(CAN つづき)

3. Design Improvements

low per-node state O(d)
short path length O(d n^(1/d))
  これらはすでに実現

しかしこれは application level hops
  not IP-level hops
    CAN 上の隣接ノードが非常に離れている可能性も

目標1: reduce the latency of CAN routing
  IP-level latency との比較

ついでに robustness も向上
  routing
  data availability

そのためには:
path length を減らす
OR
per-CAN-hop latency を減らす

目標2: simple load balancing

シミュレーション
  Transit-Stub topologies
  2-level
    transit domain (high level)
    stub domain (low level)

3.1 Multi-dimensioned coordinate spaces
次元数 d を増やすことで, path-length を減らせる
(increasing the dimensionality d reduces the path-length)

Fig. 4
  d を変えて, ノード数とホップ数の関係 (両対数)
  O(d * n^(1/d)) に合致

d が増えると隣接ノードも増える
  (increasing d implies more neighbors)
  故障時の代替ルートの候補が増えることに
  (more potential next hop nodes in case of failure)

dをどんどん増やすと
  上限は log N 次元
  隣接領域は log N 個
  ルーティングのコストも log N

3.2 Realities: multiple coordinate spaces
独立した座標空間(reality)を複数持つことにする
  空間毎に, ノードは異なるゾーンを担当
  (a node is assigned a different zone in each reality)

r 個の reality
  隣接セットも r組 (r set of neighbors)
  初期接続時に r回 場所取りをする

content も reality 毎に複製する (content is replicated on every reality)
  データの availability 向上
  たとえば location (x,y,z)に対応するデータは
   (a file storead at location (x,y,z) is stored in r nodes)
    r 個のノードが保持
  どれか生存していれば available

routing fault への耐性も向上
  失敗したら別の reality で送信

(on routing failure, try in another reality)

routing path length の減少
目的地にもっとも近い reality で送信
(use the reality which offers the shortest path)

Fig. 5
reality数ごとに, ノード数とホップ数の関係

3.1 と 3.2 の関係
(relationship between 3.1 and 3.2)

どちらも同じトレードオフ (both have the same tradeoffs:)
○ shorter path length
× per-node neighbor state
× maintenance traffic

Fig. 6
隣接ノード数とホップ数の関係
(d, r を増やしたとき)
d を増やした方が有効 (increasing d is more effective)
しかし, rを増やすと他にも利点が
(increasing r offers other benefits)

3.3 Better CAN routing metrics

隣接ノードへの IP での round-trip-time RTT を計測
(measure IP-level RTT to neighbors)

もっとも 進捗 / RTT の比の高い隣接ノードに送信
(forward the message to the neighbor with the maximum ratio
of progress to RTT)

CAN のホップ毎の latency を下げる働き
(reduces the latency of inidivisual CAN hops)

simulation
100ms (intra-transit)
10ms (stub-transit)
1ms (intra-stub)
平均 latency は 115ms
Table 1.
per-hop latency の値
ノード数は 2^8 から 2^18 まで

RTT weighting がないと (without RTT weighting)
平均 latency に一致 (per-hop latency matches the average latency)

RTT weighting があると (with RTT weighting)
24% から 40% 低下
d が高いほど, 選択肢が多いので有効
(higher d gives more next-hop choices, hence greater improvement)

3.4 Overloading coordinate zones
これまでは ゾーンひとつは1ノードが担当
(so far, one node for a zone)

複数ノードがひとつのゾーンを共有するように
(multiple nodes to share the same zone)
その複数ノードを peers と呼ぶ
MAXPEERS (3 or 4)

ノードは自分のpeerのリストも保持
(a node maintaines a list of its peers)
隣接ゾーンについては, (peer 全部ではなく)一つのノードだけを保持
(a node select one neighbor for a neighbor zone)
IP的にもっとも近いものを選ぶ(後述)
保持情報は MAXPEERS 増えるだけ

(per-node state increases only MAXPEERS)

ゾーンが拡大することによって、データ量も増加

新規ノードの参加時
(when new node joins)
既存ノードのゾーンを分割しないで
(instead of splitting the zone)
peer として参加
(the new node joins as a peer)

すでに peer が MAXPEERS 存在していれば分割
(if the number of peers >= MAXPEERS)
zone も peer list も二等分する
(split zone and peer list)
deterministic rule

隣接ノードにpeer listを定期的に要求
(request neighbors for peer list periodically)
list のノードへのRTTを計測
(measure RTT to the nodes in the list)
最小のRTTのノードを隣接ノードとして保持
(retains the node with lowest RTT as the neighbor node)
トラフィックに負荷をかけない程度に (very infrequent intervals)

contents は peer で分割しても複製しても可
(content may be divided or replicated)
複製 (replication results in)
○ availability
× data size
× consistency maintenance

overload の利点 (advantages)
reduced path length
総ゾーン数の減少になるため
(the number of zone is reduced)
reduced per-hop latency
近い隣接ノードを選べるから
(can select closest neighbor)

Table 2
Table 1 と同条件 (same topologies)
4ノードで45%減少

improved fault tolerance
peerが全滅しない限り安全
(unless all the nodes in a zone crash)

× システムの複雑化
(disadvantage: system complexity)

3.5 Multiple Hash Functions
k different hash functions
single key -> k points
(key, value) pair の複製
データが複製される

availability の向上 (improvement)

query の k 並列実行
(k parallel queries)

Fig. 7
並列実行の効果 (effect of parallel queries)

並列ではなく, もっとも近いノードに向かう方法も可能
(instead, might choose a closest neighbor from k neighbors)

3.6 Topologically-sensitive construction of the CAN overlay network
sec. 2.2 のノード配置はランダム (nodes are allocated at random so far)
  隣接ノードはIPで近いとは限らず
  (neighbor nodes are not necessarily near in IP-level)
  この問題は,ここまでは手つかず

CANのトポロジを, IPトポロジに合わせる
(contruent with the underlying IP topology)

"landmarks"
  well known set of machines

各ノードはlandmarkとのRTTを計測
(every CAN node measures RTT to each of landmarks)
  RTT昇順の landmark リストを作成
  (order landmarks in a list)
その順列(m!通り)で全空間を分割
(partition the coordinate space into m! portions)
  第1軸をm分割 (divide the first dimension into m portions)
  第2軸を(m-1)分割 (second dimension into m-1)
  ...
新規ノードは自分の計測結果を使って場所ぎめ
(a new node enters a portion using its landmark ordering)
  その部分空間内でランダム
  (at a random point in the portion)

"distributed binning"

Fig. 8
  stretch = CAN latency / average IP latency
  same simulation topology as before
  4 landmarks

その結果として, 空間は不均一になる
(the coordinate space is no longer uniformly populated)
  "background load balancing techniques" (in appendix A)

3.7 More Uniform Partitioning
新規参加ノードはランダムな点を選びノードを探す
(a new node selects a random point)
  見つかったノードが領域を分割していた
  (the owner node split its zone to the new node)

自分の領域を分割しないで
(instead of splitting its own zone)
  隣接ノードで最大の領域のノードを分割させる
  (the neighbor with largest zone will split the zone)

より均等な領域分割が可能
(more uniform partitioning of the space)
  データも均等に分配される
  (achieve load balancing)

問題点:データによってアクセス頻度に差がある
(some files are accessed heavily)
  "hot spot"
  均等に分布させるだけでは不十分
  caching, replication (in 3.8)

シミュレーション (Fig. 9)
  全空間の体積 Vt (volume of the coordinate space)
  ノード総数 n (number of nodes)
  V = Vt / n (average volume per node)
  2^16 ノードで上述の手法の有効性
  (evaluate the uniform partitioning method)
    あり 90% が V に一致
    なし 45% が V

d が大きくなると更に有効
  隣接ノード数が増えるためではないかと

3.8 Caching and Replication for "hot spot" management
(key, value) pair にアクセス頻度の差
(frequecy of access is not uniform)
  Web の世界での手法を採用
  (techniques commonly applied to the Web)

Caching
  ノードが最近アクセスした key をキャシュ
  (cache data keys it recently accessed)
  ヒットしたら forward しないですぐ返す

Replication
  要求の多いキーの複製を隣接ノードに置く
  (replicate popular data key at each of neighbors)
  オリジナルノードの周辺にキーが分布
  (original storage node is surrounded by replicas)

  へり のノードは自分で答えるかforwardするか
  (a node with replica chooses to either answer or forward the query)
    適当な確率で
    (in a certain probability)

適切な Time-to-live で expire させる
(cache and replicas are expired in a certain TTL)

4. Design Review
2,3章で, 個々の design component を説明した
(Sec. 2 and 3 described individual design components)

この章ではそれらの総合的な効果について説明
(in Sec. 4, cumulative effect of all the features)

性能の指標 (metrics)
path length
  (アプリケーションレベルの)ホップ数
  (number of (application-level) hops)
neighbor state
  ノードが情報を保有する隣接ノード数
  (number of CAN nodes for which an node must retain state)
latency
  end-to-end latency
  per-hop latency
volume
  ノードが担当する zone の広さ(体積)
  リクエスト量, データ保存量の指標
  (indicative of the request and storage load)
routing fault tolerance
  複数パスがあるか
  (availability of multiple paths)
hash table availability
  複製によるデータ利用可能性

デザインパラメタ (design parameters)
  dimensionality: d
  number of reality: r
  zone 当たりの peer 数: p
  hash function の数: k
    一つのデータがマップされる点の個数
  RTT-weighted routing を使うかどうか
  uniform patitioning feature を使うかどうか

パラメタの効果 (the effect of a parameter)
  アルゴリズムから決定できるもの
  (directly inferred from the algorithm)
  シミュレーションが必要なもの

(need simulation)

Table 3 パラメタと性能指標の関係
"-" : 効果なし
"↑" : パラメタ増大で性能向上
"↓" : パラメタ増大で性能低下
図番号 : シミュレーションの結果

全部の feature の総体での効果について測定
(cumulative effect of all the features)

n = 2^18 (= 256K)
トポロジーは前と同じ (the same topology as before)
Transit-Stub

Table 4 パラメタ設定
1. bare-bones
2. knobs-on-full
landmark ordering 以外すべて

Table 5 結果
IP latency の2倍以下で到達
(latency within a factor of two of the IP-latency)
保持するノード数は約30
(local state is 30 nodes)
次元数の増加で, 最大の利得
(increasing the dimensionality gives the biggest gain)
パス長 198->5 (pathlength)
latency reduction heuristics (RTT-weighted)
重要な効果 (plays an important role)
なければ CAN latency は 5*115 = 575 のはず
(without this, CAN latency would be 5*115 = 575)

ノード数を変化させてみた (varies the system size n)
2^14 - 2^18
その際, Transit-Stub の backbone は変化させず
(in scaling, nodes are added to the edges)
n を大きくしても, n^(1/d) より増加は少ない
(total path latency grows slowly)
近距離のホップが増えるから
(because added hops are along low-latency links)

悲観的に n^(1/d) として外挿すると
(extrapolating the pessimistic grow rate)
2^10倍にしても path latency は IPの4倍以内

Transit-Stub の遅延分布を変えてみる
knob-on-full で実験
H(100,10,1) オリジナルの設定
intra-transit 100ms
transit-stub 10ms
intra-stub 1ms
H(20,5,2)
オリジナル設定に比べて差が少ない
R(10,50)
どのリンクも 10から50msの乱数
10xH(20,5,2)
backbone を10倍の数に
ノード密度が1/10に低下

Figure 10
その結果
縦軸は latency stretch
(CAN latency/IP latency)
遅延分布は値には影響するが
上昇率はいずれも低い
最高はランダムのケース
新規ノードが追加される時のリンクが早くない

H(20,5,2) のほうが 10xH(20,5,2) より低い
  密度が高い方が RTT-weighting が有効

5. Related work
6. Discussion