Concretização de Protocolos com Fiabilidade Semântica*

Nuno Carvalho
Universidade de Lisboa
nunomrc@di.fc.ul.pt

José Pereira
Universidade do Minho
jop@di.uminho.pt

Luís Rodrigues
Universidade de Lisboa
ler@di.fc.ul.pt

Resumo

Os protocolos de difusão fiável são muito utilizados na construção de sistemas distribuídos. Neste contexto, a fiabilidade semântica é um novo modelo de coerência introduzido recentemente que explora o conceito de obsolescência: uma mensagem torna-se obsoleta quando o seu conteúdo é sobreposto por uma outra mensagem mais recente. O conhecimento de quais as mensagens obsoletas pode ser usado para libertar recursos, aumentando o desempenho do sistema.

Este artigo aborda os desafios de realizar uma concretização modular de uma pilha de protocolos oferecendo fiabilidade semântica. Esta concretização foi realizada usando uma moldura de objectos de suporte à composição e execução de micro-protocolos. Ao contrário do que acontece em concretizações monolíticas, numa concretização modular as mensagens em processamento na pilha de protocolos encontram-se dispersas por diferentes camadas, dificultando a tarefa de aplicar relações de obsolescência.

Palavras Chave: Difusão fiável, sincronia virtual, modularidade.

1 Introdução

Os protocolos de difusão fiável são muito utilizados na construção de sistemas distribuídos. Exemplos de utilização deste tipo de protocolos encontram-se em áreas tão diversas como a disseminação de eventos [9], a replicação de bases de dados [10], ou trabalho cooperativo [4]. Apesar de existir muito trabalho na área da optimização de protocolos de comunicação fiável [2, 3] verifica-se que o desempenho destes protocolos está limitado pelo desempenho do mais lento membro do grupo [1]. Deste modo, quanto maior e mais heterogéneo for o grupo menor é o desempenho deste tipo de protocolos. De facto, num grupo de grande dimensão, é provável

*Este trabalho foi parcialmente suportado pela FCT, através do projecto SHIFT (POSI/CHS/32869/2000)

que diferentes membros sofram situações de sobrecarga em diferentes momentos no tempo, limitando de modo quase permanente o débito observável no grupo [9].

De modo a abordar este problema, foi recentemente proposto um novo modelo de coerência para protocolos de difusão designado por *fiabilidade semântica* [6, 8]. Este modelo explora conhecimento semântico acerca das mensagens trocadas pelo protocolo para definir relações de *obsolescência de mensagens*: uma mensagem torna-se obsoleta quando o seu conteúdo (ou propósito) é sobreposto por uma outra mensagem mais recente. Este conhecimento pode ser usado para libertar recursos do sistema, omitindo a entrega de certas mensagens cujo conteúdo já não é relevante para a aplicação.

O modelo de coerência proposto em [6, 7, 8] pressupõe uma concretização monolítica do sistema. Este artigo aborda os desafios de realizar uma concretização *modular* de uma pilha de protocolos oferecendo fiabilidade semântica, usando uma moldura de objectos para a composição e execução de micro-protocolos. A principal vantagem de possuir uma concretização modular consiste na reutilização dos protocolos na contrução de outras pilhas protocolares onde a fiabilidade semântica seja necessária. Ao contrário do que acontece em concretizações monolíticas, numa concretização modular as mensagens em processamento na pilha de protocolos encontram-se dispersas por diferentes camadas, dificultando a tarefa de aplicar relações de obsolescência.

O artigo está estruturado da seguinte forma: a Secção 2 introduz com maior pormenor o problema que se pretende resolver. A Secção 3 mostra como é definida a fiabilidade semântica e as respectivas propriedades. A Secção 4 descreve a concretização do sistema de fiabilidade semântica num sistema modular. A Secção 5 mostra a avaliação efectuada e os resultados obtidos. A Secção 6 conclui o artigo e sintetiza as principais direcções futuras.

2 Motivação

Num sistema distribuído heterogéneo e de larga escala, a presença de elementos mais lentos é frequente. Diferenças na capacidade do processador, na quantidade de memória disponível ou na carga do sistema, variações na latência da rede ou efeitos dos mecanismos de controlo de fluxo são alguns dos motivos da existência de elementos lentos.

Para ilustrar o problema levantado pela existência destes elementos lentos, considera-se o caso de um grupo em que um único emissor difunde actualizações periódicas para os restantes membros do grupo. De modo a assegurar a entrega de todas as mensagens, o emissor deve guardar uma cópia de cada mensagem até que a sua recepção seja confirmada por *todos* os membros do grupo. Considere-se agora a a existência de um elemento mais lento, que não consegue processar as mensagens ao ritmo a que são enviadas. As mensagens recebidas mas ainda não processadas acumulam-se nos *buffers* de recepção desse nó. Quando estes *buffers* se esgotam, o membro mais lento deixa de poder receber novas mensagens. Por seu lado, o emissor é deste modo obrigado a guardar em memória uma cópia destas mensagens para futura retransmissão, acabando por esgotar também a sua memória. A partir deste momento o débito

de todo o grupo está limitado pelo débito do membro mais lento. O mesmo tipo de fenómeno pode ser verificado nos sistemas de comunicação em grupo que oferecem sincronia virtual [11].

Dado que este problema é inerente à comunicação fiável em grupo, e não pode ser eliminado independentemente das técnicas que se usem para mitigar o seu efeito [1, 3], é necessário procurar modelos de coerência alternativos que permitam obter um desempenho satisfatório preservando, tanto quanto possível, as vantagens da fiabilidade em termos da simplicidade no desenvolvimento das aplicações. A noção de fiabilidade semântica, que apresentamos na secção seguinte aborda este problema.

3 Fiabilidade semântica

A noção de fiabilidade semântica baseia-se na observação do comportamento de diversas aplicações baseadas na difusão de mensagens. Neste tipo de aplicações é frequente encontrar padrões de utilização em que o conteúdo de uma dada mensagem torna obsoleto o conteúdo de outras mensagens enviadas anteriormente. É pois possível encontrar execuções em que o descarte selectivo de mensagens obsoletas não compromete a correcção da aplicação. Por outro lado, o descarte destas mensagens permite libertar espaço para que novas mensagens com conteúdo actualizado sejam recebidas, aumentado efectivamente o débito com que a informação *relevante* é entregue à aplicação.

3.1 Obsolescência de mensagens

Para que o modelo de fiabilidade semântica seja útil a um grande número de aplicações, a obsolescência de mensagens tem de ser definida de uma forma genérica. Tem de ser também definida de forma a que seja possível manter a concretização da aplicação e de protocolos com fiabilidade semântica em separado.

A obsolescência de mensagens pode ser definida como uma relação ($\sqsubset$) entre mensagens. Para cada par de mensagens (m,m') relacionadas $m \sqsubset m'$, dizemos que m está *obsoleta* por m'. Esta relação significa que se $m \sqsubset m'$ e se o sistema inevitavelmente entrega m', a aplicação mantém-se correcta mesmo se m for omitida. Assim sendo, não interessa se m é entregue ou não. Esta relação é transitiva, anti-simétrica e coerente com a ordem causal de eventos [6].

Uma forma de propagar informação acerca da relação de obsolescência de mensagens é codificá-la como uma estrutura de dados na aplicação, que é então usada para anotar as mensagens na operação de difusão. Desta forma, o protocolo não precisa de saber o conteúdo das mensagens e continua independente da aplicação.

3.2 Descartar mensagens obsoletas

O protocolo que descarta mensagens é bastante simples. As mensagens contêm informação de controlo que captura as relações de obsolescência. Quando os *buffers* estão sobrecarregados, é

efectuada uma pesquisa por mensagens que estão obsoletas e essas mensagens são descartadas. Este processo pode ser efectuado sempre que uma nova mensagem é colocada no *buffer* ou pode ser accionado apenas quando o *buffer* contém uma quantidade de mensagens acima de um limiar configurável.

3.3 Como codificar a relação de obsolescência

A relação de obsolescência tem de ser codificada e colocada na mensagem pela aplicação antes de ser entregue aos protocolos que vão usar essa informação. A forma de codificar essa informação terá de ser compacta, para não gerar mensagens extensas, e terá de ser avaliada rapidamente, para que o processamento de encontrar e descartar mensagens obsoletas não se torne computacionalmente pesado. Foram propostas diferentes formas de codificar as relações de obsolescência de mensagens [7]: etiquetar por tipo, enumeração de mensagens e enumeração-k. Na concretização descrita neste artigo foi utlizada a técnica da enumeração-k. Esta técnica consiste em adicionar a cada mensagem informação de controlo que indica quais das k mensagens anteriores se tornam obsoletas. Esta aproximação pode ser codificada num vector de dígitos binários. Se o n-ésimo dígito estiver activo, a n-ésima mensagem anterior a esta fica obsoleta. A enumeração-k é bastante compacta, não gerando grandes cabeçalhos nas mensagens, e muito eficiente na verificação por parte do protocolo, visto que se podem usar operações binárias no seu processamento.

4 Concretização modular da fiabilidade semântica

Este artigo descreve uma concretização modular de uma pilha de protocolos que materializa o modelo de fiabilidade semântica. Para desenvolver esta concretização foi utilizada uma moldura de objectos desenhada para facilitar a composição modular de pilhas protocolares complexas: o sistema *Appia* [5].

4.1 Appia: um sistema de suporte à composição e execução de protocolos

O *Appia* [5] é um sistema modular de suporte à comunicação concretizado em Java. Cada módulo do *Appia* é uma camada, i.e. um micro-protocolo responsável por garantir uma determinada propriedade. Estas camadas são independentes e podem ser combinadas. Essa combinação constitui uma pilha de protocolos. Esta pilha de protocolos oferece uma Qualidade de Serviço (QdS) com as propriedades desejadas.

Definida uma QdS é possível criar um ou mais *canais* de comunicação. A cada canal está associado uma pilha de *sessões*: existe uma sessão para cada camada de protocolo. O objecto sessão permite manter o estado necessário à execução do protocolo da camada correspondente. Por exemplo, uma camada concretizando ordenação FIFO necessita de manter um ou mais números de sequência como parte do seu estado. O *Appia* permite que canais diferentes partilhem

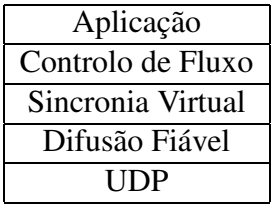


Figura 1: QdS de cada elemento do sistema.

sessões se assim o desejarem. Usando uma vez mais o exemplo de uma camada FIFO, é possível criar vários canais em que todas as mensagens partilham o mesmo número de sequência (através da partilha da sessão correspondente) ou canais cujas mensagens são ordenadas de modo independente (usando uma sessão diferente do protocolo FIFO para cada canal).

A interacção entre camadas é realizada através da troca de eventos. Os eventos são tipificados e cada camada declara ao sistema quais os eventos que cria e que está interessada em processar. O sistema optimiza o fluxo de eventos na pilha de protocolos, assegurando que os eventos só são entregues às camadas que registaram interesse no seu processamento.

Como ponto de partida para este trabalho foi utilizada uma concretização do sistema *Appia* e uma pilha de comunicação em grupo oferecendo um serviço de *sincronia virtual* [1].

4.2 A pilha de fiabilidade semântica

A pilha de protocolos suportando o modelo de fiabilidade semântica encontra-se ilustrada na Figura 1. A camada *UDP* faz a ligação entre o *Appia* e o protocolo de transporte UDP. Esta camada transforma eventos que circulam no sistema em mensagens que podem ser enviadas para a rede, e vice versa. A camada de *difusão fiável* garante que se um processo correcto envia uma mensagem m, então todos os processos correctos de um determinado grupo recebem m. Garante também FIFO na ordenação das mensagens. A camada de *sincronia virtual* oferece um serviço de filiação em grupo. A informação acerca dos processos que pertencem ao grupo é oferecida na forma de *vistas*: uma vista é constituída pelos identificadores dos vários elementos que lhe pertencem. A camada de controlo de fluxo controla o débito de mensagens entre o emissor e um conjunto de receptores. Esta concretização usa controlo de fluxo baseado em janela e só entrega uma mensagem à aplicação quando esta lhe é explicitamente pedida. A camada da aplicação é genérica e é a responsável por colocar informação de relações de obsolescência nas mensagens que envia, para que o mecanismo de descarte possa actuar nos vários *buffers* existentes, libertando recursos no sistema.

4.3 Desafios da modularidade

Como se referiu anteriormente, detectar e aplicar relações de obsolescência num sistema monolítico torna-se relativamente simples, uma vez que todas as mensagens são armazenadas por

um único componente. Aplicar estas relações num sistema modular levanta alguns problemas:

- As mensagens são processadas por diferentes camadas com um elevado grau de independência entre si o que dificulta a tarefa de saber qual a dimensão da memória ocupada num determinado momento por uma composição de protocolos.
- Não existe um *buffer* centralizado de mensagens. Os *buffers* estão distribuídos pelas sessões das diferentes camadas que constituem o canal e não podem ser acedidos directamente.
- As mensagens acumulam-se em diferentes camadas, dependendo da sua direcção e das propriedades escolhidas para a QdS.
- A quantidade de recursos usados por cada mensagem depende também da sua localização na pilha uma vez que informação de controlo vai sendo acrescentada (ou removida) aos cabeçalhos da mensagem durante o seu percurso na pilha de protocolos.

4.4 Resolução dos desafios propostos

Para resolver os desafios enumerados anteriormente, foi necessário criar um conjunto de serviços auxiliares que facilitam a interacção entre todas as camadas que necessitam de reconhecer e aplicar relações de obsolescência entre as mensagens. Estes serviços são descritos de seguida.

4.4.1 Gestor de memoria

Não existia no *Appia* uma forma de saber a quantidade de memória ocupada por mensagens num determinado canal. Assim, foi criado um objecto denominado MEMORYMANAGER que mantém esta informação. O MEMORYMANAGER está associado a um canal. Todas as camadas que pertencem a um determinado canal têm acesso ao MEMORYMANAGER.

Na criação do MEMORYMANAGER é indicado a quantidade de memória que vai gerir e um *limiar*. Este limiar serve para separar a quantidade de memória que é usada por mensagens de controlo e a quantidade de memória que é usada por mensagens da aplicação. Esta funcionalidade é necessária porque é necessário reservar memória suficiente para mensagens de controlo (e.g. da sincronia virtual e de controlo de fluxo). Do ponto de vista da aplicação, o tamanho máximo de memória é atingido quando o MEMORYMANAGER atinge o limiar definido na sua criação.

O MEMORYMANAGER exporta a seguinte interface para as camadas: USED(), indica a quantidade de memória usada num determinado momento; ABOVETHRESHOLD(), indica se o tamanho máximo reservado para as camadas foi atingido; (SET/GET)MAXTHRESHOLD(), altera ou indica a quantidade de memória máxima que pode ser usada pelas camadas.

As próprias mensagens também têm acesso ao MEMORYMANAGER. Quando uma camada coloca mais dados numa mensagem, esta acede ao respectivo MEMORYMANAGER e reserva a

quantidade de memória necessária. Quando uma camada retira dados de uma mensagem, esta liberta os respectivos recursos. Esta funcionalidade é transparente para as camadas.

Desta forma, o sistema *Appia* foi aumentado com um novo mecanismo que permite atribuir uma quantidade de memória a um determinado canal. É da responsabilidade das camadas gerir as suas mensagens de acordo com a informação que podem obter do MEMORYMANAGER.

4.4.2 *Buffers* descentralizados

Como foi já descrito neste artigo, cada camada concretiza uma determinada funcionalidade (ou propriedade). A independência entre as várias camadas é importante num sistema modular como o *Appia*. Numa primeira abordagem, poderia pensar-se que era possível aplicar a relação de obsolescência de mensagens numa única camada. No entanto, como já foi referido, as diversas mensagens que são retidas pela pilha de protocolos encontram-se distribuídas pelas diversas camadas. Isto obrigaria a que a camada de obsolescência tivesse acesso aos *buffers* das restantes camadas, o que violaria a sua modularidade.

Uma solução possível para inserir fiabilidade semântica no *Appia* é definir não uma nova camada, mas um *buffer sensível à semântica das mensagens*. Para respeitar a modularidade, todas as camadas usam uma interface comum na sua concretização. Na interface existem primitivas de inserção e remoção de mensagens. Podem existir várias concretizações para esta interface, nomeadamente concretizações com e sem fiabilidade semântica. Deste modo, só é necessário que a camada que usa um *buffer* indique qual a concretização que deseja: com fiabilidade semântica ou não.

Um *buffer sensível à semântica* é responsável por verificar se existem mensagens obsoletas e descartá-las, libertando recursos do sistema. Isto pode ser feito quando se insere uma nova mensagem no *buffer*.

4.4.3 Pontos de estrangulamento na pilha

A Figura 2 mostra como se propagam os eventos pelas diferentes camadas. As mensagens enviadas pela aplicação percorrem a pilha de protocolos no sentido descendente. Cada camada acrescenta à mensagem cabeçalhos com informação de controlo. Quando a mensagem é processada pela camada de difusão fiável é criada uma cópia da mensagem para futura retransmissão. Esta cópia é mantida até que a recepção da mensagem seja confirmada por todos os membros do grupo. Deste modo, as mensagens que percorrem a pilha no sentido descendente são acumuladas pela camada de difusão fiável.

Quando uma mensagem é recebida, essa mensagem vai percorrer igualmente a pilha, mas agora no sentido ascendente, até chegar à camada de controlo de fluxo. O tamanho desta mensagem vai ficando cada vez mais reduzido uma vez que os cabeçalhos são removidos das mensagens à medida que esta é processada pelas diferentes camadas. Se a aplicação for muito lenta a consumir mensagens, estas vão-se acumulando na camada de controlo de fluxo.

Figura 2: Fluxo de eventos (ou mensagens) dentro de um processo.

Deste modo, as mensagens acumulam-se em pontos diferentes na pilha, consoante o sentido (ascendente ou descendente) do seu processamento. Como as mensagens acumuladas por um emissor são maiores, o MEMORYMANAGER terá de ser configurado de forma diferente em relação aos receptores. Depois de testes efectuados com a pilha existente, verifica-se que a quantidade de memória necessária para um emissor é 35% maior do que para um receptor. É importante tomar em conta este pormenor no momento da configuração de cada elemento para que todo o sistema funcione com eficiência.

4.4.4 Garantir propriedades de sincronia virtual

Uma das principais garantias da sincronia virtual é que todos os processos correctos recebem o mesmo conjunto de mensagens entre duas vistas consecutivas. Para permitir o descarte de mensagens obsoletas, a definição de sincronia virtual teve de ser adaptada, sendo proposto o modelo de *sincronia virtual com semântica* [7]. O requisito de todos os processos entregarem o mesmo conjunto de mensagens é relaxado de forma a que todos os processos entreguem o mesmo conjunto de mensagens não obsoletas. Mais precisamente: Se um processo p que pertence a duas vistas consecutivas v_i e v_{i+1} entrega m na vista v_i, então todos os processos entregam uma mensagem m' na vista v_i de maneira que $m' = m$ ou $m \sqsubset m'$. Para oferecer estas garantias, o *buffer* que concretiza semântica para a sincronia virtual tem de garantir que quando descarta uma mensagem m, existe uma outra mensagem m' tal que $m \sqsubset m'$ e m' vai ser entregue nessa vista.

4.4.5 Evitar o aumento de mensagens de controlo

Algumas camadas necessitam de enviar mensagens de controlo aos outros elementos. Por exemplo, a camada que concretiza controlo de fluxo precisa de informar os outros elementos do grupo quando a sua janela de recepção está cheia. Este tipo de notificações ocorre quando a rede já

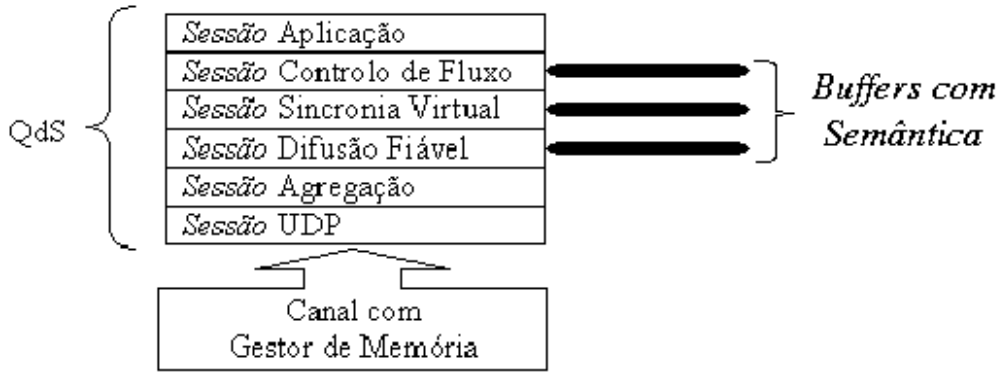


Figura 3: Arquitectura final do sistema.

está congestionada, logo, injectar ainda mais mensagens na rede é de evitar. Para resolver este problema, foi criada uma camada de *agregação* (*piggybacking*), que como o seu nome indica agrega mensagens de controlo. Esta camada é colocada entre a camada UDP e a camada de difusão fiável.

Assim, informação de controlo necessária é enviada localmente pela pilha para a camada de agregação, a qual é responsável por acrescentar a informação de controlo que recebe das restantes camadas às mensagens que são enviadas. Para as mensagens no sentido ascendente, esta camada extrai a informação de controlo e destribui-a pelas camadas interessadas. A camada de agregação não evita totalmente a transmissão de mensagens exclusivamente de controlo pois pode haver momentos em que a aplicação se abstém de transmitir dados mas, de qualquer modo, reduz substancialmente o número de mensagens de controlo enviadas.

4.5 Arquitectura final

A Figura 3 ilustra a pilha de protocolos que suporta fiabilidade semântica, com todos os componentes necessários. Neste modelo, cada canal tem um gestor de memória, ao qual todas as camadas têm acesso. Foram identificados três *buffers* onde se acumulam mensagens: o *buffer* de recepção da camada de controlo de fluxo, o *buffer* da sincronia virtual que mantém mensagens até que estas sejam consideradas estáveis e o *buffer* de envio da camada de difusão fiável. Todos estes *buffers* têm a capacidade de avaliar as mensagens e descobrir relações de obsolescência. As mensagens que estão nesta relação são descartadas e o gestor de memória é automaticamente actualizado.

5 Avaliação

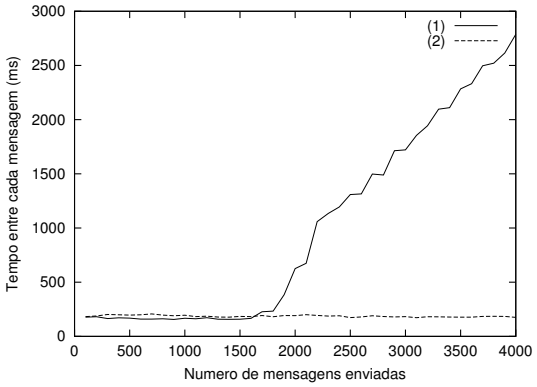
De modo a avaliar o desempenho da concretização modular de uma pilha de protocolos suportando o modelo de fiabilidade semântica foi realizado um conjunto de testes experimentais utilizando uma rede de computadores baseados em *Intel Pentium III*, a 800Mz com 320Mb de memória RAM interligados por uma *Ethernet* a 100Mbs. Para simular o receptor lento, foi criada uma camada extra que tem como função simular uma aplicação mais lenta a consumir mensagens. Deste modo, os *buffers* de recepção na camada de controlo de fluxo acumulam mensagens mais rapidamente. Foi também utilizada uma camada que perde mensagens para avaliar o desempenho dos protocolos sobre redes menos fiáveis. O activar desta camada de teste faz com que a camada de difusão fiável acumule também mensagens que necessitam de ser reenviadas.

5.1 Resultados

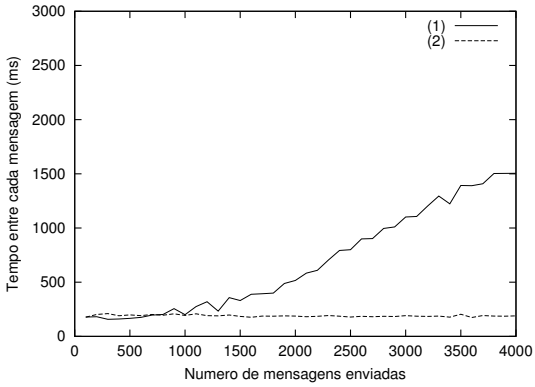
O objectivo dos testes é medir o impacto que a presença de um elemento lento tem sobre o desempenho global do grupo. Os testes baseiam-se numa configuração com três processos: um emissor, que dissemina valores para os restantes elementos do grupo, e dois receptores. Um dos receptores processa as mensagens sem qualquer atraso e outro receptor representa um membro lento, sendo possível configurar o atraso introduzido no processamento de cada mensagem. A métrica utilizada para caracterizar o comportamento da pilha de protocolos é o tempo médio entre a recepção de cada duas mensagens consecutivas no receptor mais rápido. Para obter estes valores foi desenvolvida uma camada de captura e registo dos tempos de recepção das mensagens.

A Figura 4(a) ilustra o comportamento do sistema quando a obsolescência de mensagens não é utilizada. A figura compara o comportamento do sistema quando ambos os receptores são rápidos e no caso em que um receptor é mais lento. Em ambos os casos recorreu-se a um teste em que o emissor envia 4000 mensagens separadas por um intervalo de $100ms$. O receptor rápido consome mensagens assim que estas estão disponíveis, mas o receptor lento (quando existe) começa por consumir mensagens com um intervalo de $100ms$ que vai aumentando progressivamente. Em cada 100 mensagens recebidas, o atraso aumenta em mais $100ms$. No primeiro caso, o sistema mantém-se estável. No segundo caso o sistema está estável até ao ponto em que os *buffers* do receptor lento e do emissor estão cheios. Isto acontece quando foram já enviadas aproximadamente 1700 mensagens. A partir deste ponto, a presença de um receptor que vai ficando progressivamente mais lento faz com que o tempo entre recepção de mensagens num receptor rápido seja cada vez maior. Este atraso dá-se devido a dois factores: (i) um dos receptores é lento e (ii) esse receptor perde algumas mensagens.

A Figura 4(b) mostra a vantagem da aplicação da obsolescência de mensagens. A aplicação envia dois tipos de mensagens: uma mensagem que contém informação independente (não pode ser descartada dos *buffers*) e outra que consiste numa actualização da mesma informação. Estas mensagens são enviadas de forma intercalada. Assim, no caso em que um receptor é



(a) Sem obsolescência: com (1) e sem (2) receptor lento.



(b) Com obsolescência: com (1) e sem (2) receptor lento.

Figura 4: Resultados experimentais.

mais lento e quando não são efectuadas operações de descarte, 50% das mensagens presentes nos *buffers* estão obsoletas. O facto de um dos receptores ser lento faz com que o descarte seja efectuado no *buffer* de recepção da camada de controlo de fluxo. Por outro lado, como o receptor lento também perde algumas mensagens, a obsolescência actua também no *buffer* da camada de difusão fiável do emissor e na camada da sincronia virtual que garante que as mensagens estão estáveis. É este conjunto de factores que provoca um aumento de desempenho no grupo. Finalmente, nas Figuras 4(a) e 4(b), as linhas que representam a configuração em que não existem receptores lentos estão ao mesmo nível, o que indica que o desempenho do sistema não é afectado pelo processamento extra associado à verificação das relações de obsolescência.

Os resultados experimentais obtidos com a concretização modular confirmam os resultados apresentados em [6] e obtidos com recurso à simulação.

6 Conclusões e trabalho futuro

Este artigo descreve uma concretização modular de protocolos com fiabilidade semântica. O uso deste tipo de protocolos pode ser vantajoso em sistemas distribuídos de larga escala como sistemas de troca de informação sobre stocks [6], replicação de servidores [8] e até jogos multi-utilizador [7].

O trabalho efectuado anteriormente e descrito em [6, 8, 7] pressupõe uma concretização monolítica. Neste artigo mostramos a viabilidade de concretizar este modelo de coerência recorrendo à composição de micro-protocolos. O trabalho levantou vários problemas relacionados com a necessidade de coordenar diversas camadas de protocolos no acesso aos recursos do

sistema. Estes problemas foram resolvidos aumentando o sistema *Appia* com um conjunto de novos serviços incluindo componentes de gestão de memória e *buffers* que detectam relações de obsolescência entre mensagens.

A principal vantagem de possuir uma concretização modular consiste na reutilização dos protocolos na contrução de outras pilhas protocolares onde a fiabilidade semântica seja necessária. É possível acrescentar protocolos à pilha apresentada, modificando a QoS oferecida e mantendo fiabilidade semântica. Os *buffers* com semântica contruídos podem também ser reutilizados na contrução de novos protocolos que necessitam de guardar mensagens temporariamente (e.g. protocolos de ordenação de mensagens). A modularidade é também vantajosa na manutenção e expansão do sistema, tornando mais simples a tarefa de localizar o código onde se podem realizar melhoramentos.

A concretização resultante foi avaliada experimentalmente, confirmando as vantagens estimadas recorrendo a simulações e a concretizações monolíticas. Verificámos também que eficiência do nosso protótipo é comparável à de um sistema que não suporta fiabilidade semântica nos casos em que todos os receptores são rápidos. No futuro vamos estudar o comportamento deste sistema com tráfego real usando jogos distribuídos e multi-utilizador como caso de estudo.

Referências

[1] K. Birman. A review of experiences with reliable multicast. *Software Practice and Experience*, 29(9):741–774, July 1999.

[2] Sally Floyd, Van Jacobson, Ching-Gung Liu, Steven McCanne, and Lixia Zhang. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM Transactions on Networking*, 5(6):784–803, December 1997.

[3] K. Guo. *Scalable Message Stability Detection Protocols*. PhD thesis, Cornell Univ., Computer Science, May 1998.

[4] H. Miranda, M. Antunes, L. Rodrigues, and A. Rito Silva. Group communication support for dependable multi-user object oriented environments. In *Proceedings of the International SRDS Workshop on Dependable System Middleware and Group Communication (DSMGC 2000), in conjunction with the IEEE Symposium on Reliable Distributed Systems (SRDS'19)*, Nurnberg, Germany, October 2000.

[5] H. Miranda, A. Pinto, and L. Rodrigues. Appia, a flexible protocol kernel supporting multiple coordinated channels. In *Proceedings of the 21st International Conference on Distributed Computing Systems*, pages 707–710, Phoenix, Arizona, April 2001. IEEE.

[6] J. Pereira, L. Rodrigues, and R. Oliveira. Semantically reliable multicast protocols. In *Proceedings of the 19th IEEE Symposium on Reliable Distributed Systems (SRDS'19)*, pages 60–69, Nurnberg, Germany, October 2000.

[7] J. Pereira, L. Rodrigues, and R. Oliveira. Reducing the cost of group communication with semantic view synchrony. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, pages 293–302, Washington (DC), USA, June 2002.

[8] J. Pereira, L. Rodrigues, and R. Oliveira. Semantically reliable broadcast: Sustaining high throughput in reliable distributed systems. In P. Ezhilchelvan and A. Romanovsky, editors, *Concurrency in Dependable Computing*, chapter 10. Kluwer, 2002. (to appear).

[9] R. Piantoni and C. Stancescu. Implementing the Swiss Exchange Trading System. In *Proc. of The Twenty-Seventh Annual Intl. Symp. on Fault-Tolerant Computing (FTCS'97)*, pages 309–313. IEEE, June 1997.

[10] L. Rodrigues, H. Miranda, R. Almeida, J. Martins, , and P. Vicente. Strong replication in the globdata middleware. In *Proceedings of the Workshop on Dependable Middleware-Based Systems*, Washington D.C., USA, June 2002. IEEE. (Part of Dependable Systems and Networks Conference, DSN 2002) (to appear).

[11] R. van Renesse, K. Birman, and S. Maffeis. HORUS: A flexible group communication system. *Communications of the ACM*, 39(4):76–83, April 1996.