technology
from seed

Benefícios da Aleatorização: Consenso na Presença de Falhas Dinâmicas

Miguel Pupo Correia
Trabalho conjunto com Henrique Moniz (Microsoft Research UK), Nuno Neves e Paulo Veríssimo (Universidade de Lisboa, LaSIGE)
WTF – Maio 2011

Motivação: consenso

technology
from seed

- Consenso distribuído:
  - *n* processos
  - Cada um propõe um valor
  - Têm de chegar a acordo sobre um dos valores
- Problema importante em sistemas distribuídos (e paralelos)
  - Há outros problemas úteis "semelhantes" logo:
  - Resultados teóricos sobre o consenso (p.ex., impossibilidades) aplicam-se também a esses problemas
  - Soluções para consenso podem ser usadas para resolver esses problemas
- Exemplo: replicação activa exige difusão com ordem total que é equivalente ao consenso (em vários modelos de sistema)

Motivação: consenso aleatório

technology from seed

- 7 anos de pesquisa, logo várias motivações
- Tolerância a intrusões (ou tol. a faltas bizantinas)
  - Tol. int.: aplicar tolerância a faltas em segurança
  - Consenso com detectores de falhas ou sincronia parcial é vulnerável a ataques contra tempo, logo usar aleatoriedade para não ter esse problema
- Consenso aleatório era considerado ineficiente - verdade?
  - Número de passos de comunicação exponencial ou uso de cripto de chave pública (com faltas bizantinas)
- Consenso em redes sem fios
  - Algoritmos aleatórios ineficientes – conseguimos fazer melhor?

Agenda

technology from seed

- Consenso aleatório
- De consenso binário a difusão com ordem total
- A biblioteca RITAS
- Consenso binário: moeda local vs partilhada
- Consenso aleatório em redes sem fios
- Consenso aleatório em redes sem fios com falhas dinâmicas
- Consenso com falhas dinâmicas e falhas bizantinas
- Conclusões e trabalho futuro

Consenso aleatório

Alguma terminologia

- Algoritmos distribuídos executados por um conjunto de processos P, com $|P| = n$
- Processos comunicam por passagem de mensagens
- Modelo temporal assíncrono
  - Não há tempos máximos para processamento/comunicação
- Modelo de faltas (para já):
  - Bizantino: até *f* processos podem-se desviar arbitrariamente do algoritmo, p.ex., enviando mensagens falsas ou não enviando algumas mensagens
- Processos correctos/incorrectos conforme exibam ou não faltas durante uma execução do algoritmo

Qual a necessidade de aleatoriedade?

technology from seed

- Impossibilidade de Fischer-Lynch-Paterson (FLP)
  - Num sistema assíncrono é impossível resolver consenso de forma determinística se um processo puder falhar por paragem
    - Impossibility of distributed consensus with one faulty process. Journal of the ACM, Apr. 1985
- Como "escapar" deste resultado?
  - Adicionar tempo ao modelo: sincronia parcial
  - Enriquecer o modelo com um oráculo: detectores de falhas
  - Sacrificar o determinismo: **algoritmos aleatórios**
  - (e várias outras)
- Ideia básica dos algoritmos aleatórios: nalguns passos é preciso fazer algo aleatoriamente ("moeda ao ar")

Definição de consenso vs. aleatorização

technology from seed

- Consenso é definido em termos de 3 propriedades:
  - safety:
    - *Validity. If all correct processes propose the same value v, then any correct process that decides, decides v.*
    - *Agreement. No two correct processes decide differently.*
  - liveness:
    - *Termination. Every correct process eventually decides.*
- Na definição de consenso aleatório uma das propriedades tem de ser probabilista, geralmente a Terminação:
  - *Termination. Every correct process eventually decides with probability 1-ζ.*
- Porquê geralmente a terminação?
  - Geralmente prefere-se garantir a *safety* (algoritmos indulgentes)
  - Em cada ciclo há uma probabilidade de terminar, mas o algoritmo pode rodar até terminar realmente (probabilidade = 1)

Dois tipos de algoritmos de consenso aleatório

- Todos os algoritmos de consenso bizantinos (i.e., tolerantes a faltas bizantinas) são binários (valores usados 0 ou 1)
  - Em outros modelos, p.ex. com detectores de falhas ou sincronia parcial, o algoritmo elementar é o consenso multi-valorado (valores pertencem a um domínio qualquer V)
  - Porquê só binário? Mais tarde
- Em 1983 surgiram os 2 primeiros algoritmos que definiram duas classes fundamentais:
  - Moeda local (Ben-Or): cada processo gera os seus próprios números aleatórios
    - Another Advantage of Free Choice: Completely Asynchronous Agreement Protocols, PODC 83
  - Moeda partilhada (Rabin): os processos geram cooperativamente números aleatórios partilhados
    - Randomized Byzantine Generals, FOCS 83

Consenso aleatório binário de Bracha (1)

- O primeiro consenso aleatório binário óptimo em termos de número de processos é de Bracha 1984: $n > 3f$
  - O de Ben-Or precisava de $n > 5f$ e o outro ainda mais
  - An asynchronous |_(n - 1)/3_|-resilient consensus protocol, PODC 84
- Usa “moeda local”
- Algoritmo usa como módulo um algoritmo de difusão fiável
  - Quando uma mensagem é difundida, todos os processos correctos entregam a mensagem ou nenhum entrega (não entrega só se o remetente não for correcto)
  - Algoritmo: 1) remetente envia para todos os processos, 2) todos os processos enviam para todos, 3) todos enviam para todos
  - 3 passos de comunicação, $O(n^2)$ mensagens

Consenso aleatório binário de Bracha (2)

- Passo: processo envia uma mensagem e espera por *n-f* mensagens de outros tantos processos
  - Pois *f* podem ser incorrectos e não se usa tempo
- Processos só consideram mensagens válidas
  - Uma mensagem é válida se poderia ter sido enviada por um processo correcto
  - Exemplo:
    - No passo *i* processo recebeu *n-f* mensagens *(i,1)*, sendo 1 o valor
    - No passo *i+1* cada processo envia valor que recebeu da maioria ou ⊥
    - No passo *i+1* um processo correcto recebe de outro processo *(i+1,v)*
    - *v=0* – mensagem não é válida pois nenhum processo pode ter recebido maioria de 0s no passo *i* ➔ não é considerada
    - *v=1* – mensagem é válida

Consenso aleatório binário de Bracha (3)

est – estimativa do valor a decidir; inicializ. c/valor proposto p/processo
(dec,v) – valor especial usado para tentar decidir **v**

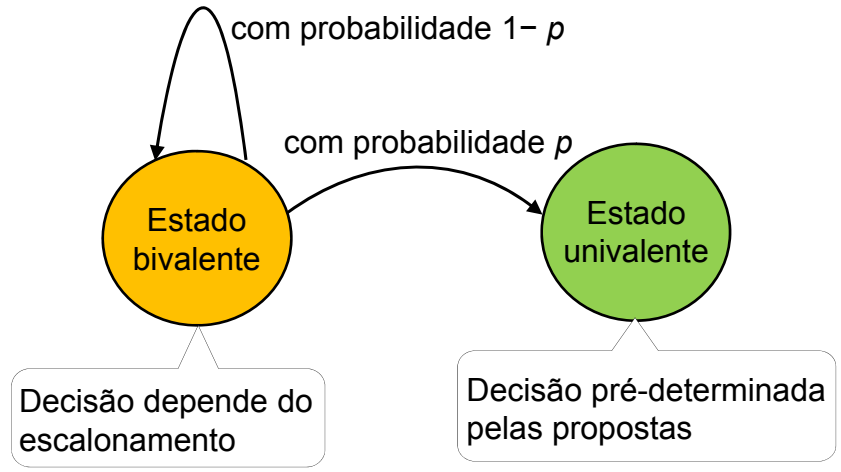
Ciclo k do processo **p**

1. Difundir **(est,v)** e esperar **n-f** mensagens válidas
   est = valor que aparece em mais mensagens
2. Difundir **(est,v)** e esperar **n-f** mensagens válidas
   se mais do que **n/2** mensagens tiverem o mesmo valor **v**,
   então **est = (dec,v)**
3. Difundir **(est,v)** e esperar **n-f** mensagens válidas
   se **n-f** mensagens tiverem **(dec,v)** então decidir **v**
   se pelo menos **n-2f** tiverem **(dec,v)** então **est=v**
   caso contrário **est=1** ou **0** com probabilidade ½
   ir para o passo 1 do ciclo k+1

Se todos os correctos propõem o mesmo **v**, termina nos ciclos 1 ou 2. Ex: **v=0**

Se todos os correctos sorteiam o mesmo **v** no passo 3, é igual

Como a aleatoriedade quebra a impossibilidade FLP



Moeda partilhada vs moeda local

- Algoritmo de Bracha (moeda local); se o número de propostas iguais iniciais não der para termin. rápida:
  - Probabilidade de terminar num ciclo: $1 / 2^{n-f-1}$
  - Número de ciclos esperado para terminação: 2^{n-f}
- Moeda partilhada: acelera terminação fazendo com que todos os processos vejam os mesmos valores da moeda
  - Rabin: um *trusted dealer* distribui inicialmente pedaços dos valores da moeda a cada processo
  - Cachin et al. (ABBA): usam um *dual-threshold coin tossing scheme* baseado no Diffie-Hellman para gerar os valores
    - Termina em 1-2 ciclos! Mas usa cripto chave pública

Porquê consenso aleatório só binário?

technology from seed

- Dois casos:
- Moeda local: se a gama de valores fosse |V|, o número de ciclos esperado para terminação seria $|V|^{n-f}$
  - Exemplo: |V| = 256, n=4, f=1, núm. ciclos esperado: $1{,}7 \times 10^7$
- Moeda partilhada: ninguém fez, mas cripto chave pública é pesada

Desafios

technology from seed

- Interessante sob o ponto de vista teórico, mas consenso binário é inútil na prática
- Algoritmos parecem ineficientes na prática

technology from seed

inesc id lisboa 10

De consenso binário a difusão com ordem total

M. Correia, N.F. Neves, P. Veríssimo. From Consensus to Atomic Broadcast: Time-Free Byzantine-Resistant Protocols without Signatures.
Computer Journal, Jan. 2006

Pilha de protocolos (1)

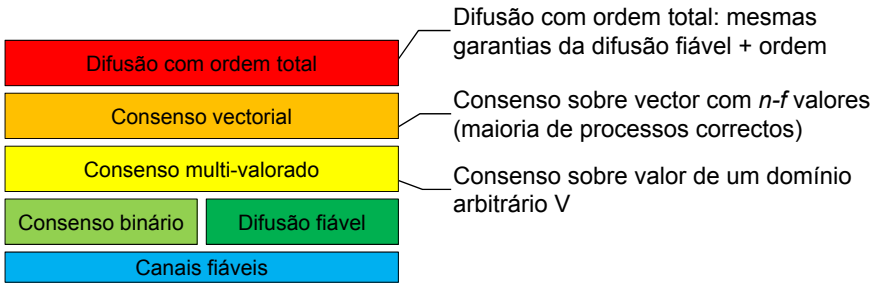
technology from seed

inesc id lisboa 10

- Entre 1999-2003 a minha pesquisa foi sobre consenso, replicação, etc., usando *wormholes*
  - i.e., componentes seguros que fornecem serviços limitados
- Em 2003/04 tentei resolver problemas semelhantes usando aleatorização
- Consenso binário muito limitado, logo definimos um conjunto de transformações / pilha de protocolos

Pilha de protocolos (2)

- Único protocolo realmente aleatório é o consenso binário
- Os restantes são transformações
  - independentes do consenso binário usado



Exemplo: consenso multi-valorado

Processo p

1. Difundir mensagem INIT com a proposta e esperar por **n-f** mensagens INIT
   Se **n-2f** mensagens tiverem o mesmo valor **v**, **w=v**, c.c. **w=⊥**
2. Difundir mensagem VECT com **w** e vector de valores recebidos e esperar por **n-f** mensagens VECT válidas
3. Se **n-2f** tiverem o mesmo **w** e não houver propostas diferentes ***Transformação***
   então *consenso_binário(1)* c.c. *consenso_binário(0)*
   Se decisão for 1, retornar o valor, c.c. retornar ⊥

Utilidade da pilha de protocolos

- Propriedades interessantes para tolerância a intrusões:
- Não uso de assinaturas (baseadas em cripto chave pública)
  - Logo potencial para serem eficientes
- Assincronia
  - Não há dependência de tempo, logo não há vulnerabilidades nisso
- Descentralização
  - Todas as decisões são descentralizadas, não há líder/coordenador
  - Pode ser gargalo de desempenho ou atrasar maliciosamente
- Nº de processos óptimo: *n >= 3f+1*
- Mas é mesmo eficiente? Projecto RITAS

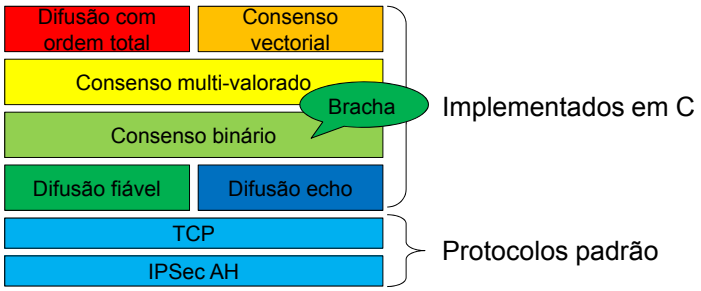
A biblioteca RITAS

H. Moniz and N. F. Neves and M. Correia and P. Veríssimo. Randomized Intrusion-Tolerant Asynchronous Services. DSN 2006.
H. Moniz and N. F. Neves and M. Correia and P. Veríssimo. RITAS: Services for Randomized Intrusion Tolerance. IEEE Transactions on Dependable and Secure Computing, Jan.-Feb. 2011

A biblioteca RITAS (1)

technology from seed

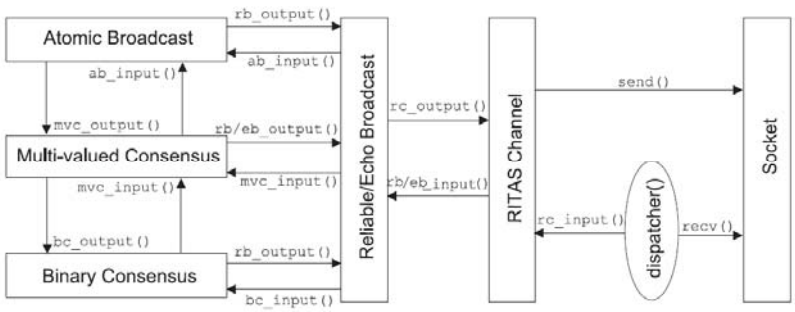
- Implementação de variantes dos protocolos



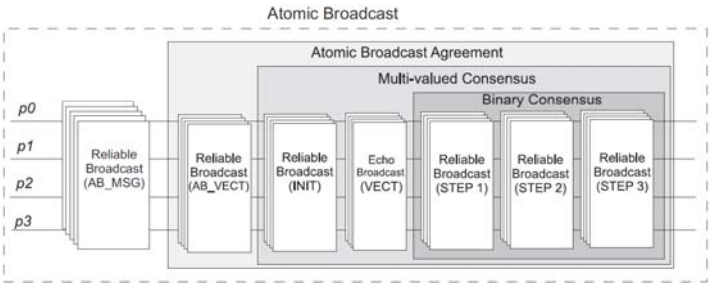
A biblioteca RITAS (2)

technology from seed

- É realmente uma biblioteca, da qual o programador pode instanciar um subconjunto dos protocolos
- Configuração para difusão com ordem total:



Latências numa LAN (execuções individuais)



CPUs Pentium 4 2.8GHz, 2GB ram

1 GB/s Ethernet switch (máx. 67,8 MB/s)

Difusão echo	0,6 ms
Difusão fiável	0,7 ms
Consenso binário	1,2 ms
Consenso multi-valorado	5,0 ms
Consenso vectorial	6,0 ms
Difusão com ordem total	6,5 ms

Valores suficientemente pequenos para servirem muitas aplicações!

Latências numa LAN Efeito do número de processos

	n	w/ IPSec (μs)	relative slowdown
Echo Broadcast	4	584	-
	7	805	38%
	10	1045	79%
Reliable Broadcast	4	667	-
	7	907	36%
	10	1172	76%
Binary Consensus	4	1204	-
	7	3521	192%
	10	7907	557%
Multi-valued Consensus	4	4952	-
	7	13335	169%
	10	25652	418%
Vector Consensus	4	6022	-
	7	16826	179%
	10	32674	443%
Atomic Broadcast	4	6467	-
	7	18496	186%
	10	33474	418%

Latências numa WAN (PlanetLab)

technology from seed

- Características da comunicação entre os 4 nós:

	Latency (ms)	Bandwidth (Kb/s)
Berkeley - Ishikawa	131 (0.26)	1894
Lisbon - Berkeley	210 (1.12)	1167
Berkeley - Campinas	243 (1.37)	990
Lisbon - Campinas	281 (1.24)	845
Lisbon - Ishikawa	322 (1.78)	740
Campinas - Ishikawa	472 (0.85)	165

- Latência dos protocolos:

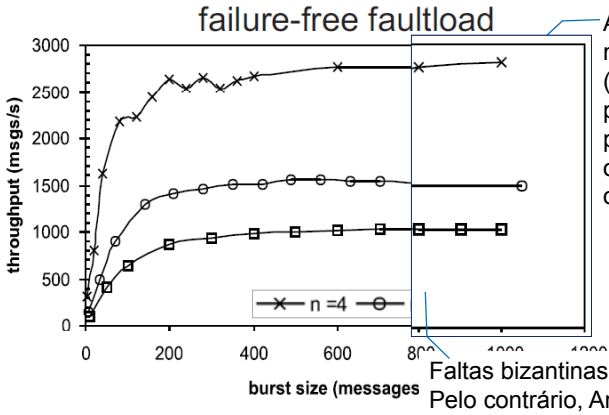
	Latency (ms)
Echo Broadcast	312.62
Reliable Broadcast	486.24
Binary Consensus	535.33
Multi-valued Consensus	2232.30
Vector Consensus	2629.34
Atomic Broadcast	2998.70

Menos do que as duas ligações mais lentas!

6 difusões fiáveis e uma echo

Débito da difusão com ordem total em LAN (mensagens/s)

technology from seed



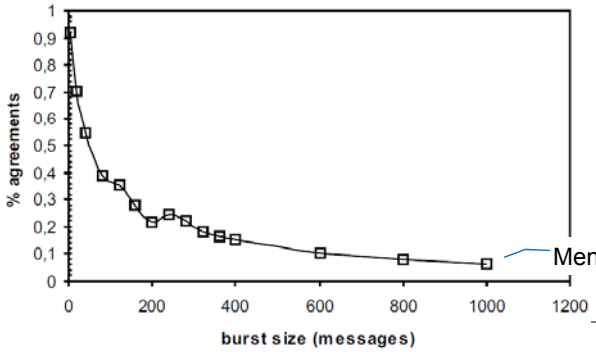
Algoritmos de replicação de máquinas de estados (PBFT, Zyzzyva) parecem muito melhores porque falam de operações/s (várias ops/mensagem)

Faltas bizantinas quase não afectam débito. Pelo contrário, Amir et al. mostraram que líder malicioso no PBFT pode afectar muito o desempenho: 1) ordenando apenas o 1º pedido da fila; 2) manipulando timeouts

Porquê tão eficiente com consenso pesado?

technology from seed

- A % de difusões (fiável/echo) causadas por consensos é baixa com *burst* grandes, porque o nº de ciclos dos consensos é pequeno pois:
  - Os protocolos de nível superior causam que as propostas do consenso binário sejam quase sempre iguais (terminação rápida)
  - Bracha assume que adversário controla escalonamento e consegue ver valores dos outros antes de enviar (irrealista; não foi assim nos experim.)



Menos de 10%

technology from seed

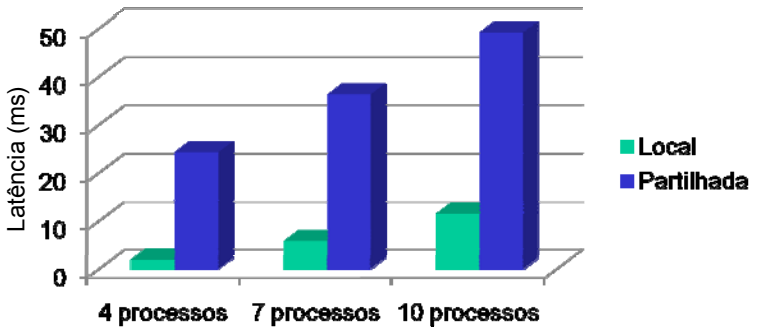
Consenso binário: moeda local vs partilhada

H. Moniz and N. F. Neves and M. Correia and P. Veríssimo. Experimental Comparison of Local and Shared Coin Randomized Consensus Protocols. SRDS 2006

Porquê a comparação?

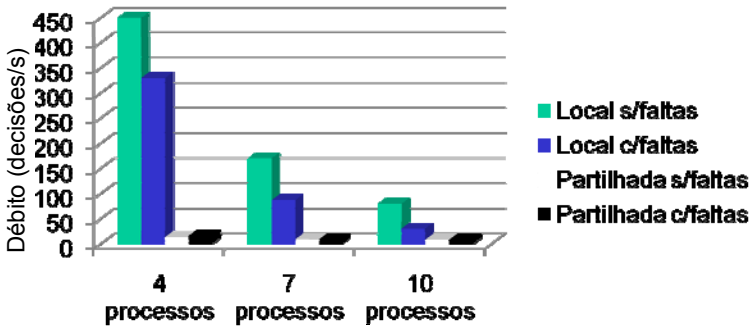
- No estudo anterior usámos o consenso de Bracha com bons resultados, mas teoricamente ABBA é melhor
- Moeda local / Bracha
  - Terminação esperada em 2^{n-f} ciclos
  - $O(n^3)$ mensagens
- Moeda partilhada / ABBA
  - Termina sempre em 1 ou 2 ciclos
  - Mas usa cripto assimétrica ("pesada")
  - $O(n^2)$ mensagens
- Nota: são os 2 melhores de cada categoria e parece difícil fazer melhor em cada uma

Latência em LAN (valores propostos aleatórios)



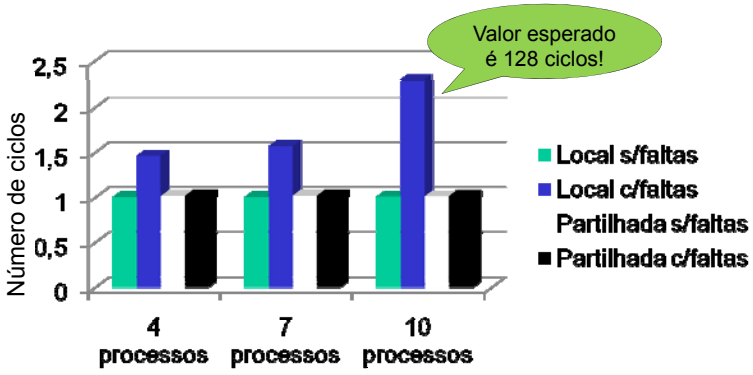
Moeda local sempre melhor (por causa do peso da cripto chave pública)

Débito em LAN (decisões/s)



Moeda local sempre melhor (por causa do peso da cripto chave pública)
Moeda local afectada pelas faltas bizantinas, partilhada não
Moeda partilhada escala melhor com o número de processos

Número de ciclos executados



O número de ciclos é sempre baixo (máx. 2,3)
Moeda local mais afectada por faltas bizantinas

Discussão

- Nos experimentos, o algoritmo de Bracha (moeda local) foi sempre melhor
  - Menor latência, melhor débito
- O ABBA (moeda partilhada) mostrou ser mais escalável
  - Deve ser melhor com mais processos
  - Cripto de chave pública está cada vez mais rápida: novos algoritmos (ESIGN), melhores implementações
  - Cripto de chave pública beneficia de CPUs multi-core

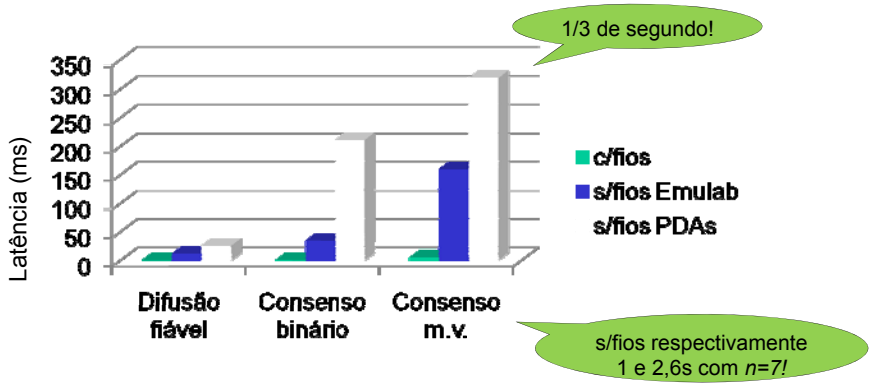
Consenso aleatório em redes sem fios

H. Moniz, N. F. Neves, M. Correia, A. Casimiro, P. Verissimo. Intrusion Tolerance in Wireless Environments: An Experimental Evaluation. PRDC 2007

RITAS em redes sem fios

- O objectivo foi entender como se comportavam os protocolos da biblioteca RITAS em redes sem fios
- Duas testbeds:
  - Emulab – PCs Pentium III 600 MHz, 256MB RAM, 802.11 b/g/a
  - PDAs – HP hw6915, Intel PXA270 416 MHz, 64MB SDRAM, 802.11 b/g
- A maioria dos experimentos foram com consenso binário de Bracha
  - ABBA demasiado lento para PDAs
- Redes ad-hoc e com ponto de acesso

Latência com 802.11b e *n=4*



Latência de ~1 s já não serve para muitas aplicações

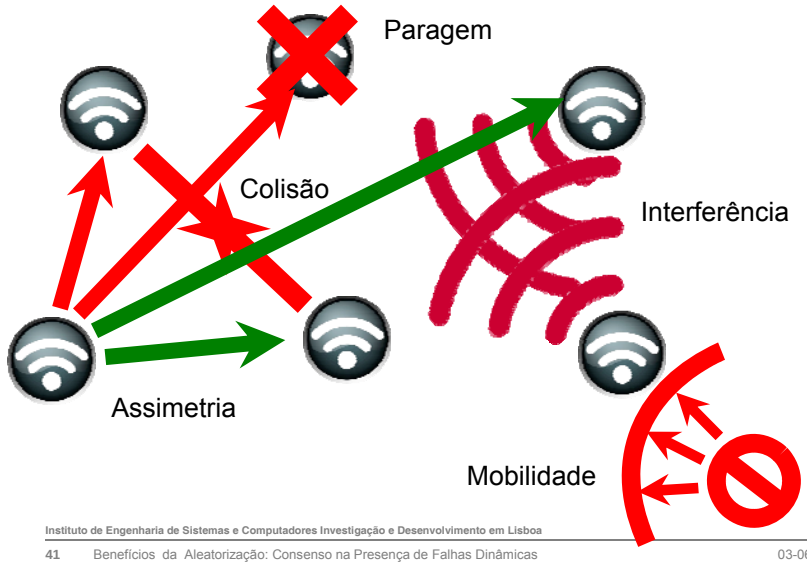
Desafio

- Como fazer consenso aleatório eficiente em redes sem fios? Mudar o modelo?

Consenso aleatório em redes sem fios com falhas dinâmicas

H. Moniz, N. F. Neves, M. Correia, P. Verissimo. Randomization Can Be a Healer: Consensus with Dynamic Omission Failures. DISC 2009 e versão estendida no Distributed Computing J.

Redes ad-hoc – falhas



Redes ad-hoc – comunicação

- A comunicação é naturalmente por difusão
  - Custo de transmitir para um nó ou para todos é igual
  - Mas as faltas são dinâmicas e assimétricas

O problema do algoritmos anteriores seria o modelo?

technology from seed

- Sim, modelo anterior inadequado para estes ambientes
- Canais fiáveis ponto-a-ponto, mas comunicaç. em difusão
  - 1 mensagem sobre um canal ponto-a-ponto custa *n-1* difusões
- Falhas atribuídas a processos quando podem existir muitas falhas nos canais
- Essa inadequação levava ao mau desempenho dos algoritmos que verificámos
  - Sobretudo o excesso de comunicação

Novo modelo

technology from seed

- Nós comunicam por difusão de mensagens (broadcast)
- Podem ocorrer falhas dinâmicas na comunicação - <u>modelo de falhas na comunicação / falhas dinâmicas</u>
  - Quando uma mensagem é difundida, pode ser recebida por nenhum, alguns ou todos os processos
  - O padrão de omissão das mensagens pode variar constantemente
  - Nota: para já só omissões (ainda não considero faltas bizantinas)
- *n* processos
- Ciclos de comunicação síncronos
  - Na realidade não têm de ser síncronos, mas foi assim que provámos a correcção

Resultado de impossibilidade

- O consenso é impossível num sistema síncrono no qual *n-2* faltas de omissão puderem ocorrer em cada passo de comunicação
  - Santoro and Widmayer, Time is not a Healer, 1989
- Resultado muito restritivo: 1 processo parar ou ficar inacessível, corresponde a *n-1* omissões! *n-1 > n-2*
  - De certo modo semelhante ao FLP: se 1 pode parar, consenso impossível
- Este trabalho foi o primeiro a contornar esse resultado
  - Usando aleatorização

k-consenso binário

- Informalmente: *k* de *n* processos decidem um valor 0 ou 1
  - Impossibilidade de Santoro&Widmayer é para este problema com acordo não trivial, ou seja, com $k > \lceil n/2 \rceil$
- Definido em termos de 3 propriedades:
  - *Validity - no process decides a value that wasn't proposed*
  - *Agreement - no two processes decide differently*
  - *Termination - k processes eventually decide with probability 1*

Algoritmo de *k*-consenso binário (1)

technology from seed

- Características
  - Garante a segurança (*safety)* independentemente do nº de omissões
  - Garante o progresso quando o número de omissões é menor que certo limiar
  - Pode terminar em dois passos de comunicação – 2 difusões de mensagens/processo

Algoritmo de *k*-consenso binário (2)

technology from seed

- Cada processo tem 3 variáveis:
- fase – a fase em que se encontra
  - Inicializada a 0
- proposta – a sua estimativa actual do valor a decidir
  - Inicializada com a sua proposta
- status – não-decidido ou decidido
- Em cada ciclo cada processo difunde uma mensagem contendo as 3 variáveis (o seu estado)
- Importante: ciclo ≠ fase
  - Quando há omissões os ciclos podem ir rodando, mas as fases só avançam quando "algo acontece"
  - Uma fase corresponde aprox. a um passo do alg. Bracha

Algoritmo actualização do estado (1)

- Em cada ciclo, um processo actualiza o seu estado em apenas dois casos (e apenas se *status* = *não-decidido)*:
- 1) Se recebe uma mensagem com um valor de fase superior ao seu; nesse caso faz:
  - *fase = fase recebida*
  - *proposta = proposta recebida*
  - *status = status recebido*

  } pois não há faltas bizantinas
- 2) Se recebe mensagens com a fase em que ele está de mais do que *n/2* processos
  - É o que acontece quando não há omissões
  - *fase = fase + 1* mas antes faz o que se explica a seguir

Algoritmo actualização do estado (2)

- Fases pares – tentar pegar um valor
  - Se processo receber o mesmo valor *v* de uma maioria de processos então
  - *proposta = v*
  - c.c. *proposta* = $\perp$ (significa sem preferência)
  - Claramente, se um processo faz *proposta = v* então qualquer outro faz *proposta = v* ou $\perp$
    - Nunca *proposta* = $\neg v$

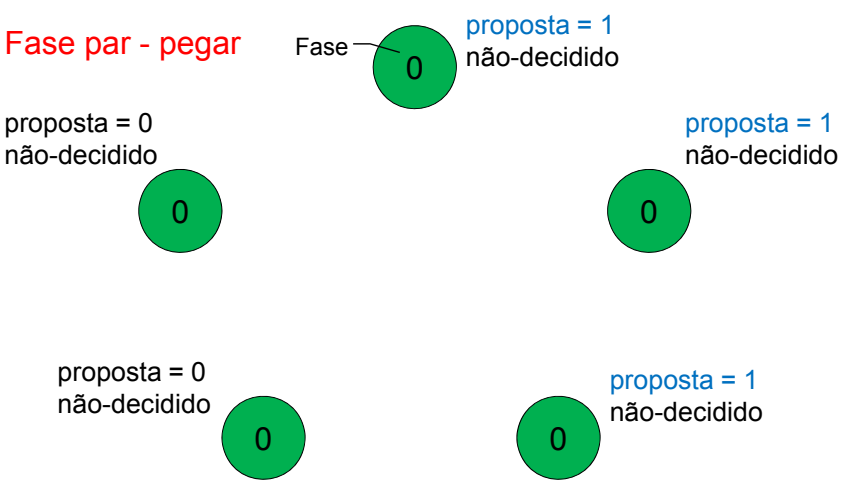
Algoritmo actualização do estado (3)

technology from seed

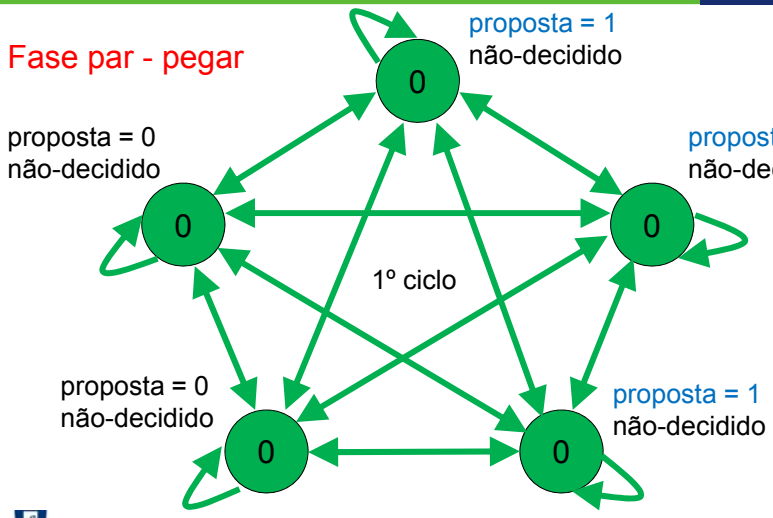
- Fases ímpares – tentar decidir um valor
  - Se o processo receber o valor $v \neq \perp$ de:
  - uma maioria de processos: *proposta = v* e *status = decidido*
  - pelo menos 1 processo: *proposta = v*
  - de nenhum processo: *proposta = 0 ou 1 com probabilidade ½*
  - Se um processo decide, os restantes fazem (pelo menos) *proposta = v* e na fase ímpar seguinte também *status = decidido*
  - Caso contrário, a probabilidade de sair o mesmo valor a todos é $O(2^{-n})$; quando isso acontece, terminam em 2 fases
  - Nota importante: são *fases,* não ciclos

Exemplo de execução sem omissões *(n=5, k=3)*

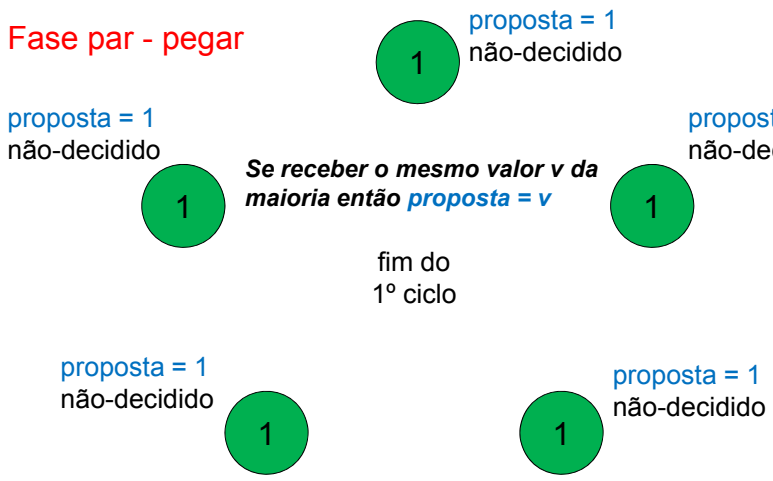
technology from seed



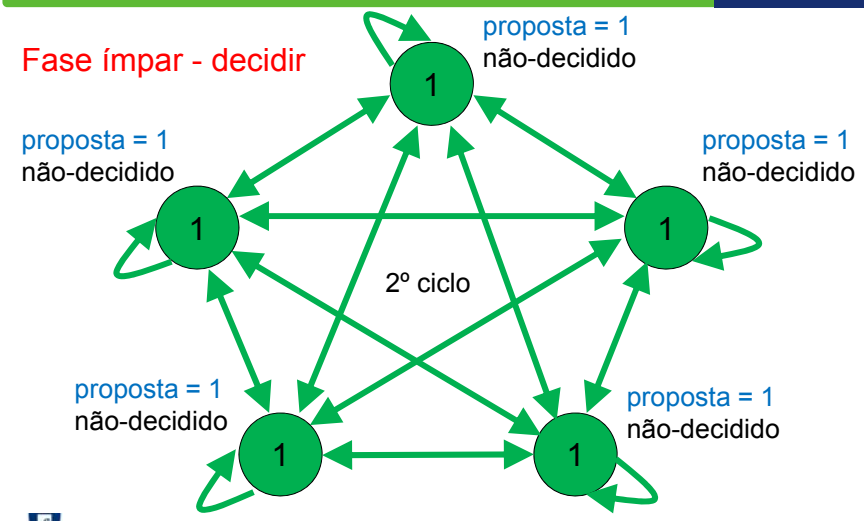
Exemplo de execução sem omissões *(n=5, k=3)*



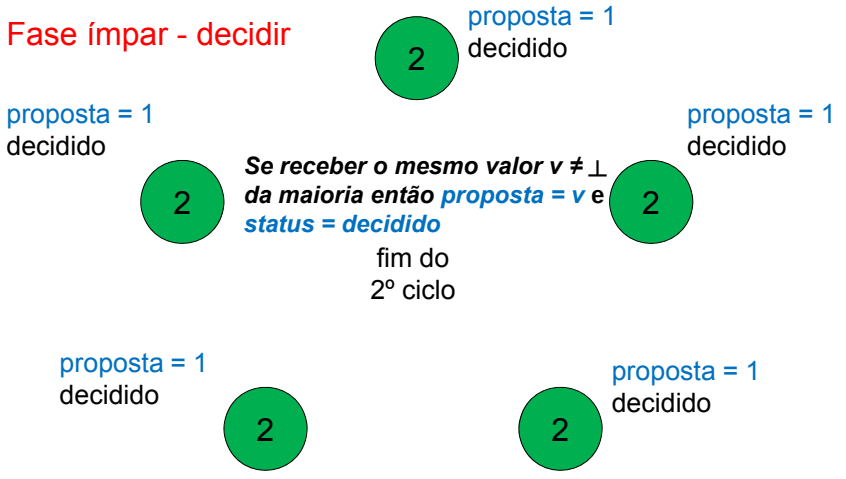
Exemplo de execução sem omissões *(n=5, k=3)*



Exemplo de execução sem omissões *(n=5, k=3)*



Exemplo de execução sem omissões *(n=5, k=3)*



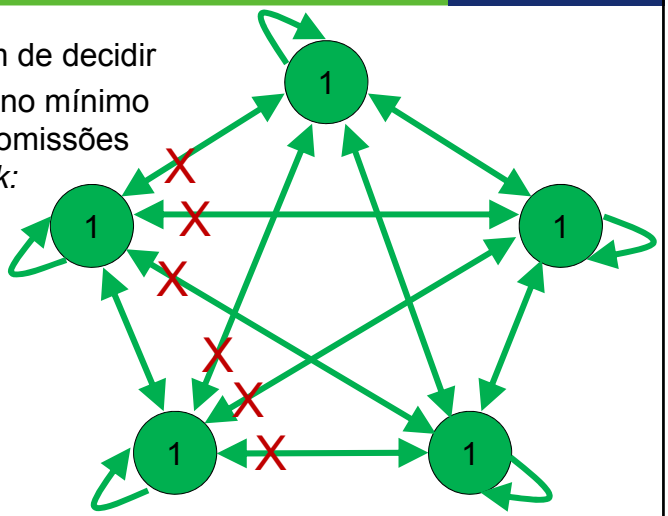
Revisitando as características desejáveis

- Características desejáveis
  - Garante a segurança (*safety)* independentemente do nº de omissões – OK
  - Garante o progresso quando o número de omissões está dentro de certo limite – **Qual é o limite?**
  - Pode terminar em dois passos de comunicação – OK

Quantas omissões podemos tolerar? *(n=5, k=3)*

- Só *k* precisam de decidir
- São precisas no mínimo as seguintes omissões para isolar *n-k:*

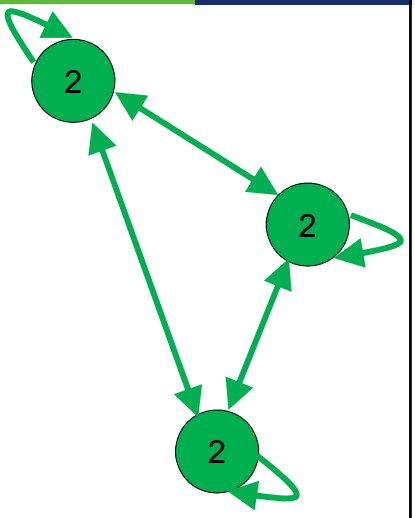
$$\lceil \frac{n}{2} \rceil (n-k)$$



Quantas omissões podemos tolerar? *(n=5, k=3)*

technology from seed

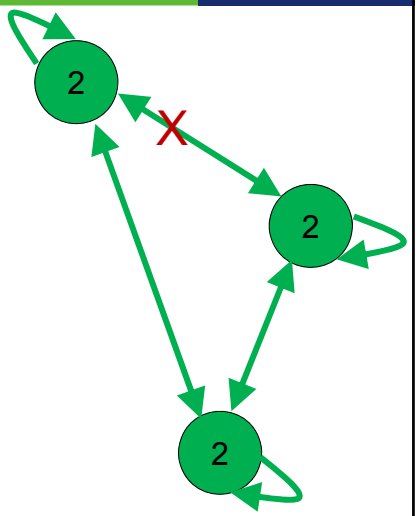
- *k* ainda conseguem decidir



Quantas omissões podemos tolerar? *(n=5, k=3)*

technology from seed

- Ainda pode haver mais *k-2* omissões sem qualquer processo ficar isolado
- Mas mais 1 pode isolar um processo



Quantas omissões podemos tolerar?

technology from seed

- Para garantir progresso podem existir no máximo as seguintes omissões: $\lceil \frac{n}{2} \rceil (n-k) + k - 2$
  - Essa explicação foi com falhas estáticas, mas com dinâmicas é semelhante
  - Com esse número de omissões há sempre pelo menos um processo que incrementa a fase
- O algoritmo tolera mais omissões
  - p.ex., se houver 100% de omissões, os ciclos avançam, mas as fases não; quando deixar de estar bloqueada há progresso
  - Se houver ciclos suficientes com menos omissões do que esse limiar, o algoritmo termina

Prova de correcção do algoritmo

technology from seed

- Apesar de o modelo ser fora do comum, o algoritmo não é muito complicado
- O verdadeiro desafio foi a prova de correcção, sobretudo da *liveness*
  - Provar que quando há $\lceil \frac{n}{2} \rceil (n-k) + k - 2$ omissões o algoritmo faz progresso e eventualmente termina
  - Foi esta a razão para não haver algoritmos neste modelo durante 20 anos?

technology from seed

inesc id lisboa 16

Consenso com falhas dinâmicas e falhas bizantinas

H. Moniz, N. F. Neves, M. Correia, Turquois: Byzantine Consensus in Wireless Ad hoc Networks, DSN 2010

Modelo: diferenças em relação ao anterior

technology from seed

inesc id lisboa 16

- Alguns nós podem ser bizantinos
  - Modelo misto: falhas na comunicação / dinâmicas + falhas nos processos
  - Não optámos pelo modelo puro de Santoro&Widmayer: mensagens podem ser omitidas ou modificadas
- Modelo assíncrono
  - No anterior já o era de certo modo, mas a prova considerava um modelo síncrono
  - A questão é que não significa muito dizer que existem limites temporais para a comunicação+processamento quando o algoritmo tolera omissões na comunicação

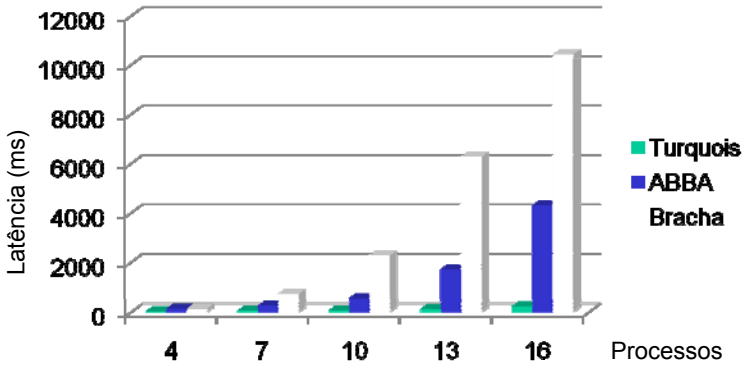
Algoritmo: Turquois

- Semelhante ao anterior mas:
- 3 tipos de fases (e não 2): convergir, trancar, decidir
- Mensagens assinadas usando um esquema de *one-time hash-based message signatures*
  - para evitar cripto chave pública
- Validação de mensagens
  - das assinaturas e semântica (como o Bracha)
- Tolera:
  - Omissões: infinitas
  - Omissões para garantir progresso: $\sigma \leq \lceil \frac{n-t}{2} \rceil (n-k-t)+k-2$
  - Bizantinas: $t < n/3$

Avaliação experimental

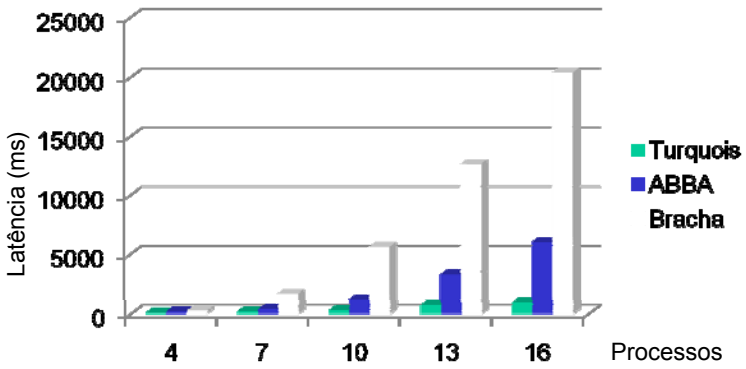
- Objectivo: comparar com Bracha e ABBA (canais fiáveis)
- Ambiente experimental: Emulab
  - Rede ad hoc sem fios 802.11b
  - 16 Pentium III 600 MHz, 256 MB RAM com FC4 Linux, kernel 2.6.18.6
  - Todos os algoritmos implementados em C
- Turquois
  - UDP broadcast
  - Ciclo despoletado cada 10ms ou se fase mudar

Latência sem falhas e propostas divergentes



Turquois 4 a 45 vezes melhor e mais escalável

Latência com falhas bizantinas e propostas divergentes



Turquois 3 a 23 vezes melhor e mais escalável

technology from seed

Conclusões e trabalho futuro

Conclusões

technology from seed

- Os protocolos aleatórios podem ser eficientes em condições realistas, apesar de em teoria não o serem
  - Demonstrámos isso com muitas medidas em diferentes condições
  - Os protocolos convencionais não são eficientes em redes sem fios
- Desenvolvemos 1º algoritmo de consenso baseado num modelo de falhas na comunicação / dinâmicas
  - Tolera faltas por omissão
  - Contornou um resultado de impossibilidade com 20 anos
- Desenvolvemos 1º algoritmo que tolera falhas na comunicação / dinâmicas e nós bizantinos
  - Latência ~1 ordem de grandeza inferior a algoritmos baseados em comunicação ponto-a-ponto

Trabalho futuro / problemas em aberto

- Trabalho com falhas dinâmicas é útil para redes ah-hoc mas está apenas no início: oportunidade!
- De que outro modo se pode contornar a impossibilidade?
- Como resolver problemas mais úteis: consenso multi-valorado, difusão com ordem total? Transformações?
- Qual o desempenho desses algoritmos?
- Que aplicações podem beneficiar destes algoritmos em redes ad-hoc?
- Que algoritmos para redes multi-hop?
- Consenso com o modelo bizantino de Santoro/Widmayer?

Publicidade: doutorado/pós-doc no Instituto Superior Técnico

- IST – a principal escola de engenharia portuguesa – 100 anos
- Na capital, Lisboa, perto de todas as capitais europeias
- Equipe de topo na Europa, participação em projectos europeus

"foi tão bom... que eu não queria sair mais de Lisboa :-) esse é o perigo de fazer doutorado aí... :-D "


HIRED AS AN EFFECTIVE DECISION MAKER;
technology
from seed
inesc id
lisboa
10 anos