ms
Confidencialidade	Mandatório
Integridade	Mandatório
Autenticação	Mandatório
Máx. *overhead*	64 bytes
Máx. latência	11 ms
Máx. memória	20 KB

Tabela A.7: Variação de parâmetros no experimento da Figura 5.4

Protocolo	**Num. médio de usos**		**Força**
	servidor	cliente	
`aes ofb 256, cmac aes 256, dsa`	441,88	306,91	265
`rc4 128, hmac md4, dsa`	103,94	0	216
`aes ofb 128, md4, dsa`	115,45	0	220
`aes cfb 256, cmac aes 256, dsa`	0	370,28	265
deadlock	575,30	559,24	0

Tabela A.8: Protocolos usados no experimento da Figura 5.4

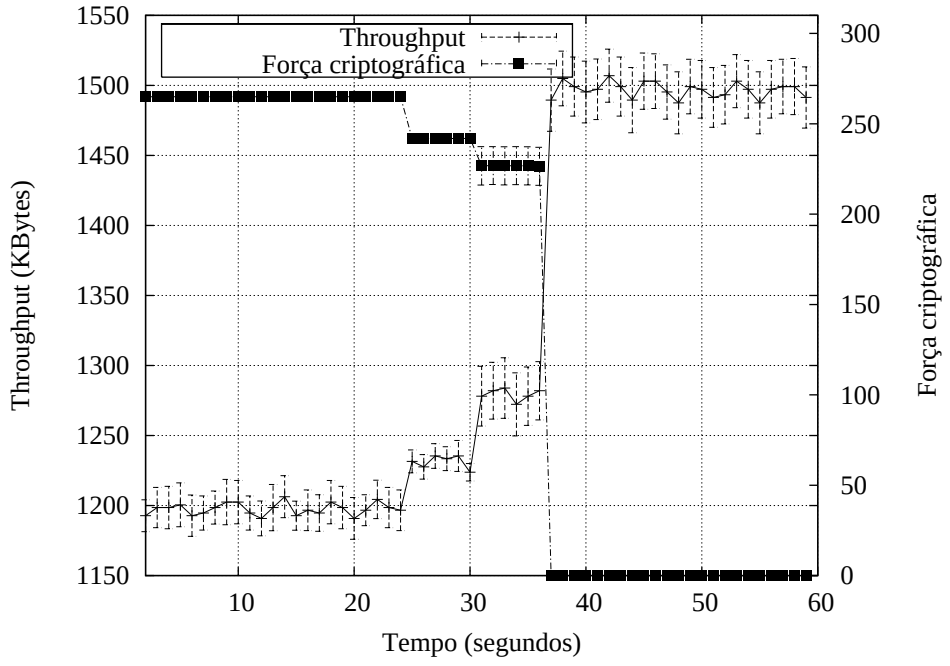


Figura A.4: Variação de força criptográfica e *throughput* do middleware, com intervalos de confiança de 90%, para o experimento da Figura 5.4(d)

Parâmetro	**Valor**
Capacidade de memória	512 KB
Bateria	100%
Memória livre	Decrescente, de 100 a 10 KB
Uso de CPU	Crescente, de 5 a 100%
Rádio	WiFi
Roteamento	Não
Mobilidade	10%
Qualidade do canal	80%
Latência	1 ms
Confidencialidade	Mandatório
Integridade	Mandatório
Autenticação	Mandatório
Máx. *overhead*	64 bytes
Máx. latência	11 ms
Máx. memória	20 KB

Tabela A.9: Variação de parâmetros no experimento da Figura 5.5

Protocolo	**Num. médio de usos**		**Força**
	servidor	cliente	
`aes ofb 256, cmac aes 256, dsa`	3198,97	3344,28	265
`aes ofb 128, md4, dsa`	262,19	0	220
`aes cfb 256, cmac aes 256, dsa`	202,64	195,64	265
`aes cfb 256, md2, dsa`	89,03	0	220
`rc4 256, md4, dsa`	0	49,55	210
deadlock	1326,67	1221,94	0

Tabela A.10: Protocolos usados no experimento da Figura 5.5

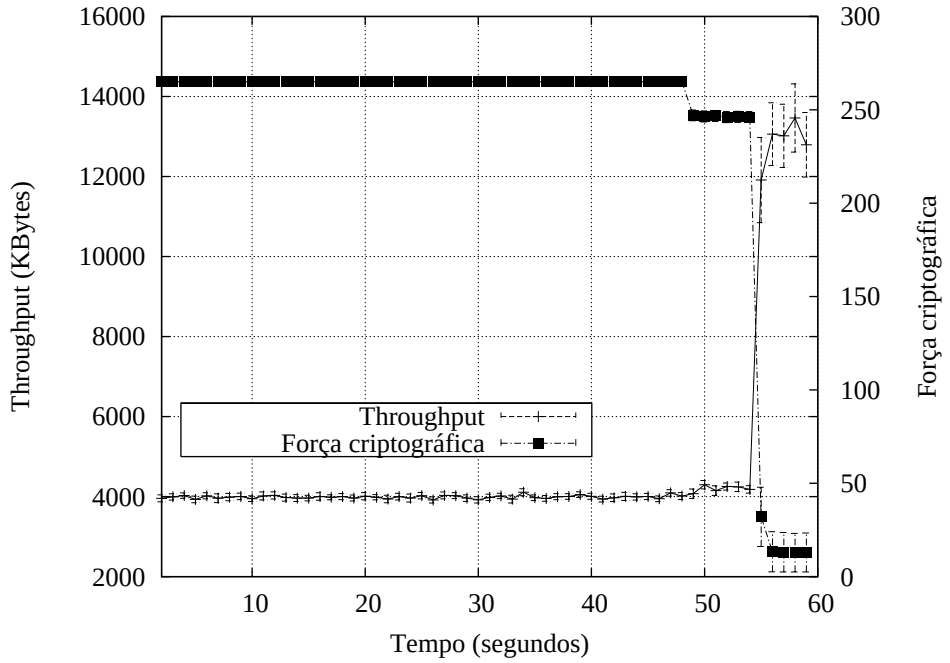


Figura A.5: Variação de força criptográfica e *throughput* do middleware, com intervalos de confiança de 90%, para o experimento da Figura 5.5(d)

Parâmetro	**Valor**
Capacidade de memória	8 MB
Bateria	Decrescente, 100 a 0%
Memória livre	Entre 10 KB e 8 MB
Uso de CPU	Entre 0 e 100%
Rádio	WiFi
Roteamento	Não
Mobilidade	Entre 0 e 100%
Qualidade do canal	Entre 0 e 100%
Latência	Entre 0 e 10 ms
Confidencialidade	Opcional
Integridade	Opcional
Autenticação	Opcional
Máx. *overhead*	512 bytes
Máx. latência	100 ms
Máx. memória	50 KB

Tabela A.11: Variação de parâmetros no experimento da Figura 5.6

Protocolo	**Num. médio de usos**		**Força**
	servidor	cliente	
`des ecb, sha1, dsa`	901,64	0	200
`rsa 1024, cmac aes 256, dsa`	0	671,79	277
`rsa 2048, cmac aes 256, dsa`	0	284,39	281

Tabela A.12: Protocolos usados no experimento da Figura 5.6

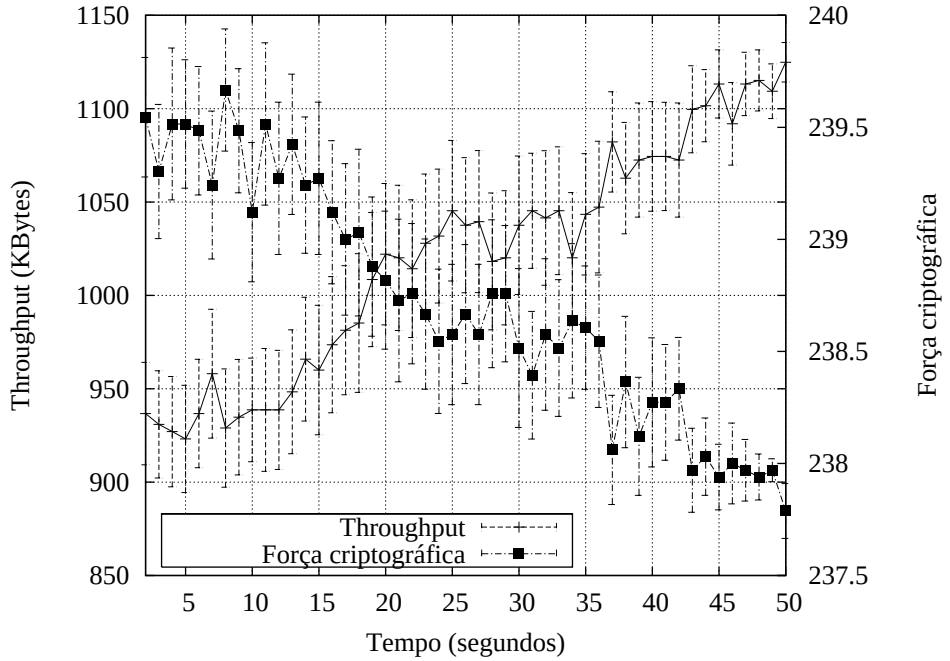


Figura A.6: Variação de força criptográfica e *throughput* do middleware, com intervalos de confiança de 90%, para o experimento da Figura 5.6(d)

Parâmetro	**Valor**
Capacidade de memória	8 MB
Bateria	Decrescente, 100 a 0%
Memória livre	Entre 10 KB e 8 MB
Uso de CPU	Entre 0 e 100%
Rádio	WiFi
Roteamento	Não
Mobilidade	Entre 0 e 100%
Qualidade do canal	Entre 0 e 100%
Latência	Entre 0 e 10 ms
Confidencialidade	Crítico
Integridade	Crítico
Autenticação	Crítico
Máx. *overhead*	-
Máx. latência	-
Máx. memória	-

Tabela A.13: Variação de parâmetros no experimento da Figura 5.7

Protocolo	Num. médio de usos		Força
	servidor	cliente	
`aes ofb 192, md5, dsa`	411,61	383,97	240
`rsa 1024, cmac aes 256, dsa`	91,88	81,15	277
`rsa 2048, cmac aes 256, dsa`	47,85	42,52	281
`rc4 128, hmac md4, dsa`	0	0,79	216
`rc4 128, hmac md5, dsa`	0	0,64	220
`rsa 1024, sha1, rsa msg 1024`	0	0,33	242
`rsa 2048, sha1, rsa msg 1024`	0	0,03	246

Tabela A.14: Protocolos usados no experimento da Figura 5.7

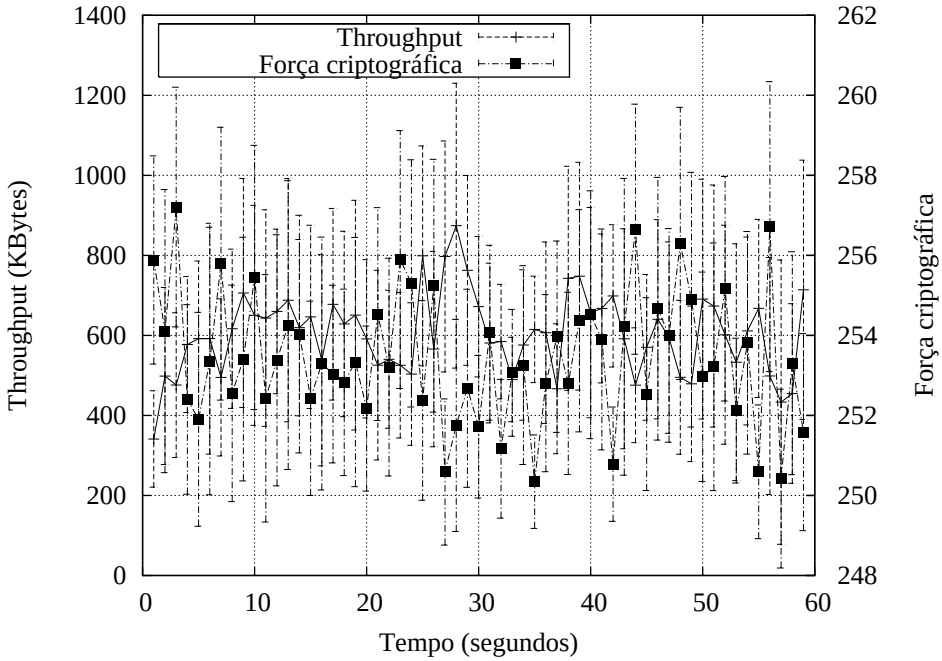


Figura A.7: Variação de força criptográfica e *throughput* do middleware, com intervalos de confiança de 90%, para o experimento da Figura 5.7(d)

Parâmetro	Valor
Capacidade de memória	8 MB
Bateria	Decrescente, 100 a 0%
Memória livre	Entre 10 KB e 8 MB
Uso de CPU	Entre 0 e 100%
Rádio	WiFi
Roteamento	Não
Mobilidade	Entre 0 e 100%
Qualidade do canal	Entre 0 e 100%
Latência	Entre 0 e 10 ms
Confidencialidade	Desejável
Integridade	Desejável
Autenticação	Desejável
Máx. *overhead*	1024 bytes
Máx. latência	20 ms
Máx. memória	50 KB

Tabela A.15: Variação de parâmetros no experimento da Figura 5.8

Protocolo	Num. médio de usos		Força
	servidor	cliente	
`des ecb, sha1, dsa`	3142,79	3455,30	200
`rsa 1024, cmac aes 256, dsa`	408,82	450,36	277
`des ecb, cmac aes 256, rsa msg 1024`	5,15	6,12	215
`rsa 1024, cmac aes 128, dsa`	0,03	0	272

Tabela A.16: Protocolos usados no experimento da Figura 5.8

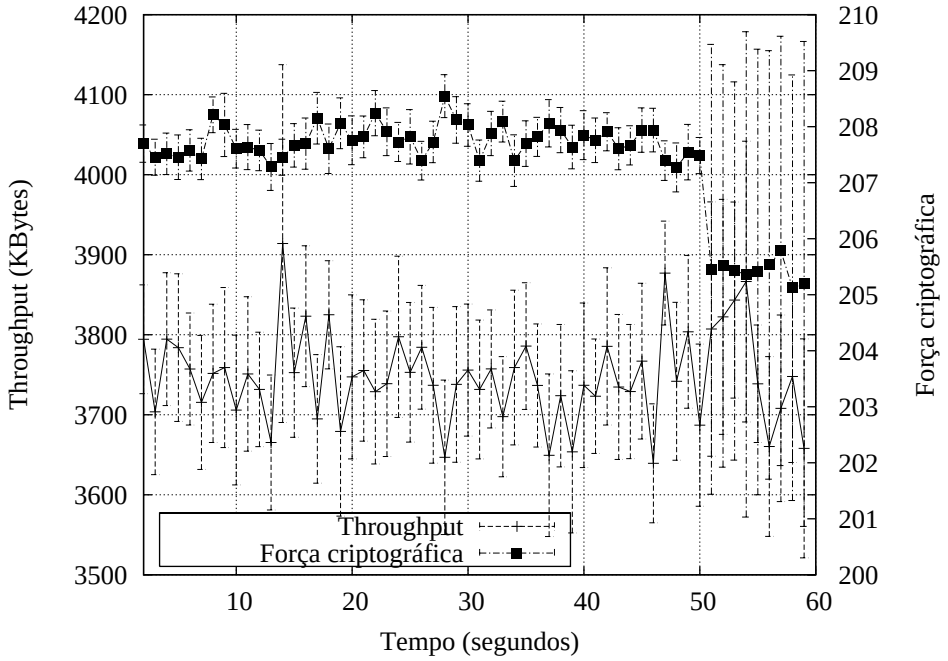


Figura A.8: Variação de força criptográfica e *throughput* do middleware, com intervalos de confiança de 90%, para o experimento da Figura 5.8(d)

Parâmetro	**Valor**
Capacidade de memória	512 KB
Bateria	Decrescente, 100 a 0%
Memória livre	100 KB
Uso de CPU	15%
Rádio	WiFi
Roteamento	Não
Mobilidade	10%
Qualidade do canal	80%
Latência	1 ms
Confidencialidade	Mandatório
Integridade	Mandatório
Autenticação	Mandatório
Máx. *overhead*	64 bytes
Máx. latência	11 ms
Máx. memória	20 KB

Tabela A.17: Variação de parâmetros no experimento da Figura 5.9

Protocolo	**Num. médio de usos**		**Força**
	servidor	cliente	
`aes ofb 256, cmac aes 256, dsa`	3562,56	3333,61	265
`aes cfb 256, cmac aes 256, dsa`	312,06	286,28	265

Tabela A.18: Protocolos usados no experimento da Figura 5.9

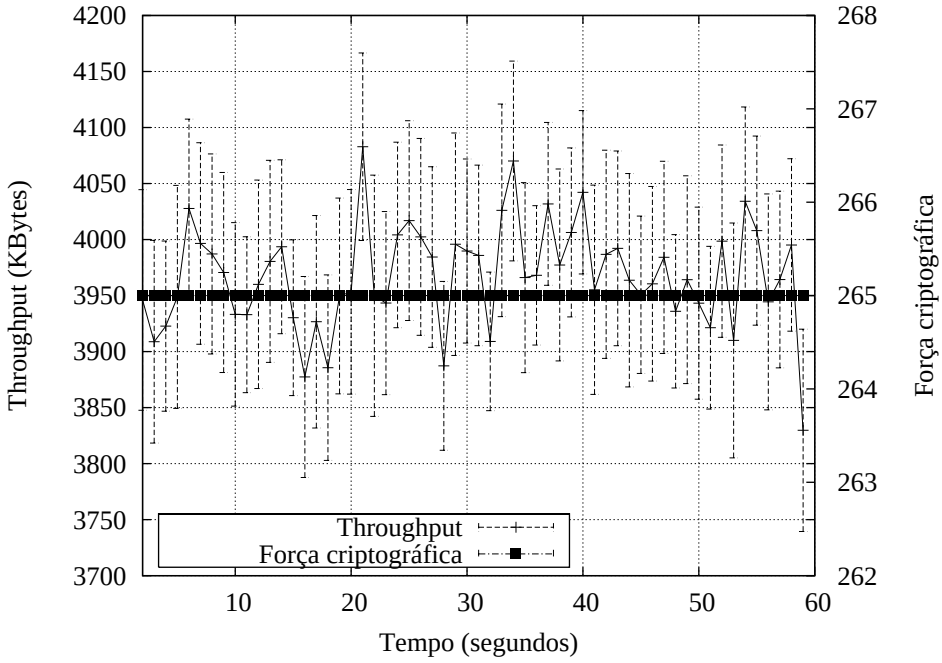


Figura A.9: Variação de força criptográfica e *throughput* do middleware, com intervalos de confiança de 90%, para o experimento da Figura 5.9(d)

A.2 Análise semântica de dados

Parâmetro	Valor
Capacidade de memória	512 KB
Bateria	Decrescente, 100 a 0%
Memória livre	Entre 10 KB e 8 MB
Uso de CPU	Entre 0 e 100%
Rádio	WiFi
Roteamento	Não
Mobilidade	Entre 0 e 100%
Qualidade do canal	Entre 0 e 100%
Latência	Entre 0 e 10 ms

Tabela A.19: Variação de parâmetros para experimentos pseudo-aleatórios

Parâmetro	Valor
Capacidade de memória	8 MB
Bateria	100%
Memória livre	1 MB
Uso de CPU	40%
Rádio	WiFi
Roteamento	Não
Mobilidade	10%
Qualidade do canal	80%
Latência	1 ms

Tabela A.20: Parâmetros para experimentos fixos

Parâmetro	**Valor**
Limitado por segurança	
Confidencialidade	Mandatório
Integridade	Mandatório
Autenticação	Mandatório
Máx. *overhead*	-
Máx. latência	-
Máx. memória	-
Limitado por QoS	
Confidencialidade	Opcional
Integridade	Opcional
Autenticação	Opcional
Máx. *overhead*	64 bytes
Máx. latência	12 ms
Máx. memória	20 KB
Limitações médias de segurança e QoS	
Confidencialidade	Desejável
Integridade	Desejável
Autenticação	Desejável
Máx. *overhead*	512 bytes
Máx. latência	20 ms
Máx. memória	50 KB

Tabela A.21: Restrições de QoS e segurança para os casos comparados

Apêndice B

Arquivos de Anotação Usados nos Experimentos

Arquivo de anotação para a transmissão FTP

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE protocol-description SYSTEM "anotacao.dtd">
<protocol-description>
  <declaration>
    <!-- Declaration of different transmission types -->
    <security-definition>
                <global-configuration>
                        <requirements>
          <authenticity level="Disabled" priority="Low" />
          <confidentiality level="Disabled"  priority="Low" />
                                <integrity level="Disabled" priority="Low"  />
                                <maximum-latency priority="Low" time = "0"/>
                                <minimum-transmission-rate  priority="Low" rate = "0"
                                    timeout="0" time-unit="Microseconds"/>
                                <maximum-byte-overhead-per-packet  priority="Low" overhead
                                    ="0" />
                        </requirements>
                </global-configuration>
      <configuration name="data_message">
        <requirements>
                        <authenticity    level="Mandatory" priority="High" />
                        <confidentiality level="Critical"  priority="High" />
                        <integrity level="Mandatory" priority="High"  />
        </requirements>
      </configuration>
      <configuration name="ack_message" />
    </security-definition>
  </declaration>
  <!-- Esse é um exemplo da transmissão FTP (envia dado/recebe ack) -->
  <transmission-model repeat="true">
              <send configuration="data_message" />
              <receive configuration="ack_message" />
  </transmission-model>
</protocol-description>
```

Arquivo de anotação para a transmissão uniforme

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE protocol-description SYSTEM "anotacao.dtd">
<protocol-description>
  <declaration>
    <!-- Declaration of different transmission types -->
    <security-definition>
```

```
                <global-configuration>
                        <requirements>
          <authenticity level="Disabled" priority="Low" />
          <confidentiality level="Disabled"  priority="Low" />
                                <integrity level="Disabled" priority="Low"  />
                                <maximum-latency priority="Low" time = "0"/>
                                <minimum-transmission-rate  priority="Low" rate = "0"
                                    timeout="0" time-unit="Microseconds"/>
                                <maximum-byte-overhead-per-packet  priority="Low" overhead
                                    ="0" />
                        </requirements>
                </global-configuration>
      <configuration name="data_message">
        <requirements>
                        <authenticity    level="Mandatory" priority="High" />
                        <confidentiality level="Critical"  priority="High" />
                        <integrity level="Mandatory" priority="High"  />
        </requirements>
      </configuration>
      <configuration name="ack_message" />
    </security-definition>
  </declaration>

  <!-- Esse exemplo possui a mesma quantidade de informações importantes e s/ importância --
      >
  <transmission-model repeat="true">
    <loop count="2000">
      <send-receive configuration="data_message" />
    </loop>
    <loop count="2000">
      <send-receive configuration="ack_message" />
    </loop>
  </transmission-model>
</protocol-description>
```

Arquivo de anotação para a transmissão MPEG

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE protocol-description SYSTEM "anotacao.dtd">
<protocol-description>
  <declaration>
    <!-- Declaration of different transmission types -->
    <security-definition>
                <global-configuration>
                        <requirements>
          <authenticity level="Disabled" priority="Low" />
          <confidentiality level="Disabled"  priority="Low" />
                                <integrity level="Disabled" priority="Low"  />
                                <maximum-latency priority="Low" time = "0"/>
                                <minimum-transmission-rate  priority="Low" rate = "0"
                                    timeout="0" time-unit="Microseconds"/>
                                <maximum-byte-overhead-per-packet  priority="Low" overhead
                                    ="0" />
                        </requirements>
                </global-configuration>
      <configuration name="intrapicture">
        <requirements>
                        <authenticity    level="Mandatory" priority="High" />
                        <confidentiality level="Critical"  priority="High" />
                        <integrity level="Mandatory" priority="High"  />
        </requirements>
      </configuration>
      <configuration name="predicted-picture">
        <requirements>
          <authenticity    level="Desired" priority="Medium" />
          <confidentiality level="Desired" priority="Medium" />
          <integrity level="Desired" priority="Medium"  />
        </requirements>
      </configuration>
```

```
      <configuration name="bidirectional-picture" />
    </security-definition>
  </declaration>

  <!-- Esse exemplo possui a mesma quantidade de informações importantes e s/ importância --
      >
  <transmission-model repeat="true">
    <send-receive configuration="intrapicture" />
    <loop count="3">
      <send-receive configuration="bidirectional-picture" />
    </loop>
    <send-receive configuration="predicted-picture" />
    <loop count="3">
      <send-receive configuration="bidirectional-picture" />
    </loop>
  </transmission-model>
</protocol-description>
```

Arquivo DTD descritor dos arquivos XML de anotações

```
<!ELEMENT protocol-description (declaration,transmission-model)>

<!-- First part: declaration -->

<!ELEMENT declaration (security-definition, message-definition?)>

<!-- Security Definition: kind of Security Configurations -->

<!ELEMENT security-definition (global-configuration, configuration+)>

<!ELEMENT global-configuration (requirements*)>
<!ELEMENT configuration (requirements*)>
<!ATTLIST configuration name ID #REQUIRED>

<!ELEMENT requirements (authenticity|confidentiality|integrity|maximum-latency|minimum-
    transmission-rate|retransmit-on-lost|schedule-on-bad-conditions|maximum-byte-overhead-
    per-packet)*>

<!ELEMENT authenticity EMPTY>
<!ATTLIST authenticity priority (Low|Medium|High) "Medium">
<!ATTLIST authenticity level (Disabled|Optional|Desired|Mandatory|Critical) #REQUIRED>

<!ELEMENT confidentiality EMPTY>
<!ATTLIST confidentiality priority (Low|Medium|High) "Medium">
<!ATTLIST confidentiality level (Disabled|Optional|Desired|Mandatory|Critical) #REQUIRED>

<!ELEMENT integrity EMPTY>
<!ATTLIST integrity priority (Low|Medium|High) "Medium">
<!ATTLIST integrity level (Disabled|Optional|Desired|Mandatory|Critical) #REQUIRED>

<!ELEMENT maximum-latency EMPTY>
<!ATTLIST maximum-latency priority (Low|Medium|High) "Medium">
<!ATTLIST maximum-latency time CDATA #REQUIRED>

<!ELEMENT minimum-transmission-rate EMPTY>
<!ATTLIST minimum-transmission-rate priority (Low|Medium|High) "Medium">
<!ATTLIST minimum-transmission-rate rate CDATA #REQUIRED>
<!ATTLIST minimum-transmission-rate timeout CDATA #REQUIRED>
<!ATTLIST minimum-transmission-rate time-unit (Minutes|Seconds|Microseconds|Milliseconds) #
    REQUIRED>

<!ELEMENT transmit-anyway EMPTY>
<!ELEMENT retransmit-on-lost EMPTY>
<!ELEMENT schedule-on-bad-conditions EMPTY>

<!ELEMENT maximum-byte-overhead-per-packet EMPTY>
<!ATTLIST maximum-byte-overhead-per-packet priority (Low|Medium|High) "Medium">
<!ATTLIST maximum-byte-overhead-per-packet overhead CDATA #REQUIRED>
```

```
<!-- Message Definition: kind of message comparators -->

<!ELEMENT message-definition (message)*>
<!ELEMENT message (mask-comparator|size-comparator)+>
<!ATTLIST message name ID #REQUIRED>

<!ELEMENT mask-comparator EMPTY>
<!ATTLIST mask-comparator first-byte CDATA #REQUIRED>
<!ATTLIST mask-comparator match-mask CDATA #REQUIRED>
<!ATTLIST mask-comparator mask-size CDATA #REQUIRED>
<!ATTLIST mask-comparator match-value CDATA #REQUIRED>

<!ELEMENT size-comparator EMPTY>
<!ATTLIST size-comparator operator (equal|less-than|less-or-equal|greater-than|greater-or-
    equal|different) #REQUIRED>
<!ATTLIST size-comparator value CDATA #REQUIRED>

<!-- Second part: declaration -->

<!ELEMENT transmission-model (send|receive|send-receive|loop|if)*>
<!ATTLIST transmission-model repeat (true|false) "true">

<!ELEMENT send EMPTY>
<!ATTLIST send label ID #IMPLIED>
<!ATTLIST send configuration IDREF #REQUIRED>

<!ELEMENT receive EMPTY>
<!ATTLIST receive label ID #IMPLIED>
<!ATTLIST receive configuration IDREF #REQUIRED>

<!ELEMENT send-receive EMPTY>
<!ATTLIST send-receive label ID #IMPLIED>
<!ATTLIST send-receive configuration IDREF #REQUIRED>

<!ELEMENT if (if-condition, if-then, if-else?)>

<!ELEMENT if-condition (compare-message|receive-message|send-message)+>
<!ELEMENT if-then (send|receive|send-receive|loop|if)*>
<!ELEMENT if-else (send|receive|send-receive|loop|if)*>

<!ELEMENT receive-message EMPTY>
<!ATTLIST receive-message name IDREF #IMPLIED>

<!ELEMENT send-message EMPTY>
<!ATTLIST send-message name IDREF #IMPLIED>

<!ELEMENT loop (stop-condition?,(send|receive|send-receive|loop|if)*)>
<!ATTLIST loop count CDATA #IMPLIED>

<!ELEMENT stop-condition (compare-message|receive-message|send-message)+>

<!ELEMENT compare-message EMPTY>
<!ATTLIST compare-message label IDREF #REQUIRED>
<!ATTLIST compare-message name  IDREF #REQUIRED>
```

Referências Bibliográficas

Arbaugh, W. A. (2003). Wireless security is different. *Computer*, 36(8):99–101.

Bhagyavati; Summers, W. C. e DeJoie, A. (2004). Wireless security techniques: an overview. In *Proceedings of the 1st annual conference on Information security curriculum development (InfoSecCD'04)*, pp. 82–87, New York, NY, USA. ACM Press.

Borodin, A. e El-Yaniv, R. (1998). *Online computation and competitive analysis*. Cambridge University Press, New York, NY, USA.

Capra, L.; Blair, G. S.; Mascolo, C.; Emmerich, W. e Grace, P. (2002). Exploiting reflection in mobile computing middleware. *ACM SIGMOBILE Mobile Computing and Communications Review*, 6(4):34–44.

Capra, L.; Emmerich, W. e Mascolo, C. (2001). Middleware for mobile computing: Awareness vs. transparency. In *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems (HOTOS'01)*, p. 164, Washington, DC, USA. IEEE Computer Society.

Capra, L.; Emmerich, W. e Mascolo, C. (2003). Carisma: Context-aware reflective middleware system for mobile applications. *IEEE Transactions on Software Engineering*, 29(10):929–945.

Corradi, A.; Montanari, R.; Tibaldi, D. e Toninelli, A. (2005). A context-centric security middleware for service provisioning in pervasive computing. In *Proceedings of the The 2005 Symposium on Applications and the Internet (SAINT'05)*, pp. 421–429, Washington, DC, USA. IEEE Computer Society.

Costa, D. N. O. (2008). Configuração de segurança para comunicação em ambientes móveis. Dissertação de mestrado a ser defendida na Universidade Federal de Minas Gerais.

Diffie, W. e Hellman, M. E. (1976). New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654.

Foley, S. N.; Quillinan, T. B.; O'Connor, M.; Mulcahy, B. P. e Morrison, J. P. (2004). A framework for heterogeneous middleware security. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04)*, volume 02, p. 108b, Los Alamitos, CA, USA. IEEE Computer Society.

Gall, D. L. (1991). Mpeg: a video compression standard for multimedia applications. *Communications of the ACM*, 34(4):46–58.

Gamma, E.; Helm, R.; Johnson, R. e Vlissides, J. (1995). *Design Patterns*. Addison-Wesley Professional.

Gurevich, Y. (2000). Sequential abstract-state machines capture sequential algorithms. *ACM Transactions on Computational Logic*, 1(1):77–111.

Kahn, D. (1996). *The codebreakers: the story of secret writing*. Scribner, New York, NY, USA, revised edição.

Karygiannis, T. e Owens, L. (2002). Wireless network security: 802.11, Bluetooth, and handheld devices. *NIST Special Publication 800-48*.

Li, Z.; Xu, W.; Miller, R. e Trappe, W. (2006). Securing wireless systems via lower layer enforcements. In *Proceedings of the 5th ACM workshop on Wireless security (WiSe'06)*, pp. 33–42, New York, NY, USA. ACM Press.

Mascolo, C.; Capra, L. e Emmerich, W. (2002a). Middleware for mobile computing (a survey). In *Tutorial Proceedings of the International Conference of Networking 2002*, pp. 20–58. Springer.

Mascolo, C.; Capra, L.; Zachariadis, S. e Emmerich, W. (2002b). Xmiddle: A data-sharing middleware for mobile computing. *Wireless Personal Communications*, 21(1):77–103.

Patiyoot, D. (2002). Security issues for wireless atm networks. *ACM SIGOPS Operating Systems Review*, 36(1):31–57.

Patiyoot, D. e Shepherd, S. J. (1999). Cryptographic security techniques for wireless networks. *ACM SIGOPS Operating Systems Review*, 33(2):36–50.

Pfleeger, S. L. e Pfleeger, C. P. (2006). *Security in Computing*. Prentice Hall, 4th. edição.

Potter, B. (2006). Wireless hotspots: petri dish of wireless security. *Communications of the ACM*, 49(6):50–56.

Ravi, S.; Raghunathan, A. e Potlapally, N. (2002). Securing wireless data: system architecture challenges. In *Proceedings of the 15th international symposium on System Synthesis (ISSS'02)*, pp. 195–200, New York, NY, USA. ACM Press.

Rocha, B. P. S.; Rezende, C. G. e Loureiro, A. A. F. (2007). Middleware for multi-client and multi-server mobile applications. In *Proceedings of the International Symposium on Wireless Pervasive Computing (ISWPC'07)*. IEEE Communications Society.

Savage, J. E. (1987). *The Complexity of Computing*. Krieger Publishing Co., Inc., Melbourne, FL, USA.

Shakkottai, S.; Rappaport, T. S. e Karlsson, P. C. (Oct 2003). Cross-layer design for wireless networks. *Communications Magazine, IEEE*, 41(10):74–80.

Soliman, H. S. e Omari, M. (2005). Application of synchronous dynamic encryption system in mobile wireless domains. In *Proceedings of the 1st ACM international workshop on Quality of service & security in wireless and mobile networks (Q2SWinet'05)*, pp. 24–30, New York, NY, USA. ACM Press.

Stallings, W. (2005). *Cryptography and Network Security.* Prentice Hall, 4th. edição.

Tanenbaum, A. S. (2002). *Computer Networks.* Prentice Hall PTR, 4th. edição.

Thayer Fabrega, F. J.; Herzog, J. C. e Guttman, J. D. (9-11 Jun 1998). Honest ideals on strand spaces. *Computer Security Foundations Workshop, 1998. Proceedings. 11th IEEE*, pp. 66–77.

Tripathi, A. (2002). Challenges designing next-generation middleware systems. *Communications of the ACM*, 45(6):39–42.

Wikcionário (2007). Wikcionário. http://pt.wiktionary.org.