Siena

Tecnologias de Middleware

2006/2007 Fernando Martins - fmp.martins@gmail.com

Tópicos

- Objectivo
- Motivação
- Serviço de Notificação de Eventos
- Abordagem Simples
- Abordagem Distribuída
- Siena – Visão
- Semântica
- Topologias
- Rotas e Estratégias
- Algoritmos e Topologias
- Testes e Resultados
- Trabalho Futuro
- Conclusão

Objectivo

- Desenhar e avaliar um serviço de notificação de eventos de larga escala.

Motivação

- Maximizar a expressividade no mecanismo de selecção sem sacrificar a escalabilidade do mecanismo de entrega.

Serviço de Notificação de Eventos

- **Serviço de Notificação de Eventos:** infraestrutura que suporta a construção de sistemas baseados em eventos onde:
  - geradores de eventos publicam notificações para a infraestrutura e
  - os consumidores de eventos efectuam subscrições na infraestrutura para receber notificações.

- Serviços Primários:
  - selecção de notificações;
  - entrega de notificações.

Abordagem Simples

- Uso de um servidor central onde:
  - todas as subscrições são guardadas;
  - todas as notificações são enviadas;
  - as notificações são avaliadas de acordo com as subscrições;
  - as notificações são enviadas a todos os subscritores.

- **Problema:** escalabilidade impraticável no âmbito da internet.

Abordagem Distribuída

- **Solução:** arquitectura assente sobre um sistema distribuído onde as actividades são espalhadas pela rede, se possível explorando a localidade, e com um crescimento de complexidade razoável.

- O sistema distribuído é composto por servidores interligados onde cada servidor serve um subconjunto de clientes do serviço, sendo que cada cliente interage com o servidor localmente acessível.

Abordagem Distribuída

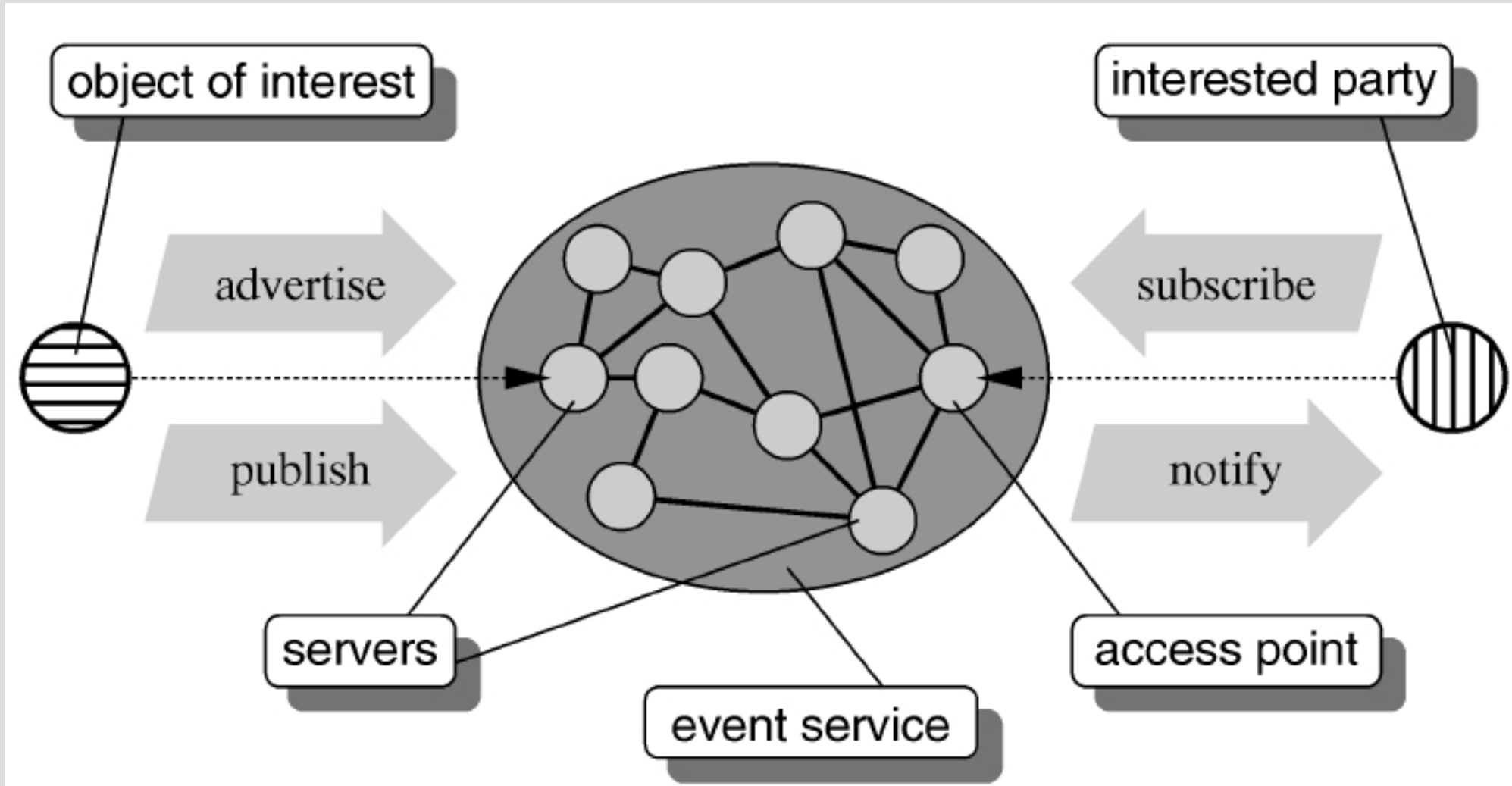


Fig. 1. Distributed event notification service.

Abordagem Distribuída

- **Topologia de Interligação:** qual a configuração de ligação dos servidores?
- **Algoritmos de Rota:** que informação deve ser comunicada entre os servidores para permitir a entrega correcta e eficiente de mensagens?
- **Estratégia de Processamento:** onde e como devem as mensagem ser processadas de forma a optimizar o tráfego de mensagens?

Siena – Visão

- Sistema genérico de notificação de eventos, e acessível a partir de qualquer ponto de uma rede de larga abrangência.
- Apto ao suporte de aplicações altamente distribuídas que necessitam de interacção com vários graus de granularidade.
- Conceptualizado com uma implementação de uma rede de servidores que disponibilizam pontos de acesso aos clientes.

Siena – Visão

- Clientes usam os pontos de acesso para:
  - publicitar informação sobre eventos e para publicar múltiplas notificações do tipo publicitado;
  - subscrever notificações de interesse.
- O serviço:
  - usa os pontos de acesso para notificar os clientes através da entrega de notificações de interesse;
  - efectua um processo de selecção determinando que notificações publicadas são interessantes para que clientes, suportando filtros e padrões;
  - usa o processo de selecção para optimizar a comunicação na rede, enviando as notificações apenas aos clientes interessados pela melhor rota.

API

- **Publish:** publicação de uma notificação.
- **Subscribe:** subscrição de um evento.
- **Unsubscribe:** cancelamento de subscrição.
- **Advertise:** publicitação de notificações potencialmente geradas mais tarde.
- **Unadvertise:** cancelamento de publicitação.

Table I. Interface of SIENA

publish(notification *n*)
subscribe(string *identity*, pattern *expression*)
unsubscribe(string *identity*, pattern *expression*)
advertise(string *identity*, filter *expression*)
unadvertise(string *identity*, filter *expression*)

Semântica

- **Notificação:** um evento de notificação, ou simplesmente uma notificação é um conjunto de atributos tipificados. Cada atributo individual possui um tipo, um nome e um valor.
- Os tipos dos atributos são tipos primitivos das linguagens de programação e bases de dados.
- Cada tipo de atributo possui um conjunto de operadores fixo e pré-definido.
- Permite uma maior escalabilidade.

string	*class = finance/exchanges/stock*
time	*date = Mar 4 11:43:37 MST 1998*
string	*exchange = NYSE*
string	*symbol = DIS*
float	*prior = 105.25*
float	*change = -4*
float	*earn = 2.04*

Semântica

- **Filtro:** um filtro de eventos, ou simplesmente um filtro, selecciona notificações através da especificação de restrições de valores em atributos.
- Operadores: =, ≠, <, >, *, >*, <*, any, ...
- O uso de múltiplas restrições resulta num filtro construído com a conjunção de todas as restrições.

string	class	> * *finance/exchanges/*
string	exchange	= *NYSE*
string	symbol	= *DIS*
float	change	> *0*

Notification		Subscription
string what = alarm *time* date = 02:40:03	$\prec_S^N$	*string* what = alarm
string what = alarm *time* date = 02:40:03	$\not\prec_S^N$	*string* what = alarm *integer* level > 3

Semântica

- **Padrão:** um padrão pode ser visto como uma combinação de filtros aplicado a uma ou mais notificações baseado na combinação formada da restrição de valores.

- Um padrão é assim um tipo de subscrição sofisticado:
  - construído à custa do agrupamento de subscrições simples;
  - expressa o interesse em receber notificações apenas se as mesmas satisfizerem um padrão.

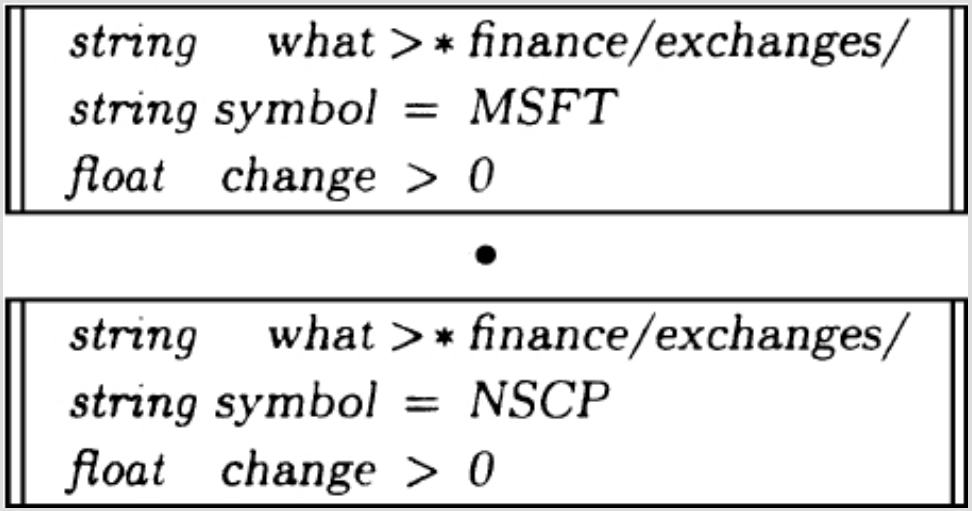
Semântica

- O Siena restringe os padrões a sequências de filtros comparados de forma temporal.

$$A \cdot B = A_i^j \cdot B_k^m, i < k \; e \; j < m.$$

$$B_4^1 A_3^2 A_1^3 B_2^4 A_5^5 B_6^6 A_7^7 B_8^8$$

$$A \cdot B = \{ A_3^2 \cdot B_6^6, A_7^7 \cdot B_8^8 \}$$



Semântica

- **Publicitação:** define o conjunto de notificações potencialmente geradas por um objecto de interesse. O uso de múltiplas restrições numa publicitação é uma disjunção.
- **Motivação:** informar o serviço sobre o tipo de notificações que serão geradas por que objectos de interesse, permitindo a melhor direcção na propagação das subscrições.

Notification		Advertisement
string what = alarm	$\prec_A^N$	*string* what = alarm *string* what = login *string* username = *any*
string what = alarm *time* date = 02:40:03	$\not\prec_A^N$	*string* what = alarm *string* what = login *string* username = *any*

Semântica

- **Cancelamento:** uma subscrição ou uma publicitação podem ser canceladas através da identificação do objecto e da especificação do filtro, ou padrão.
- Um cancelamento ou cobre uma subscrição, ou publicitação, e efectua o cancelamento, ou então não tem qualquer efeito.
- Um cancelamento com um filtro mais genérico que o especificado na subscrição cancela todas as subscrições que engloba.

```
subscribe(X, "Change > 10")
unsubscribe(X, "Change > 0")
```

Semântica

- **Tempo:** a semântica do Siena depende da ordem de recepção e processamento dos pedidos.
- Siena associa um *timestamp* a cada notificação de forma a indicar o tempo da sua publicação de forma a ajudar o serviço a detectar e a ter em conta os efeitos de latência.
- Um cancelamento de uma subscrição antes de uma subscrição não tem qualquer efeito.

Semântica

- **Semântica Baseada em Subscrição:**
  - Definida apenas pela relação de cobertura entre a notificação e a subscrição e a sua extensão aos padrões.
  - As publicitações podem ser usadas, mas apenas como optimização.
  - Uma notificação *n* é entregue a um interessado *X* se e só se *X* submeteu pelo menos uma subscrição *s* tal que $n \prec_S^N s$

Semântica

- **Semântica Baseada em Publicitação:**
  - Ambas a publicitação e subscrição são usadas.
  - Uma notificação *n* publicada pelo objecto *Y* é entregue ao interessado *X* se e só se *Y* publicitou um filtro *a* que cobre *n*, i.e. tal que $n \prec_A^N a$ e *X* registou uma subscrição *s* que cobre *n*, i.e. $n \prec_S^N s$

Topologias

- Os servidores necessitam de comunicar para cooperarem de forma distribuída, construindo assim uma topologia de interligação servidor/servidor.

- Três arquitecturas base:
  - Hierárquica;
  - Peer-To-Peer Acíclica;
  - Peer-To-Peer Genérica.

Topologias

- **Arquitectura Hierárquica:** um servidor pode ter várias ligações de entrada de “servidores cliente” mas tem apenas uma saída para o *master*.

- Servidor recebe mensagens dos seus clientes, propaga-as para o seu *master*, e envia notificações apenas para os seus clientes.
- Cada servidor representa um ponto crítico de falha.
- Modelo não é escalável para a internet.

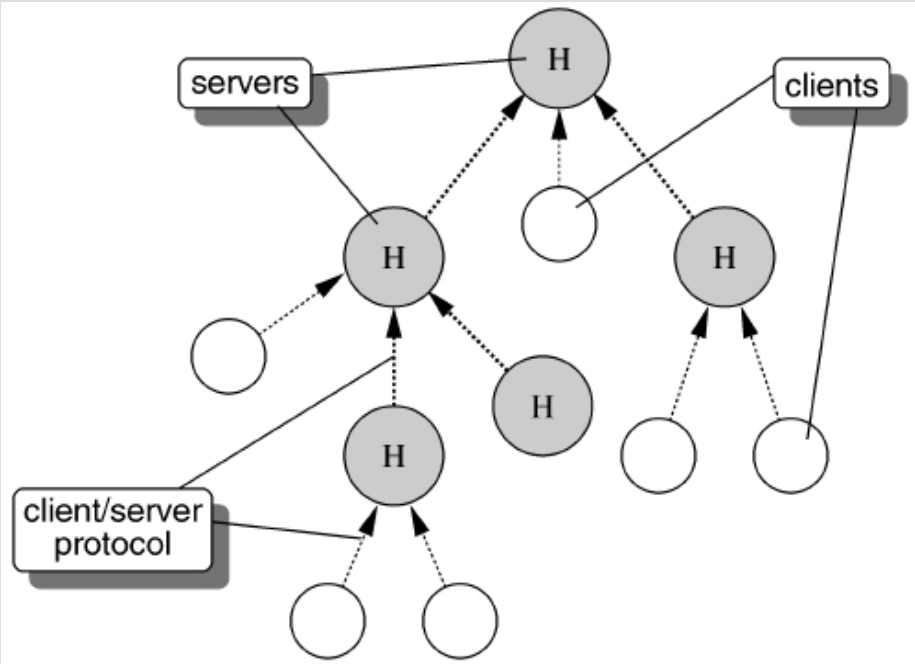


Fig. 5. Hierarchical client/server architecture.

Topologias

- **Arquitectura P2P Acíclica:** os servidores estão ligados formando um grafo acíclico e comunicam de forma bidireccional.

- Os procedimentos de comunicação entre os servidores têm de garantir a não existência de ciclos.
- Cada servidor representa um ponto crítico de falha, podendo deixar uma sub-rede isolada.

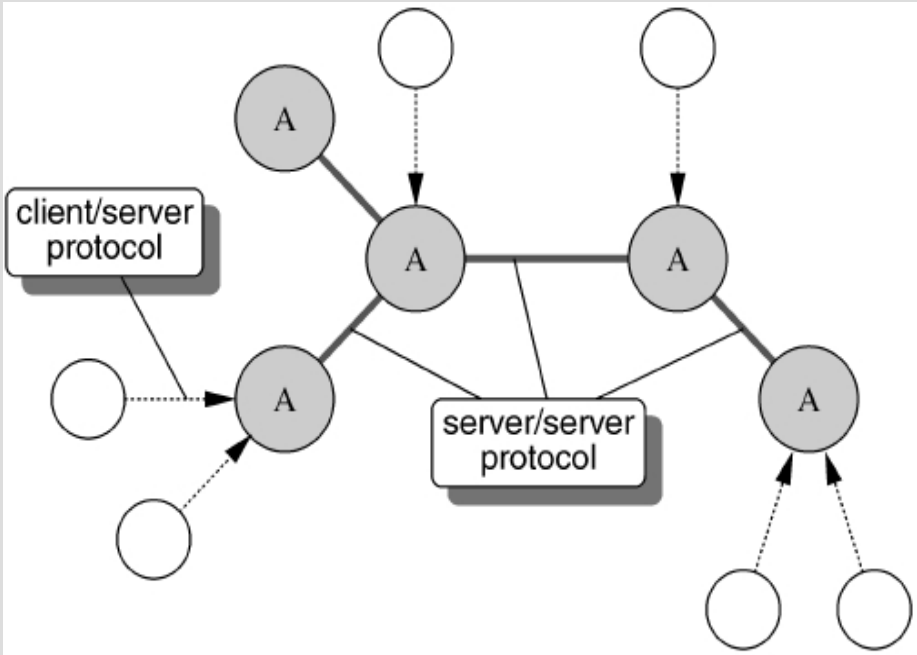


Fig. 6. Acyclic peer-to-peer server architecture.

Topologias

- **Arquitectura P2P Genérica:** os servidores estão ligados formando um grafo e comunicam de forma bidireccional.

- A nível de comunicação oferece maior flexibilidade na configuração e necessita de menos coordenação.
- A redundância permite maior robustez mas necessita de algoritmos para evitar caminhos cíclicos.

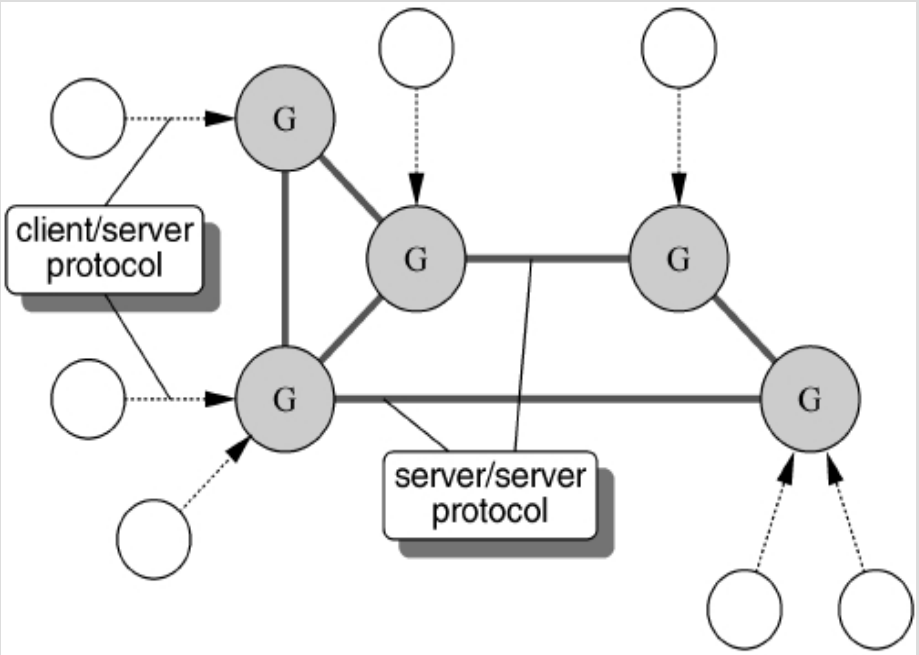


Fig. 7. General peer-to-peer server architecture.

Topologias

- **Arquitecturas Híbridas:** no mundo real existem diferentes necessidades onde certas arquitecturas se adequam melhor que outras.
- Num ambiente controlado de uma intranet, uma arquitectura hierárquica pode ser a melhor escolha.
- A ligação com serviços externos na internet pode ser feita através de uma arquitectura P2P genérica.

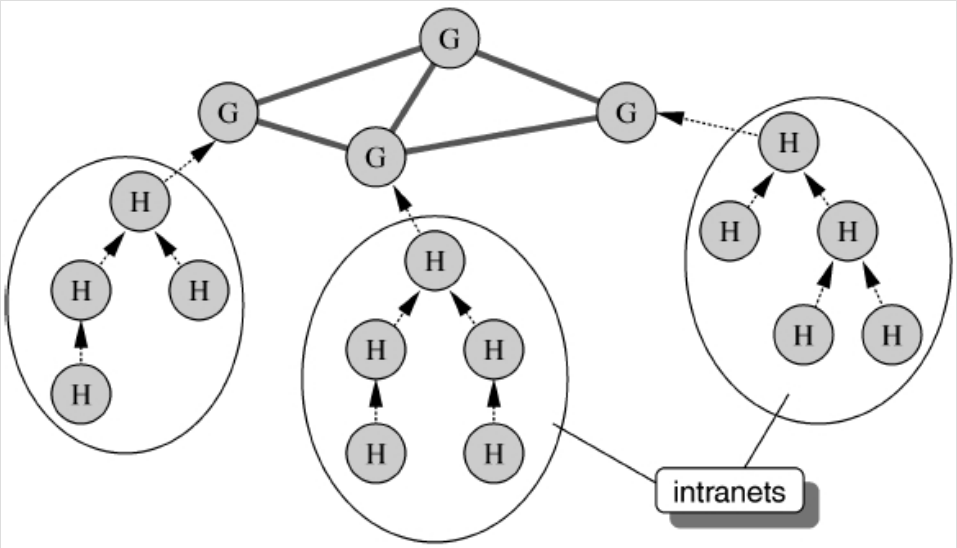


Fig. 8. Hierarchical/general hybrid server architectures.

Rotas e Estratégias

- Estando estabelecida a topologia de servidores, é necessário garantir que:
  - As notificações “encontram” as subscrições algures na rede;
  - As notificações são seleccionadas de acordo com as subscrições;
  - As notificações são entregues aos subscritores.

Rotas e Estratégias

- **Estratégias de Rota:** Siena envia a notificação em direcção aos servidores que possuem clientes interessados na notificação em causa. O mesmo se aplica aos padrões de notificação.
- Princípios genéricos que se tornaram requisitos dos algoritmos de rota do Siena:
  - Replicação a Jusante.
  - Avaliação a Montante.

Rotas e Estratégias

- **Replicação a Jusante:** uma notificação deve ser enviada como uma única cópia o mais longe possível de forma a ser replicada a jusante.

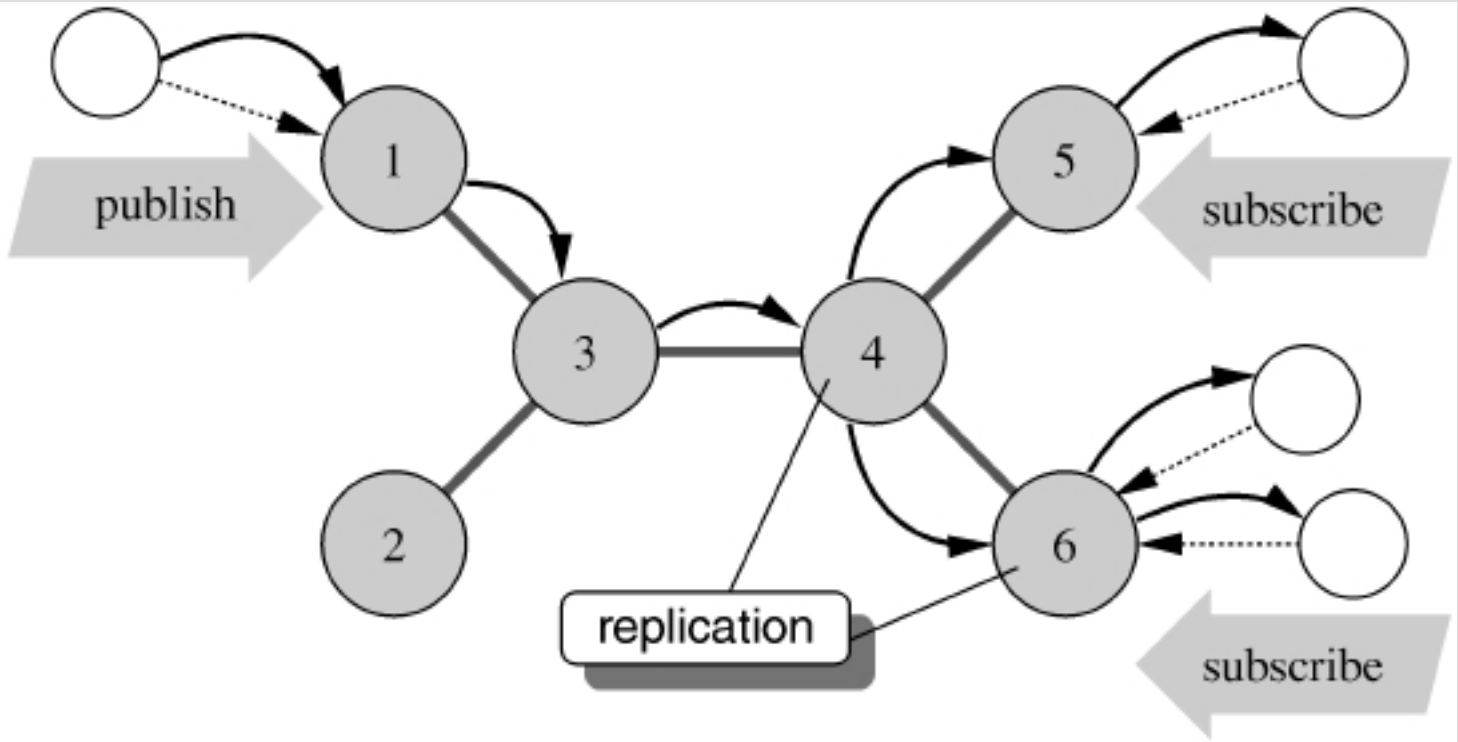


Fig. 10. Downstream notification replication.

Rotas e Estratégias

- **Avaliação a Montante:** a aplicação de filtros e a montagem de padrões deve fazer-se o mais próximo das fontes de publicação, a montante.

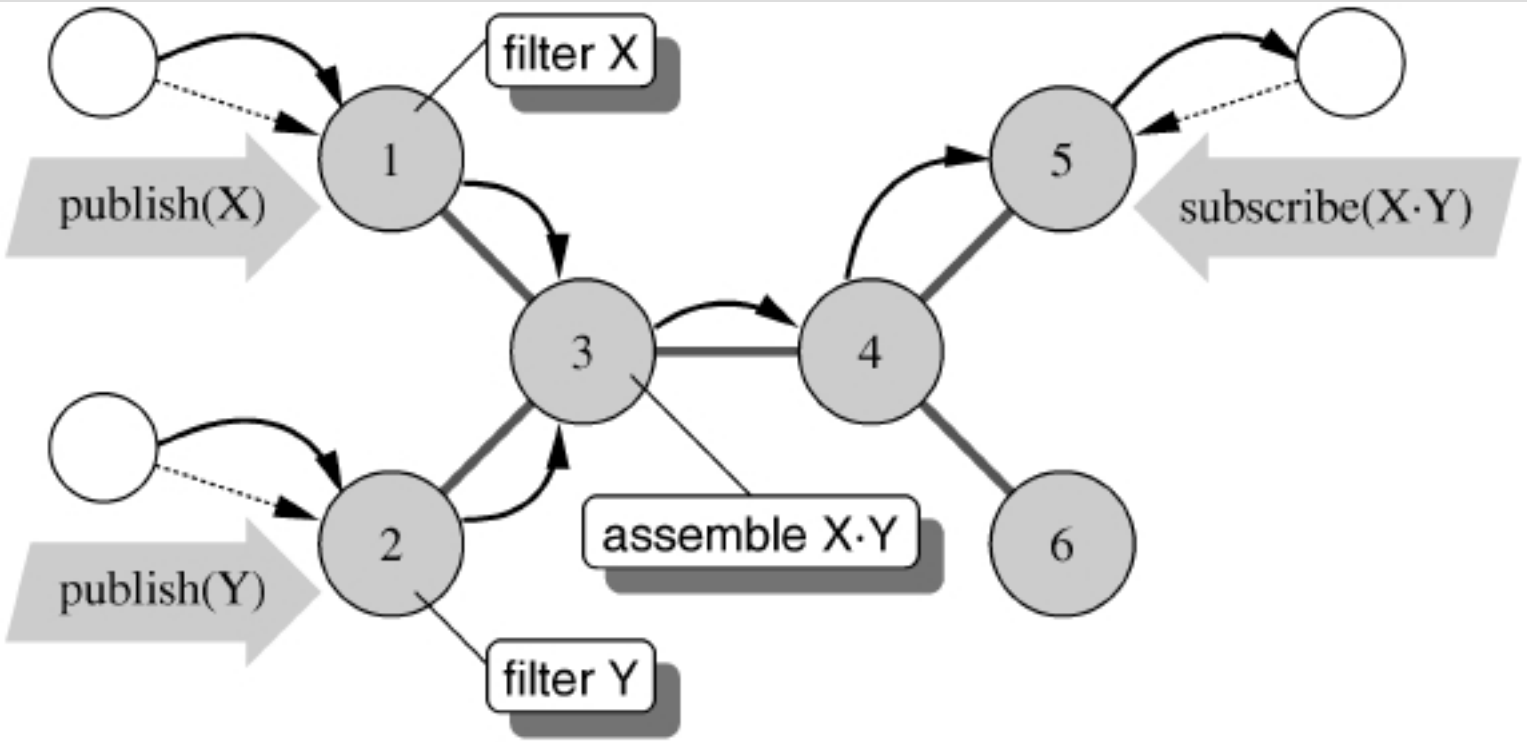


Fig. 11. Upstream filter and pattern evaluation.

Rotas e Estratégias

- Os princípios são implementados por duas classes de algoritmos de rota:
  - **Expedição de Subscrição:** subscrição define o caminho da notificação. Numa implementação que não usa publicitação, a subscrição é propagada em forma de árvore pela rede (*broadcast* da subscrição) e a rota usada pela notificação é definida pelo caminho inverso usado na subscrição.
  - **Expedição de Publicitação:** publicitação define o caminho da subscrição e da notificação. Numa implementação com publicitação, é seguro enviar a subscrição na direcção da origem da publicitação. *Broadcast* da publicitação define o caminho da subscrição e esta activa o caminho da notificação.

Algoritmos e Topologias

- Implementação dos algoritmos nas várias arquitecturas, hierárquica e P2P.

Expedição de Subscrição e Expedição de Publicitação
vs.
Arquitectura Hierárquica e Arquitectura P2P

Algoritmos e Topologias

- **Filter Poset:** (*partially ordered set*) estrutura comum aos diferentes algoritmos e topologias que permite manter o rasto dos pedidos, das suas relações, origens e propagação.
- P_S denomina um *poset* de subscrição e P_A denomina um *poset* de publicitação.

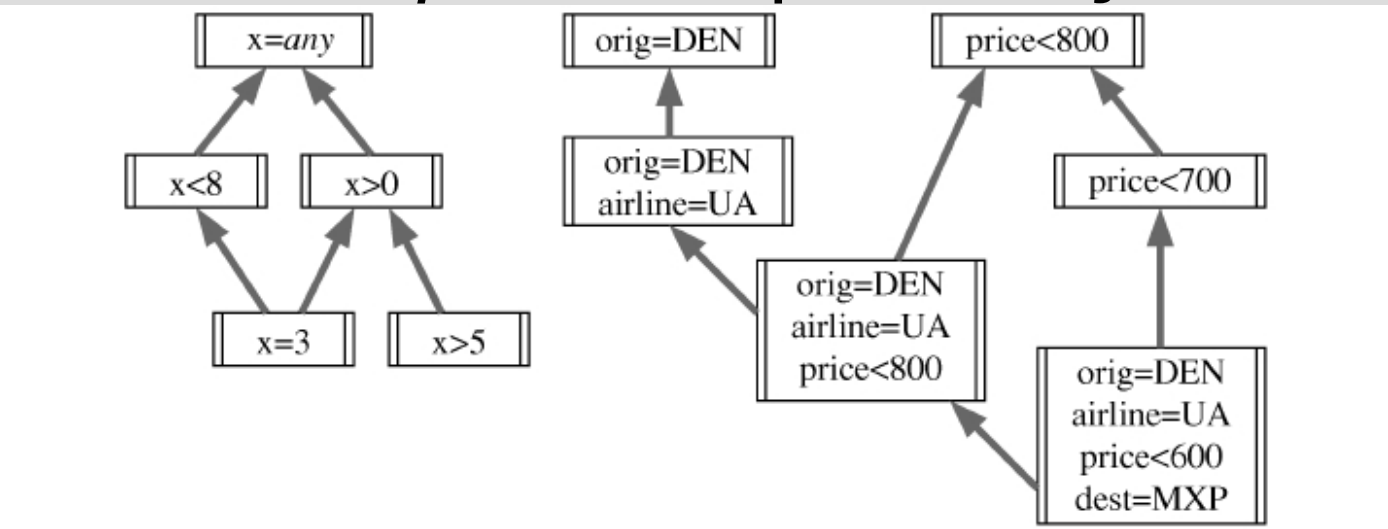


Fig. 12. Example of a poset of simple subscriptions. Arrows represent the *immediate* relation $\prec_S^S$

Algoritmos e Topologias

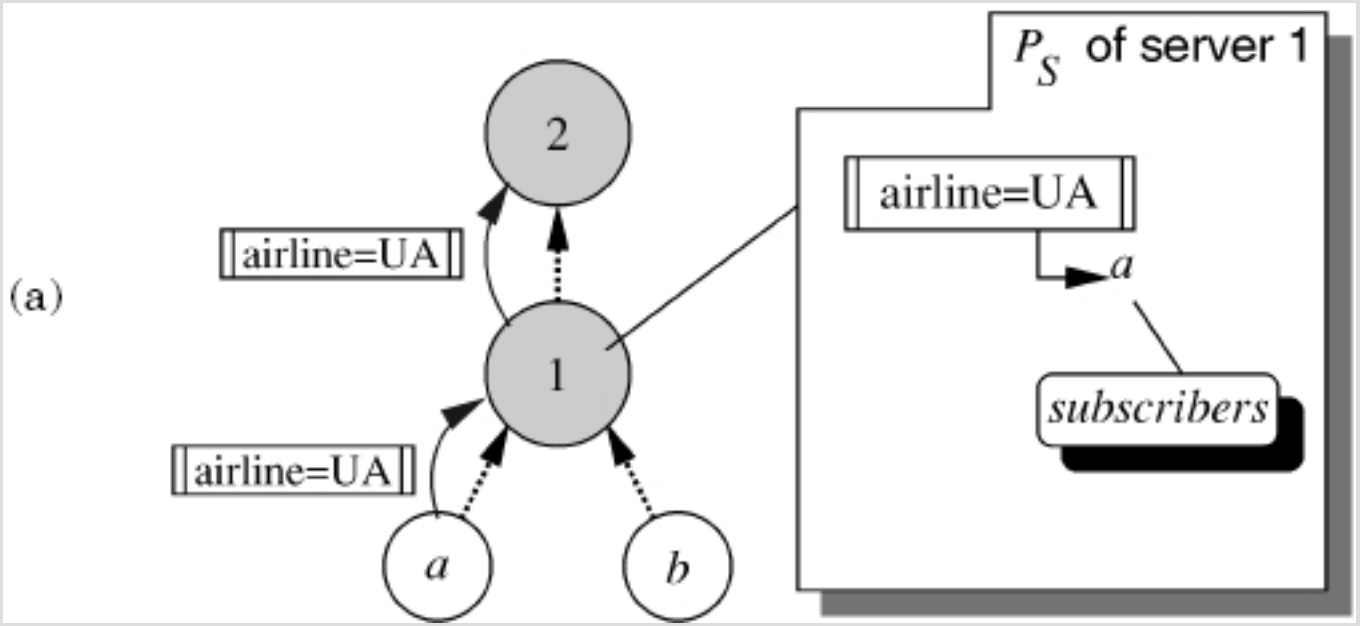
- **Arquitectura Hierárquica:** cada servidor:
  - mantém tem uma variável *master* possivelmente *null.*
  - mantém as subscrições num Ps onde cada subscrição *s* possui um conjunto associado que contém a identificação dos subscritores do filtro, *subscribers(s)*.

Algoritmos e Topologias

- **Subscrição:** após recepção de *subscribe(X,f)*:
  1. Percorrer Ps.
  2. Se encontra um *f'* subscrito por *X* que é mais genérico que *f*, então termina sem qualquer acção.
  3. Senão, se *f* já existe no Ps, então adiciona *X* à lista de subscritores e termina.
  4. Senão, insere *f* no Ps e:
     1. Se *f* é uma subscrição raiz, i.e., nó raiz de Ps, então propaga a subscrição.
     2. Se *f* é mais genérico que um *f''* então elimina *f''*.

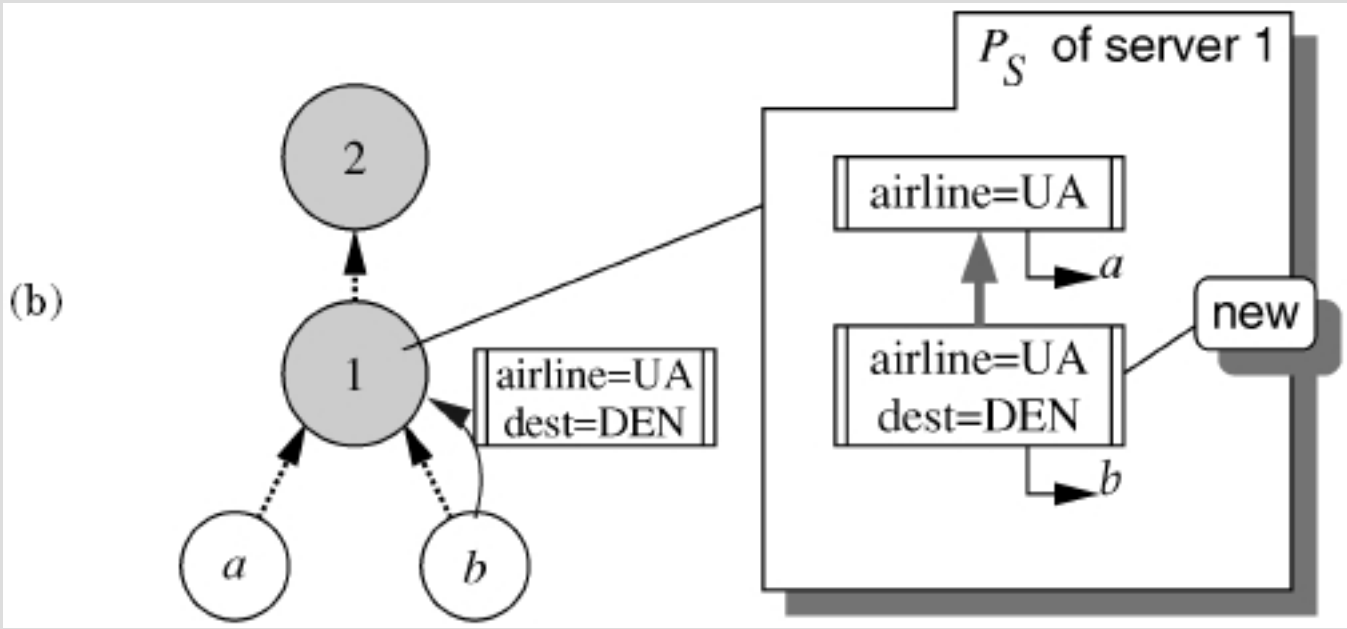
Algoritmos e Topologias

- Exemplo Subscrição:
  - Servidor 1 recebe do cliente *a* a subscrição `[airline=UA]`.
  - Subscrição é inserida como a subscrição raiz e é propagada para o servidor 2.



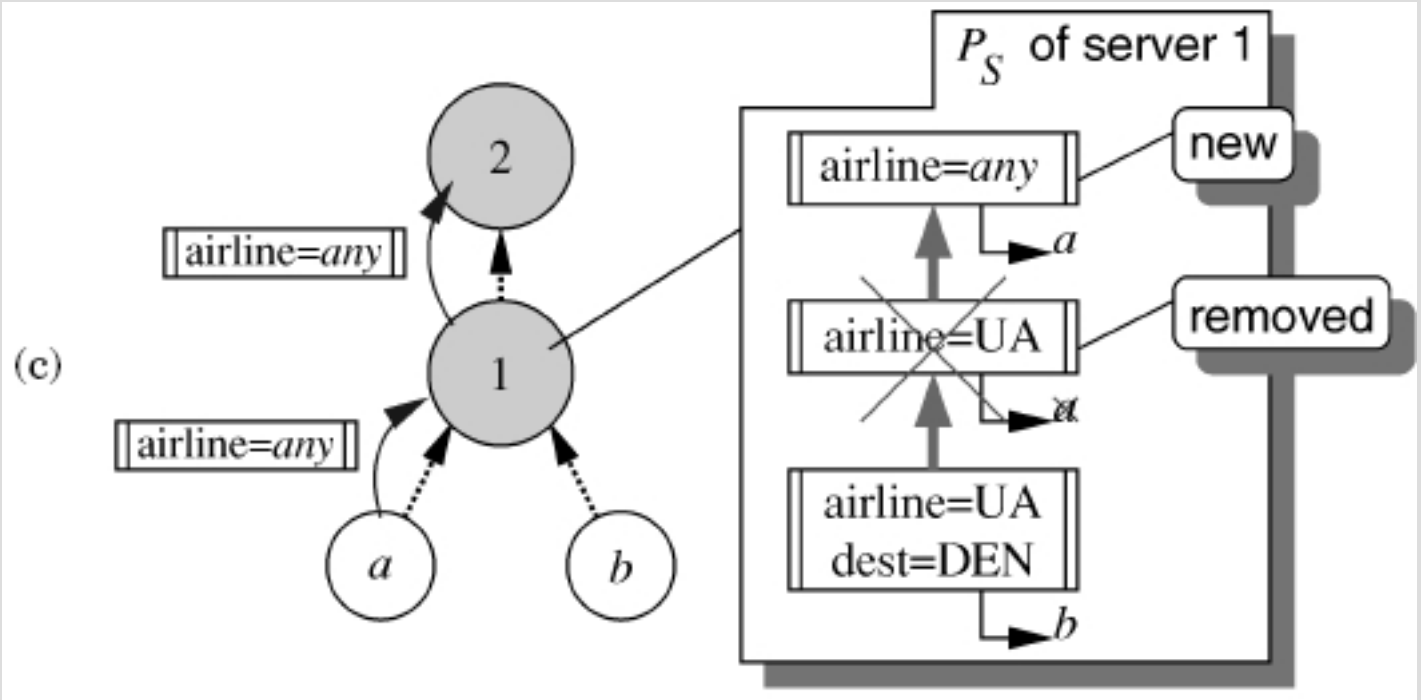
Algoritmos e Topologias

- Exemplo Subscrição:
  - Servidor 1 recebe uma nova subscrição do cliente *b* **`[airline=UA, dest=DEN]`**.
  - Como esta nova subscrição já está coberta, ela não é propagada.



Algoritmos e Topologias

- Exemplo Subscrição:
  - Servidor 1 recebe outra subscrição do cliente *a* **`[airline=any]`** .
  - Como é subscrição raiz, ela é propagada.
  - Como é mais genérica, elimina a anterior.



Algoritmos e Topologias

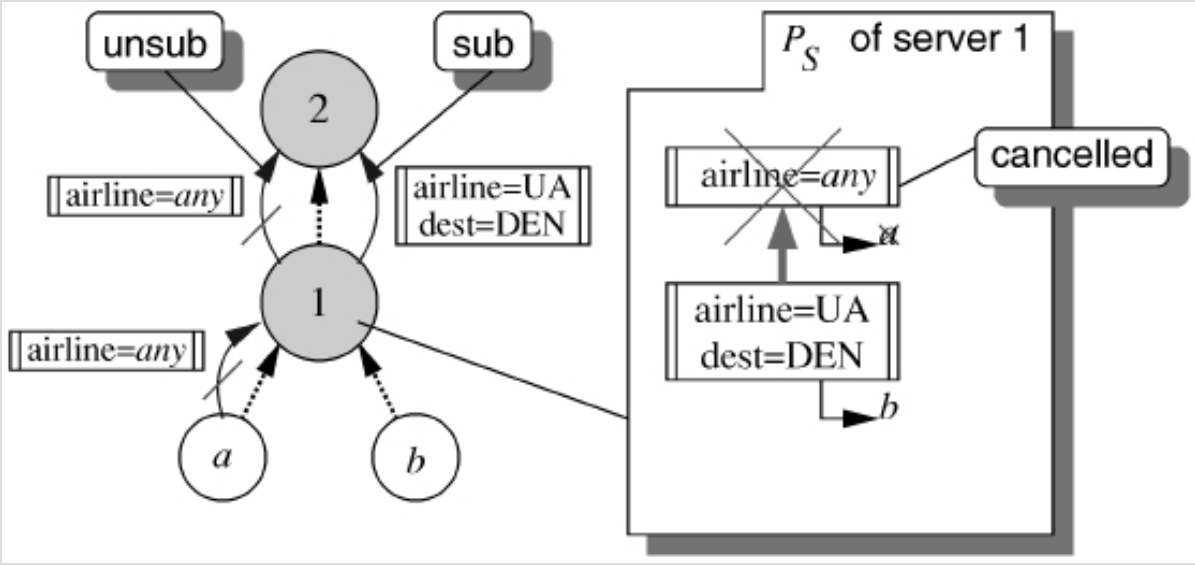
- **Notificação:** após recepção de notificação *n*:
  1. Percorrer Ps em largura procurando todas as subscrições *s* que instanciem com *n*:
     1. Inicializar uma fila *Q* com todas as subscrições raiz.
     2. Iterar sobre cada elemento *s* de *Q*:
        1. Se existe uma subscrição que instancia com a notificação, então adicionar a *Q* todos os predecessores de *s* que ainda não tenham sido visitados.
        2. Caso contrário, remove *s* de *Q*.
     3. No final, *Q* contém todas as subscrições que cobrem *n*.
  2. Enviar uma cópia de *n* a cada subscritor.
  3. Se o servidor *master* não enviou *n*, então é também enviada uma cópia de *n* ao servidor *master*.

Algoritmos e Topologias

- **Cancelamento da Subscrição:** após recepção de *unsubscribe(X,f)*:
  1. Percorrer Ps.
  2. Se encontra *f* subscrito por *X*, remove *X* das subscrições.
  3. Se *f* é uma subscrição raiz:
     1. Propaga o cancelamento da subscrição.
     2. Se existe uma nova subscrição raiz, então propaga a nova subscrição raiz.

Algoritmos e Topologias

- Exemplo Cancelamento Subscrição:
  - Servidor 1 recebe cancelamento da subscrição `[airline=any]` do cliente *a*.
  - Subscrição é removida e, por ser raiz, o cancelamento é propagado para o servidor 2.
  - Como há uma nova subscrição raiz, esta é propagada para o servidor 2.



Algoritmos e Topologias

- **Publicitação:** a técnica de Expedição de Publicitação não se aplica à arquitectura hierárquica uma vez que os servidores *master* nunca respondem aos servidores cliente com subscrições.

Algoritmos e Topologias

- **Arquitecturas P2P com Expedição de Subscrição:** cada servidor:
  - mantém um conjunto *vizinhos* com a identificação dos servidores aos quais está ligado.
  - mantém as subscrições num Ps onde cada subscrição *s* possui:
    - um conjunto associado, *subscribers(s)*, que contém a identificação dos subscritores do filtro.
    - um conjunto associado, *forwards(s)*, que contém o subconjunto dos vizinhos para onde *s* foi propagado.

Algoritmos e Topologias

- **Ligar:** um servidor E1 liga-se a um servidor E2 através de *peer_connect(E2),* se E2 aceitar:
  - E2 envia mensagem de confirmação.
  - Ambos os servidores registam o endereço do outro servidor no seu conjunto de vizinhos.
  - E2 propaga todas as subscrições raiz para E1 e regista E1 no conjunto dos vizinhos para onde as subscrições em causa foram propagadas.
- **Desligar:** um servidor E1 desliga-se de um servidor E2 através de *peer_disconnect(E1)*:
  - E2 remove E1 do seu conjunto de vizinhos.
  - E2 remove E1 de todas as suas subscrições raiz.
  - E2 remove E1 do conjunto de propagação.

Algoritmos e Topologias

- **Subscrição:** idêntica à hierárquica, após recepção de *subscribe(X,f)*:
  1. Percorrer Ps.
  2. Se encontra um *f'* subscrito por *X* que é mais genérico que *f*, então termina sem qualquer acção.
  3. Senão, se *f* já existe no Ps, então adiciona *X* à lista de subscritores.
  4. Senão insere *f* no Ps e:
     1. Adiciona *X* aos subscritores de *f*, *subscribers(f)*,
     2. Propaga a subscrição aos seus vizinhos.

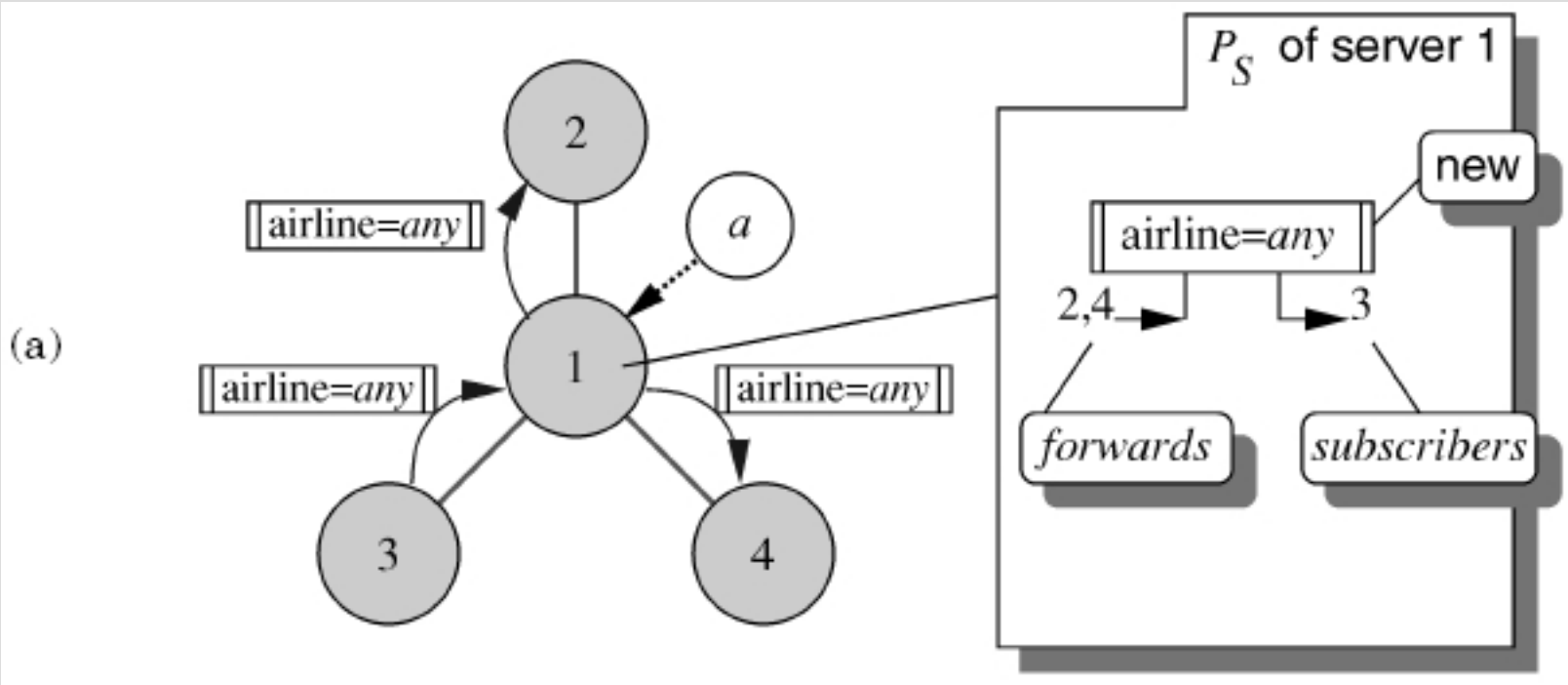
Algoritmos e Topologias

- **Propagação da Subscrição:** efectuada por todos os vizinhos excepto pelos que fazem parte do NST.
- **Not on any Spanning Tree:** NST é o conjunto de nós por onde não faz sentido propagar uma subscrição.

- Sempre que um servidor E1 propaga uma subscrição *f* a um servidor E2:
  - E1 adiciona E2 aos *forwards*(*f*);
  - E1 remove E2 de todos os *forwards* de todas as subscrições que são cobertas por *f*.

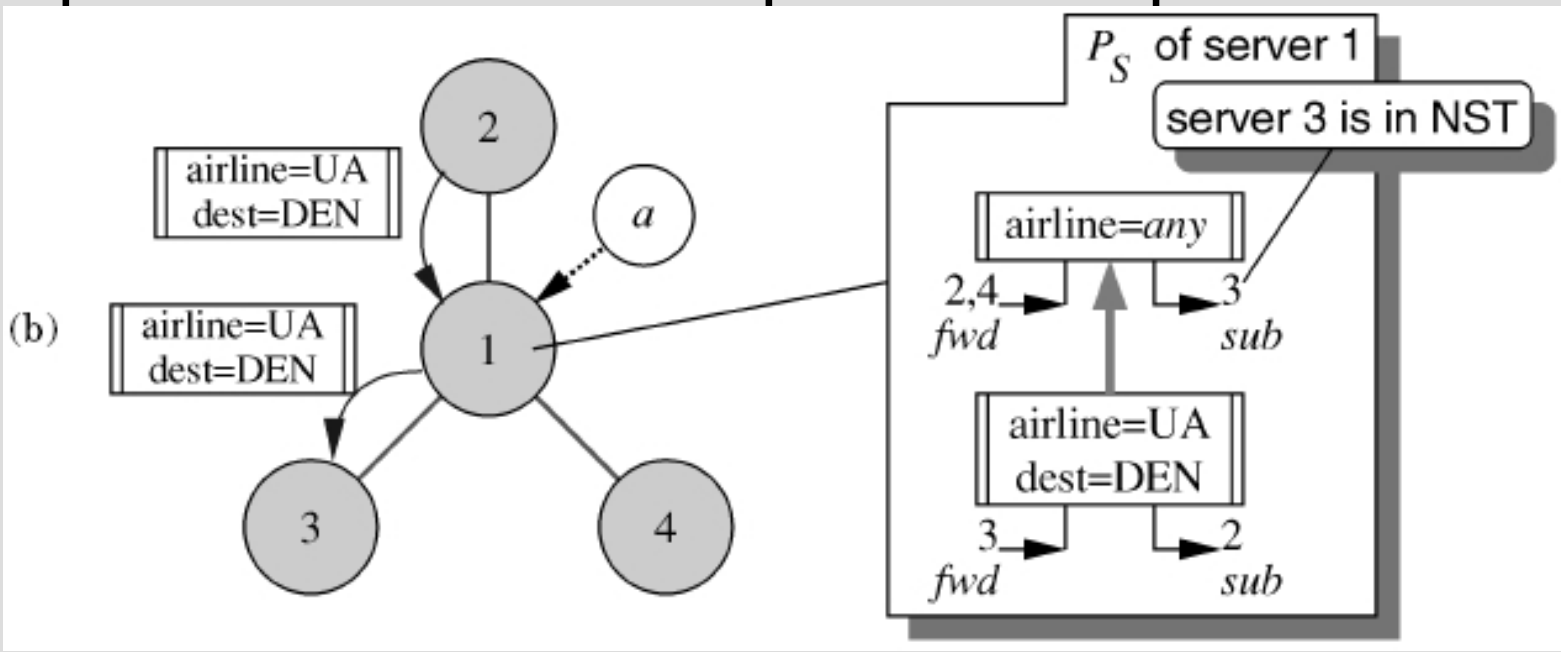
Algoritmos e Topologias

- Exemplo Subscrição:
  - Servidor 3 envia ao 1 subscrição **`[airline=any]`**.
  - Subscrição é inserida como a subscrição raiz.
  - Subscrição propagada para os servidores 2 e 4 mas não para o servidor 3, pois este faz parte do NST.



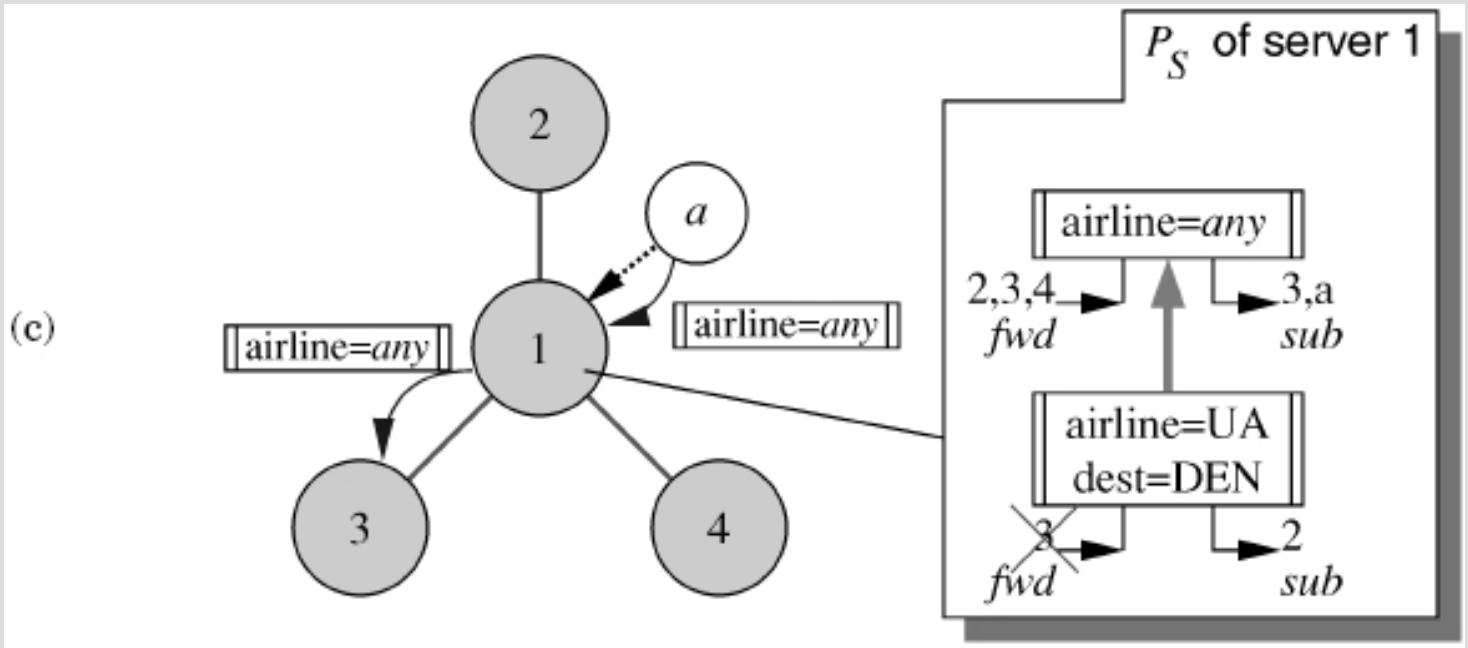
Algoritmos e Topologias

- Exemplo Subscrição:
  - Servidor 2 envia ao servidor 1 a subscrição `[airline=UA, dest=DEN]`.
  - Subscrição é inserida como predecessora.
  - Subscrição propagada unicamente para o servidor 3 por ser este o único que não faz parte dos *forwards*.



Algoritmos e Topologias

- Exemplo Subscrição:
  - Cliente *a* subscreve **`[airline=any]`**.
  - Cliente *a* é inserido como subscritor.
  - NST para a subscrição fica vazio e o servidor 3:
    - Passa a fazer parte dos *forwards* da subscrição.
    - É eliminado dos *forwards* das subscrições cobertas.

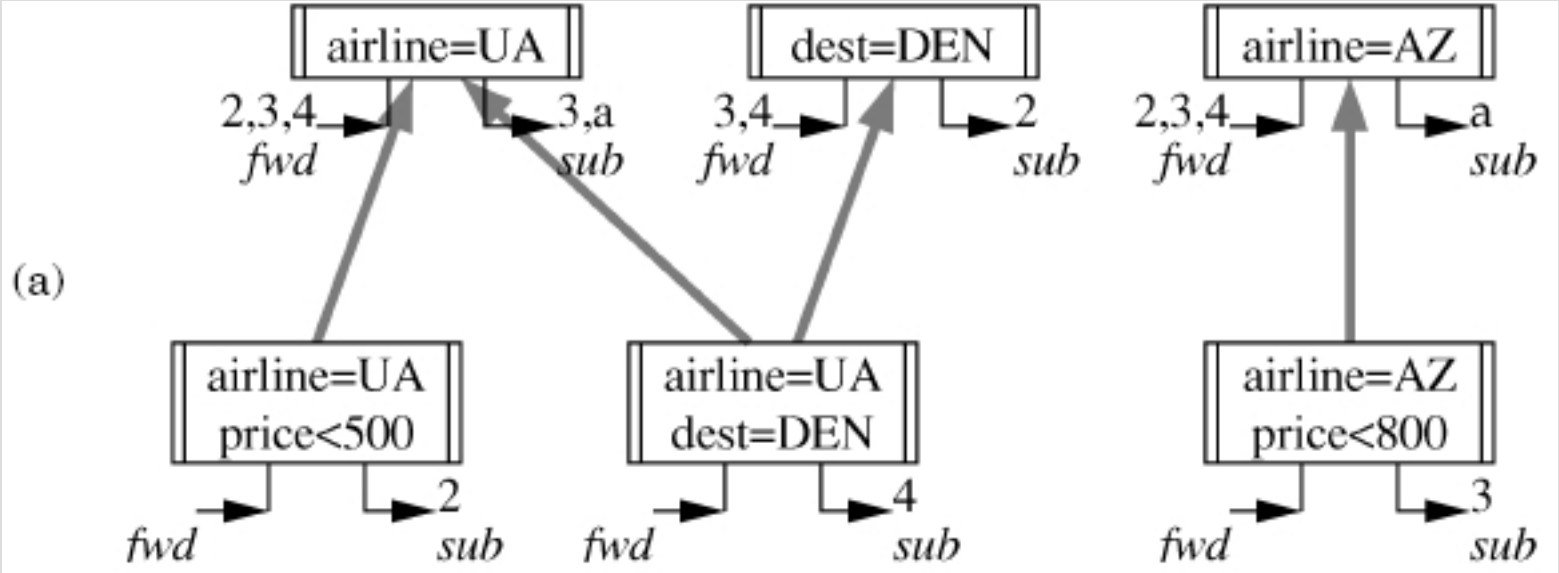


Algoritmos e Topologias

- **Cancelamento da Subscrição:** após recepção de *unsubscribe(X,f)*:
  1. Percorrer Ps.
  2. Se encontra *f* subscrito por *X*, remove *X* das subscrições.
  3. Se após o cancelamento de uma subscrição mais genérica, é necessário (re)propagar as subscrições mais específicas que estavam “bloquedas” pela subscrição mais genérica agora cancelada.

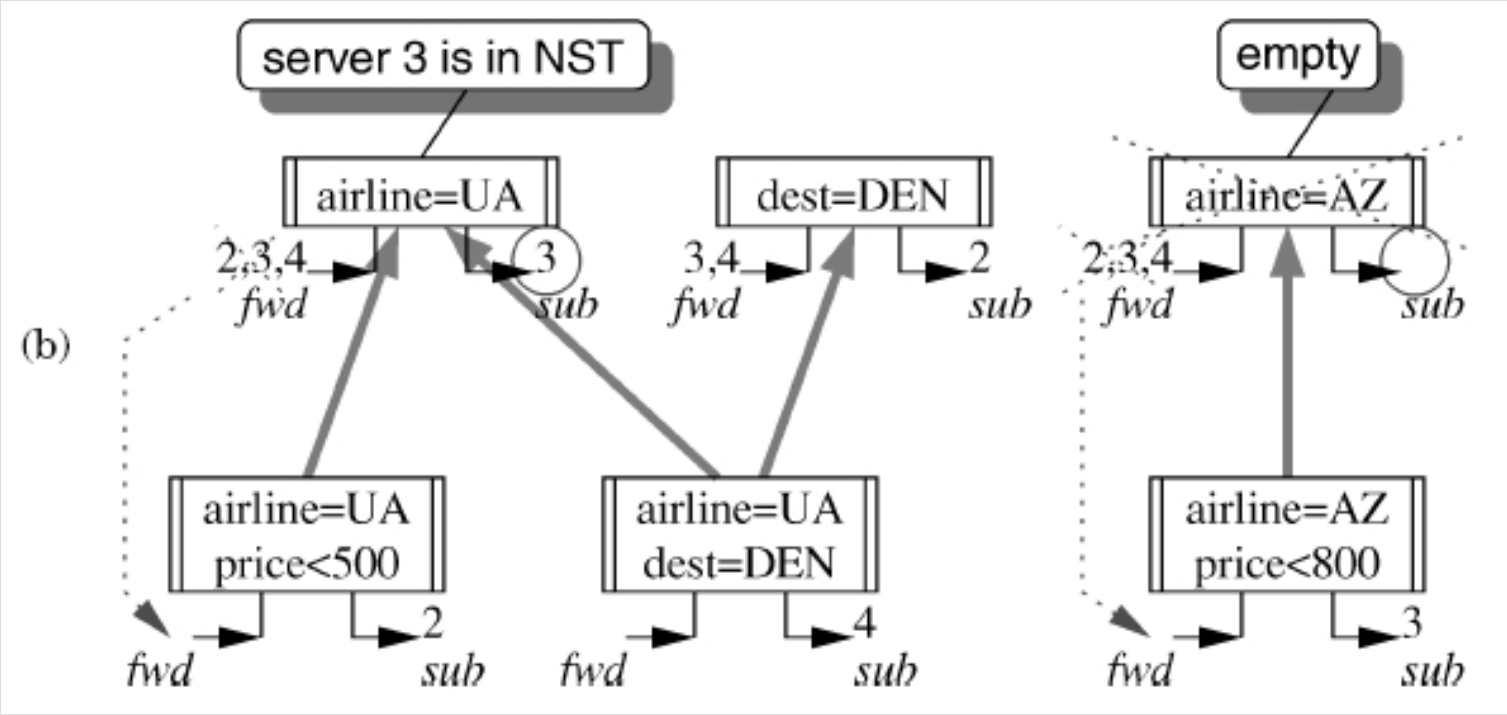
Algoritmos e Topologias

- Exemplo Cancelamento da Subscrição:
  - Ps do servidor 1 após algumas subscrições (partindo do final do exemplo anterior).
  - Estado actual, antes de receber um cancelamento do cliente *a* **`[airline=any]`**.



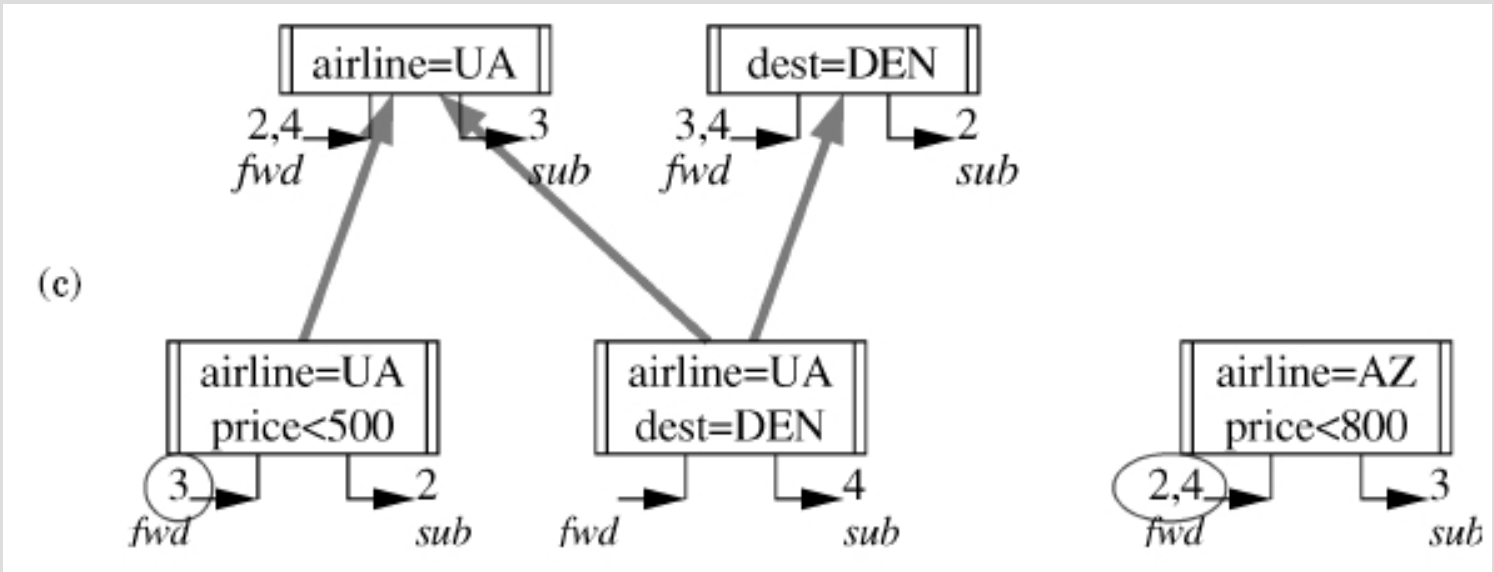
Algoritmos e Topologias

- Exemplo Cancelamento da Subscrição:
  - Cliente *a* envia cancelamento [airline = *any*].
  - Cliente *a* removido das subscrições cobertas pelo filtro especificado.
  - Conjunto NST passa de vazio a conter servidor 3 e servidor 1 propaga o cancelamento da subscrição aos servidores vizinhos contidos no NST.
  - Subscrição `[airline = AZ]` fica vazia e servidor 1 propaga o cancelamento aos *forwards*.



Algoritmos e Topologias

- Exemplo Cancelamento da Subscrição:
  - Ps do servidor 1 após o cancelamento.
  - Subscrição **`[airline=UA, price<500]`** tem de ser propagada para o servidor 3 pois o seu sucessor não foi propagado para o servidor 3.
  - Subscrição **`[airline=UA, price<800]`** tornou-se raiz e tem de ser propagada aos vizinhos excepto NST.



Algoritmos e Topologias

- **Notificação:** análogo ao da arquitectura hierárquica.

Algoritmos e Topologias

- **Arquitecturas P2P com Expedição de Publicitação:** análogo à Expedição de Subscrição com as seguintes diferenças:
  - Existem dois *Posets*, um para as subscrições, P_S, e um para as publicitações, P_A.
  - A propagação de subscrições está limitada ao uso do P_A, que define o caminho das propagações das subscrições.

Algoritmos e Topologias

- **Instanciação de Padrões:** para a instanciação de padrões, os servidores constroem sequências de notificações a partir de sub-sequências de notificações individuais.

- Devido à publicitação, cada servidor sabe quais as notificações e sub-padrões que podem ser enviados a partir de cada vizinho, razão pela qual esta técnica requer uma semântica baseada em publicitação.

Algoritmos e Topologias

- **Tabela de Padrões:** cada servidor possui uma tabela T_P de padrões disponíveis. Esta tabela é simplesmente P_A expandida com os padrões que o servidor já processou.
- Cada padrão *p* tem associado um conjunto de identificadores *providers(p)* que contém os servidores onde *p* está disponível.

	Pattern	Providers
a_1	*string* alarm = "failed-login" *integer* attempts > 0	2
a_2	*string* file = *any* *string* operation = "file-change"	2, 3

Algoritmos e Topologias

- **Factorização de Padrões:** processo pelo qual o servidor decompõe a subscrição.
- Exemplo: servidor recebe por ordem:
  1. `[alarm=failed-login, attempts=1]`,
  2. `[alarm=failed-login, attempts=2]`,
  3. `[file=/etc/passwd, operation=file-change]`
- Após a factorização, usando a tabela de padrões anterior, conclui-se a seguinte sequência: $a1 \cdot a1 \cdot a2$

	Pattern	Providers
a_1	*string* alarm = "failed-login" *integer* attempts > 0	2
a_2	*string* file = *any* *string* operation = "file-change"	2, 3

Requested	Available	
string alarm = "failed-login" *integer* attempts = 1	*string* alarm = "failed-login" *integer* attempts > 0	(a_1)
string alarm = "failed-login" *integer* attempts = 2	*string* alarm = "failed-login" *integer* attempts > 0	(a_1)
string file = "/etc/passwd" *string* operation = "file-change"	*string* file = *any* *string* operation = "file-change"	(a_2)

Algoritmos e Topologias

- **Delegação de Padrões:** a partir dos componentes elementares, o servidor tem de:
  - Enviar as subscrições necessárias para recolher os sub-padrões necessários.
  - Montar um monitor para receber as notificações de instanciação dos sub-padrões.
- O servidor tenta agrupar os elementos em subscrições compostas que possam ser delegadas noutros servidores.
- A escolha dos sub-padrões para delegação segue critérios intuitivos, e.g., apenas padrões contíguos disponíveis de uma única fonte podem ser agrupados e delegados nessa fonte.

Algoritmos e Topologias

- Exemplo Delegação de Padrões:
  - Servidor 1 recebe subscrição $f \cdot g \cdot h$
  - Servidor 1 delega $f \cdot g$ no servidor 2
  - Servidor 1 delega h usando uma subscrição simples
  - Servidor 1 monitoriza $(f \cdot g) \cdot h$

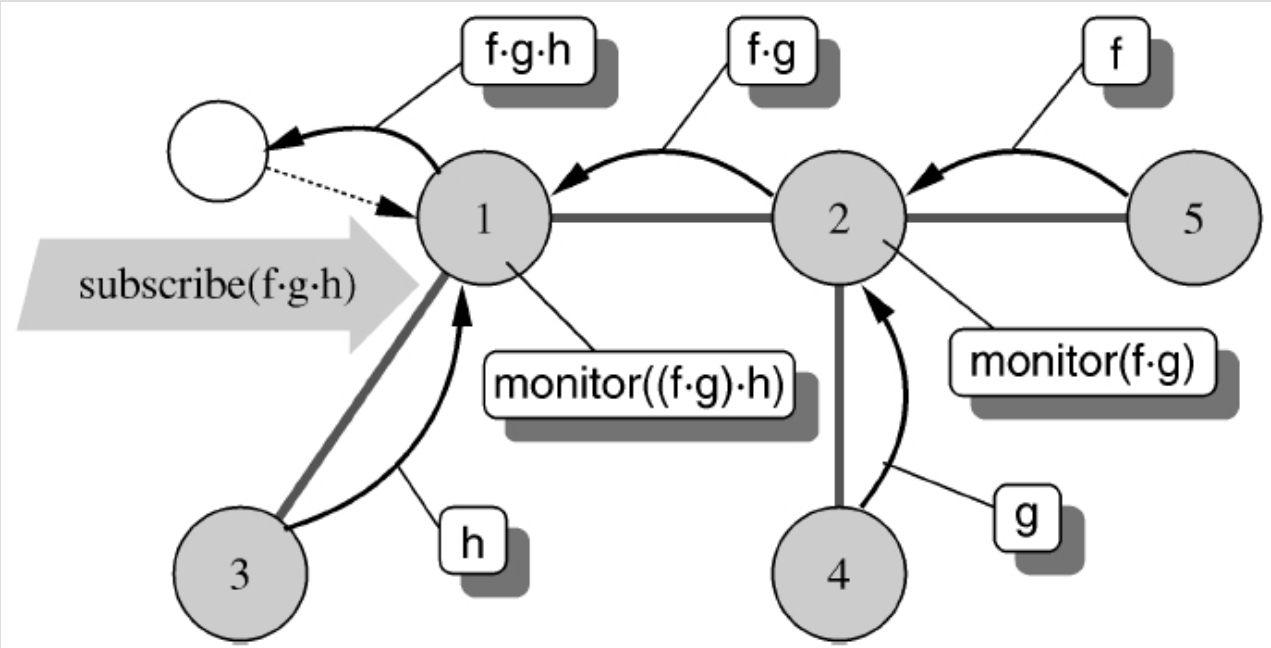


Fig. 17. Pattern monitoring and delegation.

Avaliação

- Arquitectura hierárquica possui um custo de subscrição mais baixo que a arquitectura P2P acíclica.
- O custo de subscrição é amortizado com o aumento da densidade de objectos interessados.
- O custo de entrega de notificações é semelhante.

Trabalho Futuro

- Expansão do desenho das interfaces e dos algoritmos para suportar mobilidade de clientes.
- Implementação da Expedição da Publicitação no protótipo de forma a aplicar as optimizações de instanciação de padrões.
- Modelo de comunicação segura e tolerante a falhas.

Conclusão

- Arquitectura Hierárquica é adequada para baixas densidades de clientes que efectuam subscrições e cancelamentos frequentes.
- Arquitectura P2P obtém melhor performance quando o custo total de comunicação é dominado pelas notificações e quando existe um grande número de notificações ignoradas.

Bibliografia

- Antonio Carzaniga, David Rosenblum, Alexander Wolf. “Design and Evaluation of a Wide-Area Event Notification Service”. ACM Transactions on Computer Systems,Vol. 19, No. 3, August 2001.
- Alonso, Casati, Kuno, Machiraju. “Web Services: Concepts, Architectures and Applications”, Springer, 2004.