Teil III: Routing - Inhalt I

Einführung
- Problemstellung
- Internet
- Ad-hoc Netze

Link Reversal Routing
- Dags
- Algorithmus
- Korrektheit und Eigenschaften

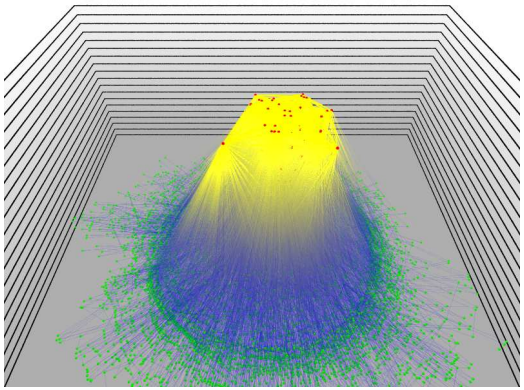
Routing

- Gegeben: Graph $G = (V, E)$, Quellknoten s_i, Zielknoten t_i
- Gesucht: Verteilter Algorithmus um Nachrichten von s_i nach t_i zu senden („Routing Protocol“)
- Bewertung von Routen ($=$ Wege in G)
  - Anzahl der verwendeten Kanten
  - Datendurchsatz auf der Route
  - Kosten für die Provider
  - ...
- Oft zwei Stufen
  - Vorberechnung („proaktiv“) von Informationen an Knoten
    - Problematisch bei mobilen Funknetzen: Topologieänderungen, Energieverbrauch, ...
  - Versenden von s_i nach t_i gemäß diesen Informationen

Routing im Internet

- Autonome Systeme (AS)
- Router zwischen Autonomen Systemen
  - Border Gateway Protocol (BGP)
- Router innerhalb eines Autonomen Systems
  - „Distance-Vector Protocols“, z.B. RIP
  - „Link-State Protocols“, z.B. OSPF
- Clients mit Verbindung zu einem Router („Default Gateway“)

Border Gateway Protocol: Top-Level Routing



AS-Graph

Routing Information Protocol (RIP)
Intra-AS Routing

- Basiert auf Bellman-Ford Algorithmus für kürzeste Wege:

1. $d_s := 0$, $d_v := \infty$ für alle $v \in V \setminus \{s\}$
2. Vorgänger vor(v) :=null für alle $v \in V$
3. für $i := 1$ bis $|V|$
4. für jede Kante $e = \{u, v\}$
5. falls $d_v > d_u + w_e$
6. $d_v := d_u + w_e$
7. vor$(v) := u$
8. (teste auf negative Kreise)

RIP: Verteilter Bellman-Ford Algorithmus (DBF)

- Knoten (Router) speichern Routing Tabelle
  - Für jede Zieladresse wird Distanz und Vorgänger gespeichert „Distance Vector“
  - Route Aggregation: Zusammenfassen von Subnetzen
- Berechnung der Routing Tabellen
  - Verteilter „all-pairs“ Bellman-Ford Algorithmus
  - Weglänge z.B. Anzahl Kanten („Hop Count“)
  - Periodische (asynchrone) Kommunikation (alle 30 Sekunden)
- Finden der Route zu Zieladresse
  - Sende Nachricht zu in der Routing Tabelle gespeichertem Vorgänger

RIP: Verträglichkeit mit Topologieänderungen

- Kein Abbruch
- Falls vor$(v) = u$ wird immer $d_v := d_u + w_e$ gesetzt, selbst wenn $d_v \leq d_u + w_e$
- Time-out: Nach 180 Sekunden ohne Nachricht von u
  - setze für alle Einträge mit vor$(v) = u$ die Distanzen auf ∞

Open Shortest Path First (OSPF)

- Gebräuchlichstes Inter-AS Routingprotokoll
- Knoten (Router) speichern den Kommunikationsgraph
  - „Link-State“
  - Verteilter Aufbau des Graphen durch „advertise“ Nachrichten
- Jeder Router berechnet einen Kürzeste-Wege Baum
  - Algorithmus von Dijkstra
- Hierarchisch
  - AS kann in Gebiete und Backbone aufgeteilt werden
  - Jeder Knoten speichert nur gebietsinterne Graphstruktur

Routing in Sensor und Ad-hoc Netzen

- Energie ist knapp
  - Ständiges „Flooding“ z.B. von Distanzvektoren ungeeignet
- Mobilität, Kommunikation über Funk
  - Topologie kann sich oft ändern
  - Routing-Tabellen werden ungültig und werden zu langsam aktualisiert
- Begrenzte Ressourcen
  - Speichern von kompletten Routing-Informationen evtl. nicht möglich
- Adressierung von Anfragen an „das Netz“
  - Punkt zu Punkt Kommunikation ungeeignet

Routing-Protokolle für Ad-hoc Netze

- Proaktiv: CGSR, DBF, DSDV, HSLS, HSR, LCA, MMRP, OLSR, STAR, TBRPF, WRP, ...
- Reaktiv („On-demand“): ARA, ABR, AODV, BSR, CHAMP, DSR, DSRFLOW, DNVR DYMO, FORP, GB, LBR, LMR, LUNAR, MOR, MPRDV, RDMAR, SSR, TORA, PLBR, ...
- Geo-Routing: Face, GOAFR, GOAFR+, ...
- Hierarchisch: BRP, CBRP, CEDAR, DART, DDR, FSR, GSR, HARP, HSR, IARP, IERP, LANMAR, OORP, ZRP, ...
- Geographisch: DREAM, GLS(Grid), LAR, GPSAL, ZHLS, GPSR, ...
- Energiebewußt: ISAIAH, PARO, EADSR, PAMAS, ...
- Multicast: ABAM, ADMR, AMRIS, DCMP, AMRoute, CAMP, CBM, DDM, FGMP, LAM DSR-MB, MAODV, MCEDAR, MZR, ODMRP, SRMP, LBM, GeoGRID, GeoTORA, MRGR, ...
- Datenbasiert: Directed Diffusion, ...

Link Reversal Routing „Gafni-Bertsekas Algorithmus“

- Sehr ähnliche Vorgehensweise im „Temporally-Ordered Routing Algorithm“ (TORA)
  - Reaktives Protokoll
  - Routing-Information wird nur bei Bedarf berechnet
- Gegeben: Graph G, Startknoten s und Zielknoten t
- Aufbau einer gerichteten azyklischen Version von G
  - Wird hier auch als gegeben vorausgesetzt
- Aktualisieren der Richtungen bei Änderungen im Graphen
  - Änderungen sind Einfügen oder Löschen von Kanten
  - Aktualisierung durch Umkehren einiger Kanten („link reversal“)

Gerichteter azyklischer Graph (DAG)

Definition 1
Ein gerichteter Graph G ohne gerichteten Zykel heißt Dag („directed acyclic graph“). Weiter heißt der Graph t-zielorientiert genau dann wenn t der einziger Knoten ohne ausgehende Kante ist.

Topologische Nummerierung

Definition 2
Eine Abbildung $t : V \to \mathbb{N}$ mit $t(u) \neq t(v)$ für $u \neq v$ heißt topologische Nummerierung von G falls

$$(u, v) \in E \Rightarrow t(u) > t(v).$$

Die sich ergebende absteigende Anordnung der Knoten heißt topologische Anordnung.

Lemma 3
Ein gerichteter Graph ist genau dann ein Dag wenn es eine topologische Anordnung der Knoten gibt.

Berechnen topologischer Nummerierungen

- Sei D ein gerichteter azyklischer Graph
  - Tiefensuche in D
  - Nummer eines Knotens:
    Letzter Bearbeitungszeitpunkt während der Tiefensuche

Berechnen eines t-zielorientierten Dags

- Sei G ein ungerichteter Graph und $A = (v_1, \ldots v_n)$ eine beliebige Anordnung der Knoten
- Breitensuche in G ausgehend vom Ziel t
- $l(u)$ sei der Level von $u \in V$ im Breitensuchbaum
- Orientiere eine Kante zwischen zwei Knoten: (u, v)
  - Zeigt vom größeren zum kleineren Level ($l(u) > l(v)$), oder
  - ist gemäß A gerichtet ($u = v_i, v = v_j, i < j$)

Routing mit DAGs

- Der Graph muss t-zielorientiert sein
  - Das Ziel t ist einziger Knoten ohne ausgehende Kante
- Routing von s nach t gemäß der Richtungen
  - Versende Nachricht über eine ausgehende Kante
  - Kein „kürzeste Wege Routing“
  - Mehrere Wege möglich
- Nachricht kommt bei t an weil
  - G keine Kreise enthält und
  - G zielorientiert ist.

Aktualisieren eines desorientierten Dags Spezialfall „Full Reversal“

Verteilter Algorithmus

1. Falls der Knoten nicht Zielknoten ist
2. Falls der Knoten keine ausgehende Kante hat
3. Kehre die Richtungen aller eingehenden Kanten um

Aktualisieren eines desorientierten Dags Höhenwerte für Knoten

- Die Knoten seien $v_1, \ldots, v_{n+1} = t$
- Sei $\mathcal{H}$ (Höhe) eine abzählbar unendliche Menge mit
  - einer totalen Ordnung $<$
  - einer Unterteilung in unendliche Teilmengen $\mathcal{H}_1, \ldots \mathcal{H}_n$
  - $\mathcal{H}_i$ ist nicht beschränkt bzgl. $<$
  - $\mathcal{H}_i$ ist eine abelsche Gruppe bzgl. einer Addition $+$
- $X := \mathcal{H}_1 \times \mathcal{H}_2 \times \ldots \times \mathcal{H}_n$
- $x = (a_1, \ldots, a_n) \in X$ definiert eine Orientierung der Kanten:
  - $a_i > a_j \Rightarrow$ Kante ist gerichtet von i nach j
    (Kanten von „oben“ nach „unten“ gerichtet)
  - Der so gerichtete Graph ist azyklisch
    (x liefert topologische Anordnung)

Aktualisieren eines desorientierten Dags Zu aktualisierende Knoten

- $S(x) := \{i \mid \{i, n+1\} \notin E$ und
  $a_i < a_j$ für alle j mit $\{i, j\} \in E, 1 \leq i \leq n\}$
- Der x-orientierte Graph ist zielorientiert g.d.w. $S(x)$ leer
- $S(x)$ ist Menge der Knoten ohne ausgehende Kanten

Aktualisieren eines desorientierten Dags
Algorithmus

Algorithmus

- Gegeben x_0 (top. Nummerierung des ursprünglichen Dags)
- Berechne iterativ $x_{k+1} \in \mathcal{A}(x_k)$
- $\mathcal{A}(x_k)$ ist eine Teilmenge von X

Bemerkung

- Es wird verteilte Berechnung modelliert
- Mehrere Möglichkeiten aus x_k neue Höhen x_{k+1} zu berechnen
  - Reihenfolge, in der die Knoten in $S(x_k)$ abgearbeitet werden
  - Parallelität
- $\mathcal{A}(x_k)$ enthält alle möglichen verteilten Berechnungsschritte

Aktualisieren eines desorientierten Dags
Festlegung des Algorithmus

Voraussetzungen an $\mathcal{A}$

1. Es gibt Funktionen $g_i : X \to \mathcal{H}_i$, so dass entweder
   - $\mathcal{A}(x) = \{x\}$ falls $S(x)$ leer, ansonsten
   - $\mathcal{A}(x) = \{\overline{x} \mid \overline{x} \neq x$ und entweder $\overline{a}_i = a_i$ oder $\overline{a}_i = g(a_i)\}$
2. Es gilt
   - $g_i(x) > a_i$ falls $i \in S(x)$, und $g_i(x) = a_i$ sonst
   - $g_i(x)$ hängt nur von a_j ab für die $\{i, j\} \in E$
3. Sei $(x^k)_{k\in\mathbb{N}}$ unendliche Folge, Knoten i unendlich oft in $S(x^k)$
   - Dann ist
   $$a_i^0 + \sum_{r=0}^{k} (g_i(x^r) - a_i^r)$$
   nicht beschränkt in $\mathcal{H}_i$

Beispiel: Full Reversal

- Definition der Höhenwerte $\mathcal{H}_i$ und der Funktionen g_i
  - $\mathcal{H} = \mathbb{N} \times \{1, \ldots, n\}$, lexikographisch sortiert
  - $(\alpha_i, i) + (\alpha_i', i) := (\alpha_i + \alpha_i', i)$
  - Sei $x = ((\alpha_1, 1), \ldots, (\alpha_n, n))$:

$$g_i(x) = \begin{cases} (\max\{\alpha_j \mid \{i,j\} \in E\} + 1, i) & \text{falls } i \in S(x) \\ (\alpha_i, i) & \text{falls } i \notin S(x) \end{cases}$$

- Es gelten die Voraussetzungen 1-3

Korrektheit

Satz 4

Sei G zusammenhängend. Für jede Folge $(x_k)_{k\in\mathbb{N}}$, die vom Algorithmus generiert wird, existiert ein Index $\overline{k}$, so dass $S(x_k)$ leer ist für alle $k \geq \overline{k}$.

Bemerkung

Der Algorithmus konvergiert also nach endlich vielen Schritten in einem $(n+1)$-zielorientierten Dag. ($S(x_k)$ ist leer für $k \geq \overline{k}$.)

Eindeutigkeit

Satz 5
Sei G zusammenhängend und $x_0 \in X$. Dann gibt es ein $x^* \in X$ (das nur von G und x_0 abhängt), so dass es für jede Folge $(x_k)_{k\in\mathbb{N}}$, die von obigem Algorithmus berechnet wird, ein $\overline{k} \in \mathbb{N}$ gibt mit $x_k = x^*$ für alle $k \geq \overline{k}$.

(ohne Beweis)

Stabilität

Lemma 6
Sei G zusammenhängend und $(x^k)_{k\in\mathbb{N}}$ eine Folge, die vom Algorithmus generiert wurde. Dann gilt für jeden Knoten i $(1 \leq i \leq n)$, für den im x^0-orientierten Graph ein Weg $(i, j_1, j_2, \ldots, j_m, n+1)$ existiert (d.h., $a_i^0 > a_{j_1}$, $a_{j_1}^0 > a_{j_2}^0, \ldots$):

$$a_i^0 = a_i^k \text{ für alle } k > 0.$$

Laufzeit

Lemma 7
Sei b die Anzahl der „schlechten" Knoten in x_0, d.h. die Knoten ohne Weg zum Ziel t. Dann endet der Full-Reversal Algorithmus nach $O(b^2)$ Berechnungsschritten.

(ohne Beweis)

Lemma 8
Sei $b \in \mathbb{N}$ gegeben. Es gibt Graphen und Ausgangshöhen x_o mit b „schlechten" Knoten, für die jeder Link-Reversal Algorithmus $\Omega(b^2)$ Berechnungsschritte ausführt.

(ohne Beweis)

Link Reversal Routing
Literatur

- Eli M. Gafni und Dimitri P. Bertsekas: Distributed Algorithms for Generating Loop-Free Routes in Networks with Frequently Changing Topology. *IEEE Transactions on Communications*, Com-29:1, 1981.
  `http://web.mit.edu/dimitrib/www/Gafni_Loopfree.pdf`
- C. Busch, S. Surapaneni und S. Tirthapura: Analysis of Link Reversal Routing Algorithms for Mobile Ad Hoc Networks. *Proc. ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 210-219, 2003.
  `http://www.cs.rpi.edu/~buschc/papers/reversal.pdf`