Roteamento Baseado em Vetor de Distâncias e Roteamento Hierárquico

Diego Passos

31 de Maio de 2017

1 Vetor de Distâncias

Ao contrário do roteamento baseado em Estado de Enlaces que utiliza um algoritmo sequencial para encontrar os caminhos mínimos, no roteamento baseado em vetor de distância utiliza-se um algoritmo distribuído para o cálculo das melhores rotas. Isso significa que cada roteador que executa o protocolo de roteamento baseado em vetor de distância realizará uma parte do processamento do algoritmo e, através da troca de mensagens com seus **vizinhos diretos**, roteadores colaborarão para determinar as melhores rotas.

Da mesma maneira que o Algoritmo de Dijkstra é quase sempre usado como o algoritmo de caminhos mínimos pelos protocolos de roteamento baseados em Estado de Enlaces, o **Algoritmo de Bellman-Ford** é o utilizado pelos protocolos baseados em Vetor de Distâncias. Este algoritmo utiliza o paradigma de **programação dinâmica**, no qual as soluções para partes mais simples do problema são pré-computadas e armazenadas para, posteriormente, simplificarem a solução de partes mais complexas. Colocando isso em termos de roteamento, o Algoritmo de Bellman-Ford encontra primeiro as rotas ótimas com poucos saltos e depois as utiliza como base para encontrar as rotas ótimas que possuem mais saltos.

A ideia central do Algoritmo de Bellman-Ford é resumida na chamada Equação de Bellman-Ford:

$$d_x(y) = min_v \left\{ c(x,v) + d_v(y) \right\}, \tag{1}$$

onde $d_x(y)$ e $d_v(y)$ denotam as distâncias dos caminhos mais curtos de x e de v até y e $c(x,v)$ denota o custo do enlace de x para v. Basicamente, o que esta equação diz é que o melhor caminho do nó x para um certo destinatário y é necessariamente composto pelo enlace de x para um de seus vizinhos anexado do melhor caminho deste vizinho para o destinatário y. A partir desta observação, se, de alguma forma, assumirmos que x conhece os custos dos caminhos mais curtos de cada um de seus vizinhos até y, basta que x some a estes valores os pesos dos seus enlaces para cada um destes vizinhos e verifique qual soma resulta no menor valor. Este valor será, certamente, o custo do caminho mínimo de x para y e o próximo salto neste caminho ótimo é o vizinho v de x associado a este custo mínimo.

Repare que este raciocínio não fornece a x a rota completa até y, apenas seu custo e o próximo salto. Estas informações, no entanto, são suficientes para a construção da tabela de roteamento.

A Equação de Bellman-Ford, no entanto, não especifica como um nó x obtém estas distâncias mais curtas de seus vizinhos até y. Se um vizinho v de x soubesse, com certeza, qual é a sua distância mínima até y, ele poderia simplesmente anunciá-la a x através do envio de uma mensagem. Entretanto, como v determina esta distância? Ele poderia utilizar a própria Equação de Bellman-Ford, mas para isso precisaria receber as distâncias mínimas de todos os seus vizinhos até y. Podemos continuar estendendo este raciocínio para outros nós, mas repare que estamos apenas criando uma cadeia informações que rapidamente resulta em uma dependência cíclica (*e.g.*, x precisa da melhor distância de v para y; para calculá-la, v precisa da melhor distância de x para y).

Felizmente, esta dependência cíclica pode ser resolvida através de um processo iterativo no qual cada nó recebe **estimativas** dos custos mínimos de seus vizinhos para outros destinatários e, com base nelas e na Equação de Bellman-Ford, gera suas próprias estimativas. É possível demonstrar que, sob condições que são **razoáveis na prática**, este processo gradualmente converge para que todos os nós da rede encontrem os custos mínimos deles para cada destinatário da rede e o nó de próximo salto na rota ótima associada.

Mais especificamente, o Algoritmo de Bellman-Ford funciona da seguinte forma:

1. **Inicialização**. Cada nó mantém uma estrutura de dados chamada de **Vetor de Distâncias**, na qual o nó armazena o custo mais baixo conhecido dele para cada um dos demais nós da rede — *i.e.*, as entradas do vetor de distâncias mantido pelo nó x serão os valores $D_x(y)$ para todos os

demais nós y da rede. No início da execução do algoritmo, cada nó conhece apenas os pesos dos seus enlaces para seus vizinhos diretos. Por isso, o vetor de distâncias de um nó é inicializado para conter estes pesos como estimativas iniciais das distâncias para seus vizinhos e **infinito** para os demais nós da rede.

2. **Envio do Vetor de Distâncias**. Após gerar esta versão inicial do seu Vetor de Distâncias, um nó o envia para cada um de seus vizinhos diretos.

3. **Recebimento de um Vetor de Distâncias**. Ao receber um Vetor de Distâncias de um vizinho, um nó deve varrer todos os destinatários da rede verificando se é possível melhorar sua estimativa dos custos mínimos considerando o peso de seu enlace para este vizinho e os custos que o vizinho está reportando através do seu vetor de distância. Mais especificamente, se x recebe o Vetor de Distâncias do seu vizinho v, para cada destinatário y, x deve calcular a soma $c(x, v) + d_v(y)$ e compará-la com sua estimativa atual $d_x(y)$. Caso a soma seja mais baixa, x atualiza sua estimativa do seu custo mínimo para y, armazenando que o próximo salto na rota associada é v.

4. **Atualização do Vetor de Distância**. Toda vez que houver uma mudança em pelo menos uma das entradas do seu Vetor de Distâncias, o nó deve reenviá-lo para seus vizinhos para que estes possam atualizar suas próprias informações.

Para uma rede totalmente estática, após a convergência do algoritmo os nós poderiam simplesmente parar sua execução. No entanto, redes de computadores são normalmente dinâmicas, podendo haver mudanças topologias — menos frequentes — ou dos custos dos enlaces — mais frequente. Por este motivo, protocolos de roteamento baseados em Vetor de Distâncias continuam executando o algoritmo de Bellman-Ford indefinidamente, mesmo após a convergência inicial. Cada nó é responsável por monitorar a sua vizinhança e, caso alguma alteração seja detectada — *e.g.*, surgimento, desaparecimento ou mudança de custo de algum enlace —, o nó deve re-computar seu vetor de distâncias de acordo. Qualquer alteração do Vetor de Distâncias demanda que o nó o retransmita para seus vizinhos — para que estes possam atualizar suas próprias informações, caso necessário.

Repare, ainda, em protocolos de roteamento reais baseados em Vetor de Distâncias, **nós realizam o envio de seus Vetores de Distâncias para seus vizinhos periodicamente, ainda que não haja mudanças nos valores reportados**. Para entender porque isso é necessário, devemos notar que a mensagem que carrega o anúncio do Vetor de Distâncias de um nó é um pacote como outro qualquer. Se a rede em questão oferece um serviço de melhor esforço — o que é o caso da Internet, por exemplo — este pacote é susceptível a corrupções e descartes. Em outras palavras, quando ocorre uma alteração no Vetor de Distâncias de um nó x e este realiza o anúncio para seus vizinhos, esta mensagem pode ser perdida. Ao invés de empregarmos algum mecanismo complexo de *acks* e retransmissões, simplesmente optamos por realizar o envio periódico dos Vetores de Distância. Esta solução têm a vantagem adicional de dar suporte direto à adição de novos nós. Como será discutido em aulas posteriores, protocolos reais também se utilizam desta periodicidade para detectarem falhas nos enlaces: se um nó deixa de receber os Vetores de Distância de um vizinho por um longo período, ele pode inferir que o enlace não está mais funcional.

1.1 Vetores de Distância e Mudanças nos Custos dos Enlaces

Na seção anterior, discutimos como o Algoritmo de Bellman-Ford converge sob condições razoáveis na prática. Embora isso seja verdade, protocolos de roteamento baseados em Vetor de Distâncias podem apresentar alguns comportamentos indesejados em situações em que os custos de um enlace mudam abruptamente.

No caso de uma **redução de custo**, estes protocolos conseguem lidar com a mudança de maneira satisfatória. Os nós que estabelecem o enlace em questão detectam a mudança, recalculam seus vetores de distância e, se houver alteração — *i.e.*, redução do custo de alguma rota —, o enviam para seus respectivos vizinhos. Os vizinhos, por sua vez, realizam o mesmo processo. Esta informação da melhora das condições da rede continua se propagando, até que, "rapidamente", todos os nós da rede convergem novamente para as melhores rotas.

Uma situação mais problemática é **o aumento abrupto do custo de um enlace**. Suponha, por exemplo, que um nó y tem um enlace com um nó x com peso inicial 4. Suponha que a única rota alternativa de y para x é passando por um outro nó z — por um enlace de peso 1 — que por sua vez alcança x por um enlace direto de peso 50. Claramente, a melhor rota para y é utilizando o enlace direto, já que $4 < 50 + 1$. Da mesma forma, embora possua um enlace direto para x, z deve utilizar a rota passando por y, já que $1 + 4 < 50$. Após algumas iterações, um protocolo baseado em vetor de distâncias facilmente convergiria para estas rotas.

Suponha, no entanto, que após a convergência inicial de protocolo, o enlace direto de y para x piore de forma repentina, resultando em um novo peso 60. Com esta mudança, as melhores rotas para x se alteram: para z, a melhor rota passa a ser o enlace direto, com custo 50, enquanto para y a melhor rota passa a ser através de z, com custo total 51.

Em um protocolo baseado em Vetor de Distâncias, no entanto, um nó não tem acesso ao estado completo da rede e, portanto, tem que basear suas decisões de roteamento apenas no que conhece da sua própria vizinhança. Em particular, note que quando y detecta a mudança de peso no seu enlace para x, ele possui apenas o Vetor de Distâncias de z como base para tomar uma decisão. Observe que quando y detecta esta alteração, o Vetor de Distâncias que y conhece de z lista ainda o custo antigo — *i.e.*, antes da alteração do peso do enlace — da rota que z conhecida para x. Em outras palavras, este Vetor de Distâncias ainda lista que o melhor caminho conhecido por z para alcançar x tem custo 5. Com base nesta informação, y realiza a seguinte decisão: devo manter minha rota para x usando o enlace direto — que agora tem custo 60 — ou devo utilizar z como próximo salto, resultando em uma rota com custo total de $1+5 = 6$? O resultado desta computação será que y optará pela rota passando por z, atualizará seu Vetor de Distâncias aumentando seu custo para x de 4 para 6 e o reenviará para seus vizinhos — em particular z.

Por sua vez, ao receber o novo Vetor de Distâncias de y, o nó z notará que a distância anunciada de y para x piorou de 4 para 6. No entanto, isso não mudará a opção de z em utilizar y como próximo salto, já que $6 + 1 = 7$ ainda é menor que o custo do caminho alternativo usando o enlace direto de peso 50. De toda forma, o Vetor de Distâncias de z será alterado por conta no aumento do seu custo em direção a x. Isso resultará em z enviar seu novo Vetor de Distâncias para seus vizinhos — em particular, y.

Repare o que acontece se continuarmos este processo:

1. y atualizará seu custo para x de 6 para $7 + 1 = 8$, ainda passando por z e reenviará seu Vetor de Distâncias.

2. z atualizará seu custo para x de 7 para $8 + 1 = 9$, ainda passando por y e reenviará seu Vetor de Distâncias.

3. ...

Em resumo, y e z prosseguirão neste processo que gradativamente aumenta suas estimativas de custo em relação a x de duas em duas unidades. Repare, no entanto, que estes custos não são reais: não existe caminho nesta topologia entre y e x com custo 6 (ou $8, 10, 12, \dots$), assim como não existe caminho entre z e x com custo 7 (ou $9, 11, 13, \dots$). Note ainda que o problema não se resume a uma inconsistência no *custo* da rota, mas a própria rota escolhida não é correta: y acredita que deve encaminhar pacotes destinados a x por z, que pensa que deve estes mesmos pacotes por y. Em outras palavras, há um *loop* no roteamento.

Este problema — denominado de **contagem ao infinito** — é apenas temporário, já que os custos destas rotas inexistente aumentam gradativamente. Em certo momento, esta rota fictícia passa a ter custo superior ao da rota ótima, fazendo com que o roteamento convirja para uma situação normal novamente. Entretanto, isso demora algum tempo: são necessárias várias iterações e trocas de Vetores de Distância. Durante esse período, a inconsistência no roteamento pode levar a pacotes ficarem em *loop* e, eventualmente, serem descartados.

Protocolos baseados em Vetor de Distância geralmente incluem uma contra-medida para combater a contagem ao infinito chamada de **envenenamento reverso** (ou *poisoned reverse*, em inglês). O mecanismo é simples. Um nó gera versões diferentes do seu Vetor de Distâncias para vizinhos diferentes. Em particular, se z utiliza y como próximo salto para o destinatário x, então no Vetor de Distâncias z anunciado para y constará uma distância infinita em relação a x. Para seus outros vizinhos, z anuncia sua estimativa correta para sua distância para x.

Se aplicarmos o envenenamento reverso ao exemplo anterior, é possível notar que a contagem ao infinito não ocorre. Quando o protocolo converge na situação inicial da rede — quando o enlace entre y e x tem peso 4 — z descobre que sua melhor rota para x tem custo 5, utilizando y como próximo salto. Nesta situação, ao anunciar seu Vetor de Distâncias para y, z propositalmente omite sua distância conhecida para x, substituindo-a por infinito. Quando y detecta o aumento abrupto no peso do seu enlace para x, ele acredita que z não possui uma rota para x e, portanto, se vê obrigado a continuar usando a rota pelo enlace direto — agora com custo 60. Mesmo assim, dada a mudança no custo, y ainda atualiza seu Vetor de Distâncias e o reenvia para seus vizinhos. Ao receber este novo Vetor de Distâncias, z percebe que sua rota para x utilizando y como próximo salto teve um aumento súbito de custo — de 5 para 61. Neste caso, sua rota alternativa — usando seu enlace direto para x — se torna melhor, com custo 50. Devido à alteração no seu Vetor de Distâncias, z o reenvia para todos os

seus vizinhos. Agora, ao receber esta nova atualização de z, o nó y descobre uma rota melhor para x utilizando z como próximo salto, resultando em um custo 51.

O envenenamento reverso, portanto, é eficaz em evitar o problema de contagem a infinito *em várias situações*, **embora não em todos os casos**.

1.2 Vetor de Distâncias *vs.* Estado de Enlaces

Uma questão natural neste ponto é: qual das duas abordagens — Vetor de Distâncias ou Estado de Enlaces — é melhor? Na realidade, não existe uma resposta absoluta. Cada abordagem apresenta vantagens em determinados aspectos, demandando que a opção por uma ou outra seja baseada nas características da rede e outros requisitos relevantes.

Do ponto de vista de complexidade computacional, protocolos baseados em Estado de Enlaces são, normalmente, mais custosos, tanto em termos de processamento (já que cada nó precisa computar sozinho a instância do algoritmo de caminho mínimo), quanto em termos de armazenamento (já que cada nó necessita manter uma representação completa da topologia da rede em memória). Estes fatores podem limitar a aplicabilidade deste tipo de protocolo a redes de grandes dimensões.

Outro fator relevante é a **complexidade de troca de mensagens**, ou seja, a quantidade de mensagens de controle que precisam ser trocadas entre os nós para que o protocolo consiga convergir. Neste aspecto, é preciso considerar a natureza diferente das mensagens geradas por cada tipo de protocolo: os baseados em Estado de Enlaces precisam que suas mensagens sejam difundidas por toda a rede, enquanto o escopo das mensagens usadas por protocolos baseados em Vetor de Distâncias é apenas local — *i.e.*, para os vizinhos diretos de cada nó. Por outro lado, os Vetores de Distâncias crescem com o aumento do tamanho rede, enquanto o tamanho das mensagens de controle dos protocolos baseados em Estado de Enlaces dependem apenas do número de vizinhos de um nó — que, normalmente, é bem menor que o número total de nós na rede.

O tempo de convergência dos protocolos é outro ponto de comparação. A princípio, o tempo de convergência para um protocolo baseado em Estado de Enlaces depende do tempo de disseminação das informações de topologia, seguido do tempo de execução do Algoritmo de Dijkstra — normalmente, o tempo de comunicação é dominante neste contexto. Já nos protocolos baseados em Vetor de Distâncias, a convergência depende do número de iterações necessárias para o Algoritmo de Bellman-Ford. Por sua vez, este número de iterações depende do **diâmetro da rede** — *i.e.*, o maior número de saltos entre quaisquer dois nós da rede.

Ambas as abordagens estão susceptíveis a problemas como oscilações — geralmente associadas à métricas de roteamento que variam com o tráfego que passa por um enlace — e *loops*. A causa dos *loops*, no entanto, é diferente para cada tipo de protocolo: para protocolos baseados em Estado de Enlaces, *loops* pela inconsistência nas visões da topologia dos vários nós; já nos baseados em Vetor de Distâncias, a causa é geralmente o problema de contagem ao infinito.

É possível, ainda, compará-las em termos de robustez a defeitos ou *bugs* em equipamentos. Se um roteador defeituoso anuncia custos ou pesos errados nas mensagens de controle do protocolo de roteamento, o escopo das consequências negativas tende a ser mais localizado na vizinhança deste nó em protocolos baseados em Estado de Enlace, já que um nó é responsável por anunciar pesos apenas dos seus enlaces. Por outro lado, em um protocolo baseado em Vetor de Distâncias, este escopo tende a ser bem maior, já que as informações constantes nas mensagens de controle dizem respeito ao custo de caminhos completos.

2 Roteamento Hierárquico

Embora tenhamos concluído na seção anterior que protocolos de roteamento baseados em Vetor de Distância têm melhor escalabilidade que aqueles baseados em Estado de Enlaces, sua escalabilidade ainda é problemática se considerarmos uma rede das dimensões da Internet com potencialmente bilhões de nós. Pense, por exemplo, no tamanho dos Vetores de Distância trocados pelos nós.

Outro problema é a dificuldade de uniformização na escolha de um protocolo de roteamento para a Internet como um todo: como garantir que uma rede que não possui um único órgão administrativo centralizado convirja para a escolha de um único protocolo? Certas organizações podem ter suas preferências particulares neste aspecto por diversos motivos.

Para resolver estes problemas, a Internet utiliza uma abordagem de **roteamento hierárquico** — em oposição a um **roteamento plano**. Nesta abordagem hierárquica, a topologia da Internet é dividida regiões chamadas de **Sistemas Autônomos** (ou AS, do inglês *Autonomous System*). Um sistema autônomo é basicamente uma região controlada por alguma entidade que tem, portanto, autonomia para configurar e manipular seus dispositivos da maneira como julgar mais adequado.

Com esta divisão em sistemas autônomos, o problema de roteamento na Internet é dividido em dois escopos: determinar as rotas entre nós pertencentes a um mesmo AS — o chamado **roteamento intra-AS** — e determinar as rotas entre nós pertencentes a ASs diferentes — o chamado **roteamento inter-AS**.

A divisão do roteamento nestes dois escopos permite que sejam utilizados protocolos de roteamento diferentes para estes dois problemas: um protocolo intra-AS e um protocolo inter-AS. O protocolo intra-AS é de escolha livre de cada AS e tem seu escopo de atuação limitado à topologia interna do AS. Por outro lado, o protocolo de roteamento inter-AS **precisa ser unificado na Internet**, sendo sua execução um esforço colaborativo entre os vários ASs. Em particular, esta colaboração entre ASs se dá através da comunicação dos ***gateways* de borda**, roteadores que se encontram nas bordas dos seus respectivos ASs e, portanto, estabelecem enlaces com os roteadores de borda de ASs vizinhos.

Assim, ao olharmos para a tabela de roteamento de um roteador em um sistema autônomo, encontraremos tanto entradas aprendidas através do roteamento intra-AS — para destinatários do mesmo AS —, quanto através do roteamento inter-AS — para destinatários externos.

Um ponto que muitas vezes causa confusão é onde o protocolo inter-AS é executado. Claramente, os roteadores que atuam como *gateways* de borda precisam executar o protocolo de roteamento inter-AS para que eles possam anunciar e receber informações de roteamento para/de outros ASs vizinhos. Mas também é necessário que os anúncios recebidos de outros ASs sejam propagados para o interior do AS para que estes roteadores internos possam preencher suas tabelas de roteamento com rotas para endereços externos.

Em resumo, o escopo de atuação do protocolo inter-AS inclui as seguinte atividades:

1. **Anúncios e recebimentos de prefixos**. Os *gateways* de borda de um AS se comunicam com os *gateways* de borda de outros ASs vizinhos para trocar informações sobre **prefixos alcançáveis**. Isto é, cada *gateway* de borda anuncia aos seus pares prefixos de sub-rede para as quais seu AS consegue encaminhar pacotes. Note que estes prefixos incluem sub-redes pertencentes a outros ASs, desde que o AS deste *gateway* conheça uma rota para alcançá-la.

2. **Disseminação dos anúncios recebidos para dentro do AS**. Ao receber anúncios de prefixos de outros ASs, o *gateway* de borda deve disseminar estas informações para os demais roteadores do seu AS.

3. **Escolha entre potenciais múltiplos anúncios**. Caso um AS receba mais de um anúncio de rota para um mesmo prefixo, é preciso aplicar algum critério para escolher uma das opções de rotas recebidas. Um critério comum é o chamado **roteamento batata quente**, no qual cada roteador do AS escolhe a rota que minimiza o custo de encaminhamento **dentro do AS**. Em outras palavras, neste critério um roteador escolhe a rota através que utiliza *gateway* de borda mais próximo dele. Note que este critério pode resultar no uso de rotas diferentes por roteadores diferentes do mesmo AS. Finalmente, note que este critério utiliza informações coletadas pelo roteamento intra-AS — *e.g.*, o custo do caminho entre o roteador e os *gateways* de borda do AS.