ACH 2147 — DESENVOLVIMENTO DE SISTEMAS DE INFORMAÇÃO DISTRIBUÍDOS

NOMES

Daniel Cordeiro

9 e 11 de maio de 2018

Escola de Artes, Ciências e Humanidades | EACH | USP

NOMEAÇÃO DE ENTIDADES

- Nomes, identificadores e endereços
- Resolução de nomes
- Implementação de um espaço de nomes

Essencialmente

Nomes são usados para denotar entidades em um sistema distribuído. Para realizar operações em uma entidade, é preciso ter acesso a ela usando um ponto de acesso. Pontos de acessos são entidades que são identificadas por um endereço.

Nomes “puros”
são nomes que não tem significado próprio; são apenas strings aleatórias. Nomes puros podem ser usados apenas para comparação.

Identificador
Um nome que possui as seguintes propriedades:

1. Cada identificador se refere a, no máximo, uma entidade
2. Cada entidade é referenciada por, no máximo, um identificador
3. Um identificador sempre se refere à mesma entidade (reutilização de identificadores é proibida)

Problema
Dado um nome não estruturado (ex: um identificador), como localizar seu ponto de acesso?

- Solução simples (broadcasting)
- Abordagens baseadas em um local pré-determinado (*home-based*)
- Tabelas de hash distribuídas (P2P estruturado)
- Serviço hierárquico de nomes

Broadcasting

Solução: fazer o *broadcast* do ID, requisitando que a entidade devolva seu endereço

- Não escala para além de redes locais
- Requer que todos os processos escutem e processem os pedidos de localização

Address Resolution Protocol (ARP)

Para encontrar o endereço MAC associado a um endereço IP, faz o *broadcast* da consulta “quem tem esse endereço IP?”

Ponteiros de redirecionamento

Solução: quando uma entidade se mover, ela deixa para trás um ponteiro para sua nova localização

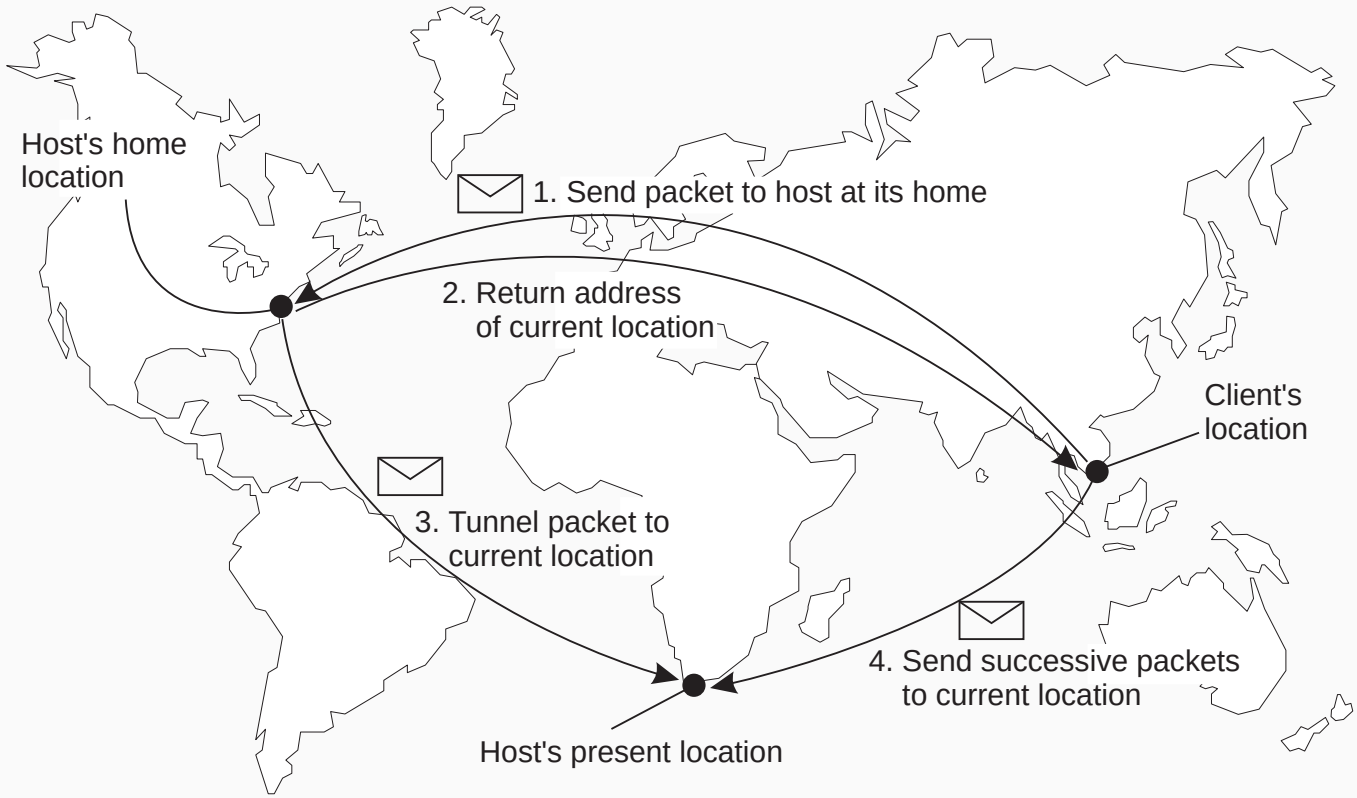
- Dereferenciamento pode ser automático e invisível para o cliente, basta seguir a sequência de ponteiros
- Atualize a referência do ponteiro quando o local atual for encontrado
- Problemas de escalabilidade geográfica (que podem requerer um mecanismo separado para redução da sequência)
  - Sequencias longas não são tolerantes à falhas
  - Maior latência de rede devido ao processo de dereferenciamento

Abordagens baseadas em local pré-determinado

Faça com que um local fixo (sua “casa”) sempre saiba onde a entidade está:

- O endereço pré-determinado é registrado em um serviço de nomes
- O endereço mantém um registro do endereço externo da entidade
- O cliente primeiro contacta o endereço pré-determinado e depois continua para seu endereço externo

Princípio do endereçamento IP móvel:



Abordagens baseadas em local pré-determinado

Abordagem em dois níveis
Mantenha um registro das entidades que foram “visitadas”:

- Verifique o endereço local primeiro
- Se falhar, só então vá para o endereço pré-determinado

Problemas

- O endereço pré-determinado deve existir enquanto a entidade existir
- O endereço é fixo e pode se tornar desnecessário se a entidade se mover permanentemente
- Escalabilidade geográfica ruim (a entidade pode estar do lado do cliente)

Pergunta: como resolver o problema da mudança permanente? (talvez com outro nível de endereçamento (DNS))

Chord

Considere que os nós estejam organizados em forma de anel lógico

- A cada nó é atribuído um identificador aleatório de m-bits
- A cada entidade é atribuído uma única chave de m-bits
- Entidades com chave k estão sob a jurisdição do nó com o menor $id \geq k$ (seu sucessor)

Solução ruim

Faça com que cada nó mantenha o registro de seus vizinhos e faça uma busca linear ao longo do anel

Notação

Chamamos de nó p o nó cujo identificador é p.

Ideia:

- cada nó p mantém um finger table $FT_p[\,]$ com no máximo m entradas

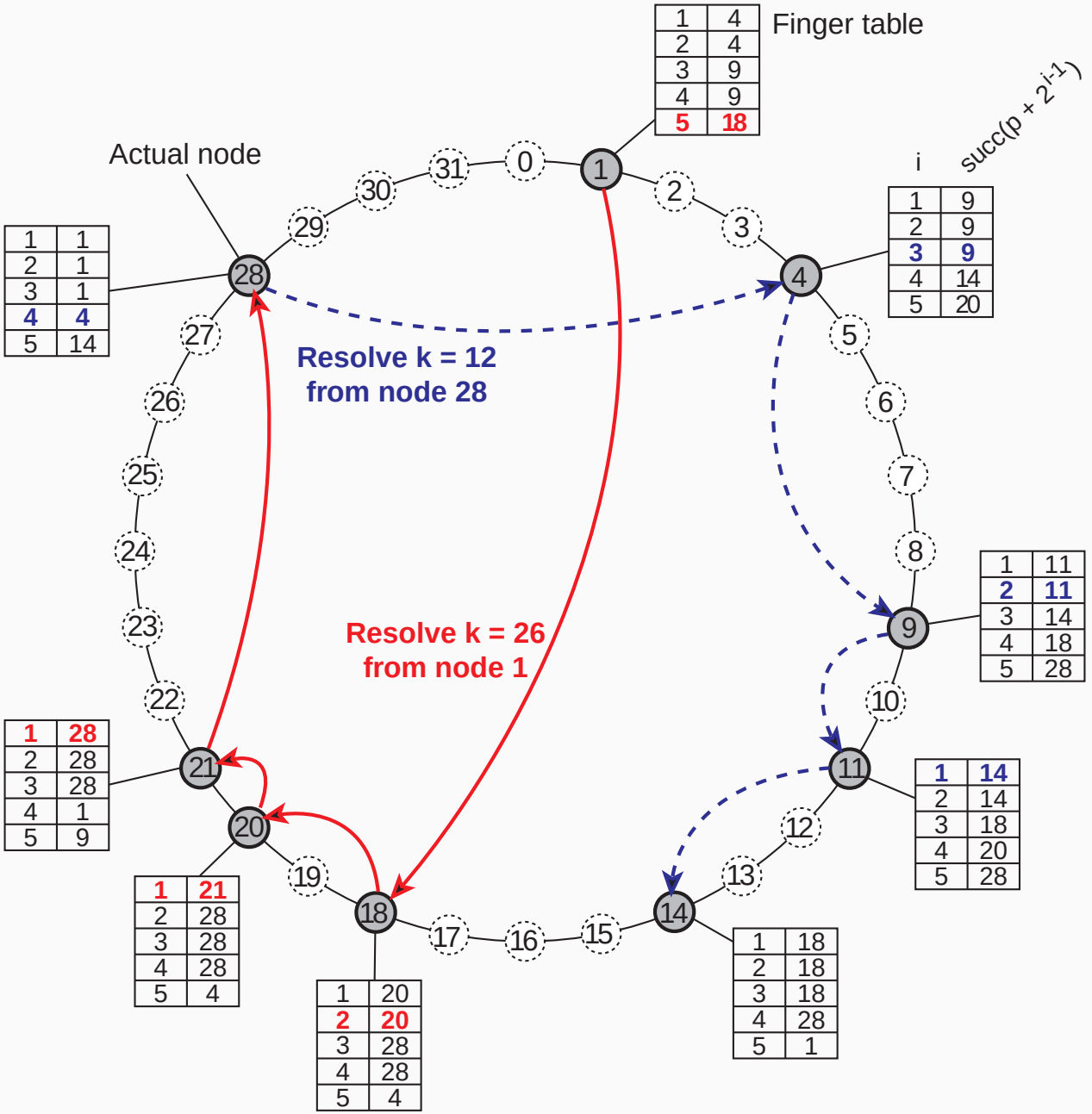
  $$FT_p[i] = succ(p + 2^{i-1})$$

  $FT_p[i]$ aponta para o primeiro nó que sucede p com distância de pelo menos 2^{i-1} posições
- Para procurar por uma chave k, o nó p encaminha o pedido para o nó com índice j tal que:

  $$q = FT_p[j] \leq k < FT_p[j+1]$$

- Se $p < k < FT_p[1]$, a requisição é encaminhada para $FT_p[1]$

DHTS: FINGER TABLES



CHORD EM PYTHON

```
class ChordNode:
  def finger(self, i):
    succ = (self.nodeID + pow(2, i-1)) % self.MAXPROC    # succ(p+2^(i-1))
    lwbi = self.nodeSet.index(self.nodeID)               # self in nodeset
    upbi = (lwbi + 1) % len(self.nodeSet)                # next neighbor
    for k in range(len(self.nodeSet)):                   # process segments
      if self.inbetween(succ, self.nodeSet[lwbi]+1, self.nodeSet[upbi]+1):
        return self.nodeSet[upbi]                        # found successor
      (lwbi,upbi) = (upbi, (upbi+1) % len(self.nodeSet)) # next segment

  def recomputeFingerTable(self):
    self.FT[0]  = self.nodeSet[self.nodeSet.index(self.nodeID)-1] # Pred.
    self.FT[1:] = [self.finger(i) for i in range(1,self.nBits+1)] # Succ.

  def localSuccNode(self, key):
    if self.inbetween(key, self.FT[0]+1, self.nodeID+1): # in (FT[0],self]
      return self.nodeID                                 # responsible node
    elif self.inbetween(key, self.nodeID+1, self.FT[1]): # in (self,FT[1]]
      return self.FT[1]                                  # succ. responsible
    for i in range(1, self.nBits+1):                     # rest of FT
      if self.inbetween(key, self.FT[i], self.FT[(i+1) % self.nBits]):
        return self.FT[i]                                # in [FT[i],FT[i+1])
```

EXPLORAÇÃO DE PROXIMIDADE DA REDE

Problema:
A organização lógica dos nós em uma rede de overlay pode levar à transferência de mensagens de forma errática na Internet: os nós k e $succ(k+1)$ podem estar muito longes um do outro.

Soluções

EXPLORAÇÃO DE PROXIMIDADE DA REDE

Problema:
A organização lógica dos nós em uma rede de overlay pode levar à transferência de mensagens de forma errática na Internet: os nós k e $succ(k+1)$ podem estar muito longes um do outro.

Soluções

- Atribuição ciente da rede: ao atribuir um ID ao nó, assegure-se de que nós próximos no espaço de endereçamento estão próximos na rede real. **Pode ser muito complicado.**

EXPLORAÇÃO DE PROXIMIDADE DA REDE

Problema:
A organização lógica dos nós em uma rede de overlay pode levar à transferência de mensagens de forma errática na Internet: os nós k e $succ(k+1)$ podem estar muito longes um do outro.

Soluções

- Atribuição ciente da rede: ao atribuir um ID ao nó, assegure-se de que nós próximos no espaço de endereçamento estão próximos na rede real. **Pode ser muito complicado.**
- Roteamento de proximidade: mantenha mais de um sucessor possível e encaminhe a mensagem para o mais próximo. Ex: no Chord, $FT_p[i]$ aponta para o primeiro nó em $INT = [p + 2^{i-1}, p + 2^i - 1]$. O nó p pode guardar também ponteiros para outros nós em INT.

EXPLORAÇÃO DE PROXIMIDADE DA REDE

Problema:
A organização lógica dos nós em uma rede de overlay pode levar à transferência de mensagens de forma errática na Internet: os nós k e $succ(k+1)$ podem estar muito longes um do outro.

Soluções

- Atribuição ciente da rede: ao atribuir um ID ao nó, assegure-se de que nós próximos no espaço de endereçamento estão próximos na rede real. **Pode ser muito complicado.**
- Roteamento de proximidade: mantenha mais de um sucessor possível e encaminhe a mensagem para o mais próximo. Ex: no Chord, $FT_p[i]$ aponta para o primeiro nó em $INT = [p + 2^{i-1}, p + 2^i - 1]$. O nó p pode guardar também ponteiros para outros nós em INT.
- Seleção de vizinho por proximidade: quando houver uma escolha para determinar quem será seu vizinho, escolha o mais próximo.