IIJ Internet Initiative Japan

トラフィック計測のための多次元フロー集約アルゴリズム

長　健二朗
IIJ技術研究所

IIJ Technical WEEK 2017. 2017-11-10

コンパクトなトラフィック情報

- トラフィックの監視の必要性
  - 利用状況の把握
  - 異常検出 （トラフィック集中、攻撃、設定ミスなど）
- 簡潔なサマリ情報の必要性
  - 問題発見に十分で、かつ、見落としが起きない情報量
- サマリ情報の応用
  - 検出した異常時のトラフィック状況把握
  - APIの提供によりシステム統合を可能にする

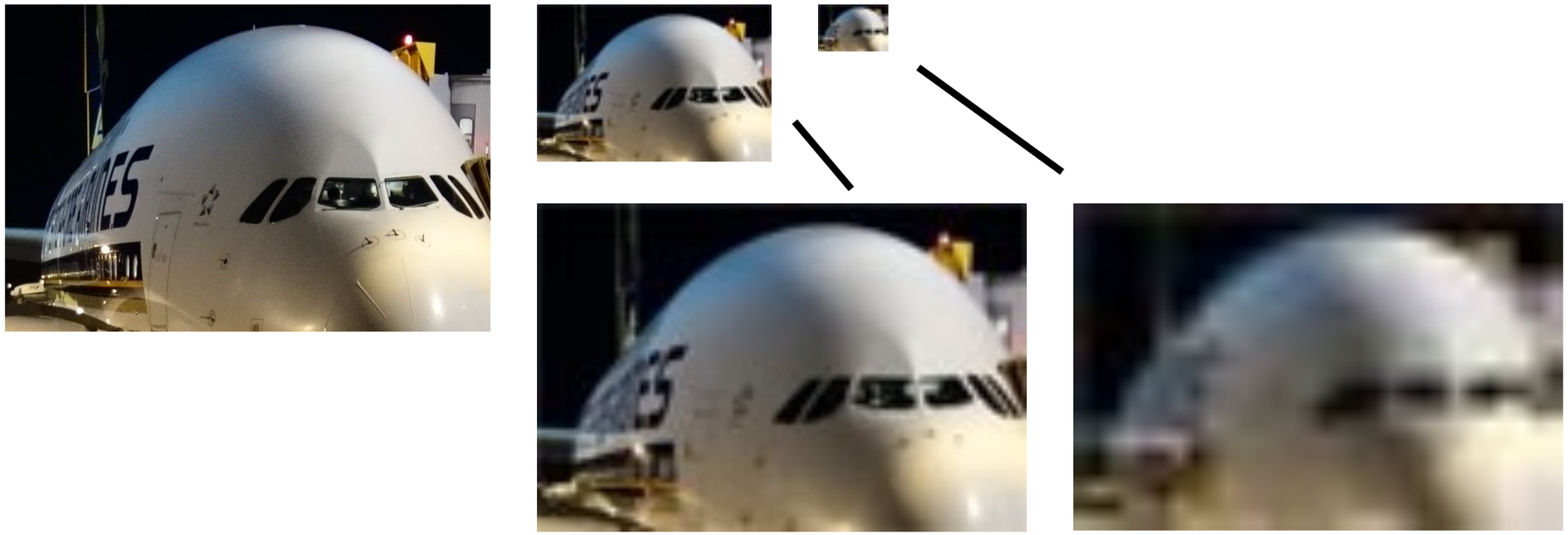
フロー集約

- フロー：5-tuple (src_IP, dst_IP, src_port, dst_port, protocol)
  - 単一のTCPセッションなど (NetFlow/sFlow/IPFIXで取得可能)
- フローを集約することで、重要なパターンを抽出
- 集約フロー：共通属性で集約されたフロー
  - 例：TCPセッション from 10.0.0.0/29:80 to 10.1.0.0/24:*

Flow	10.0.0.7:80 - 10.1.0.2:3003　TCP
Aggregated Flow	10.0.0.0/29:80 – 10.1.0.0/24:* TCP

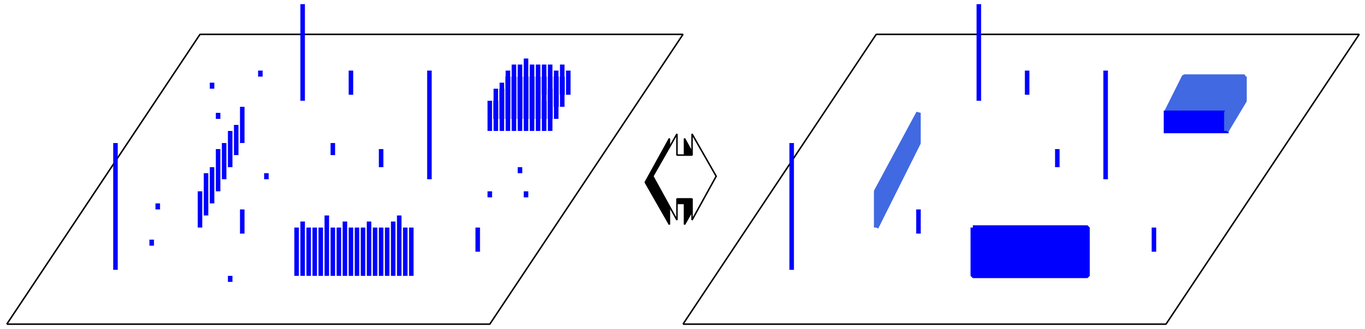
情報理論から見たフロー集約

- データ圧縮：画像圧縮との類似点
  - 高解像度 (情報量大) ↔ 低解像度 (情報量小)
  - 情報量（エントロピー）の符号化



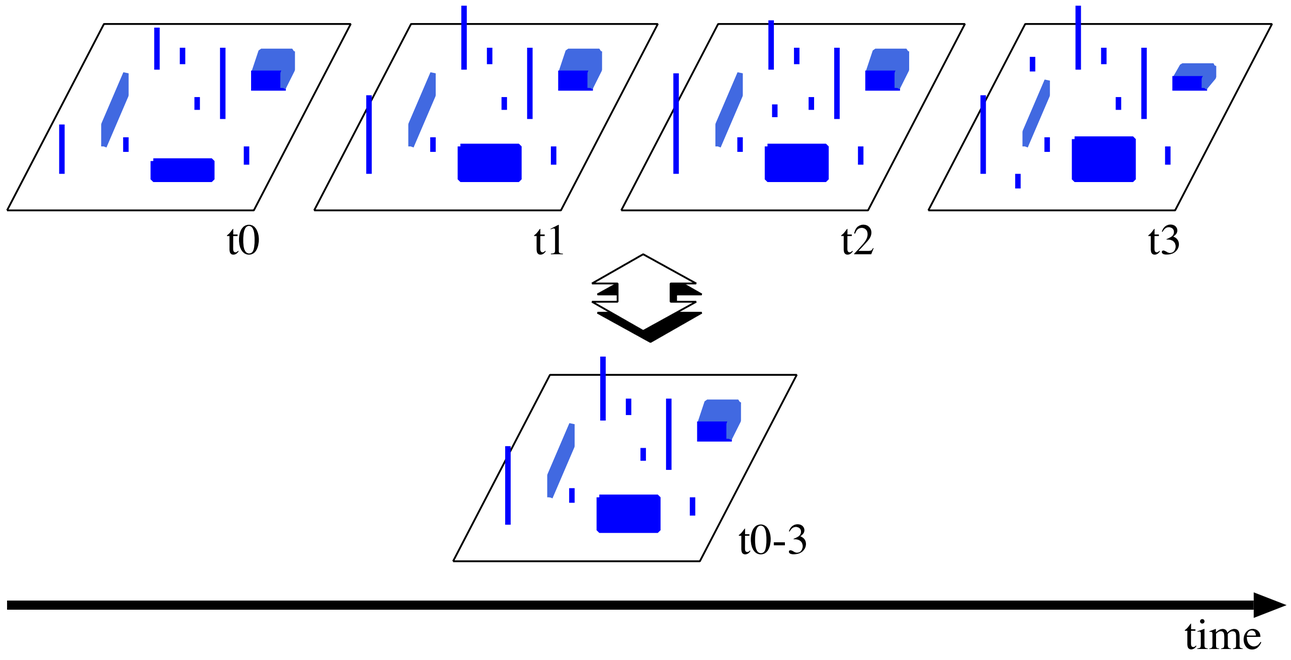
空間的フロー集約

- ２次元の例 (src_IP, dst_IP) （実際には５次元）
  - 値：トラフィック量（またはパケット量）
- 画像圧縮との違い
  - 空間に対して疎ら（空間的可視化は難しい）
  - 点の集合：限られた集約通信パターン
    - １対多：縦線分、多対１：横線分、多対多：矩形



時間方向への集約

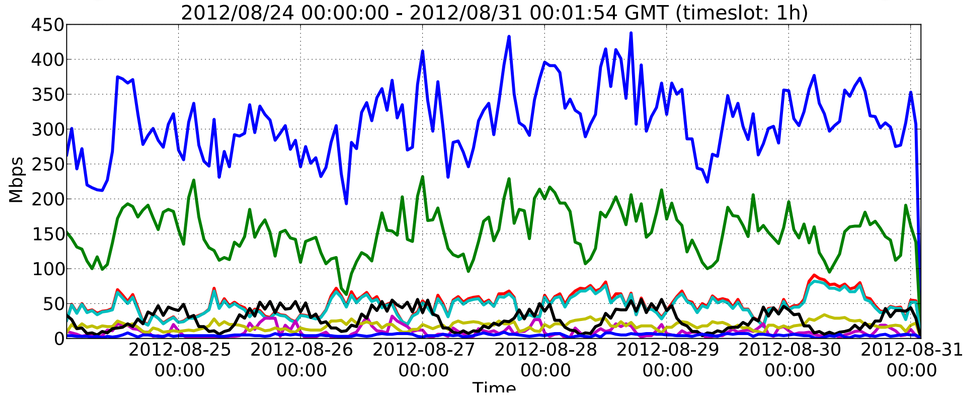
- 同じ考え方で時間方向の粒度を変えることが可能

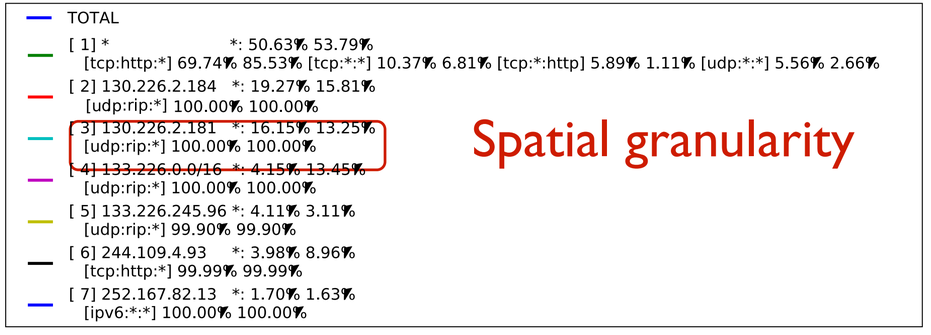


集約粒度変更

- 既存の集約フローの可視化ツールでは粒度変更は面倒
  - 処理速度と柔軟性のトレードオフ
- 自由に粒度変更可能なツールが欲しい
  - 時間方向：表示期間と時間粒度
  - 空間方向：フロー集約の粒度

Temporal granularity

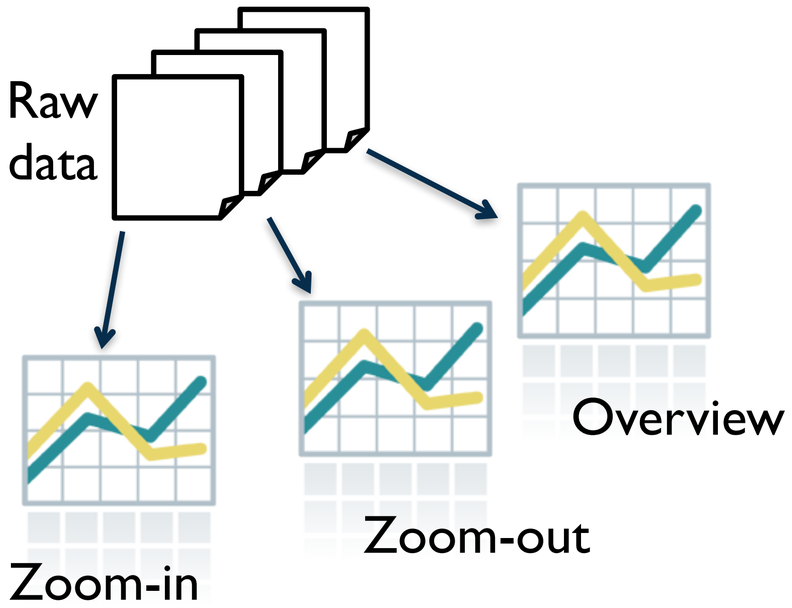




Spatial granularity

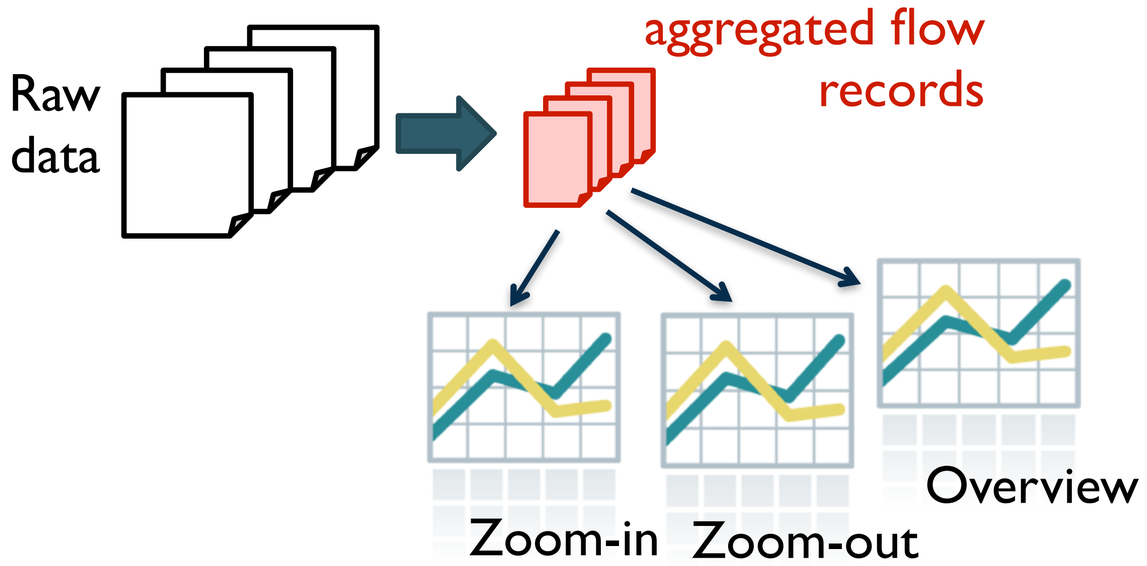
技術的チャレンジ

- 広大な多次元空間でフローを対話的にクラスタリング
  - 数百万フローから10個程度の集約フローを作成
- ビュー（粒度）を変更する際のオーバヘッドの削減



agurim

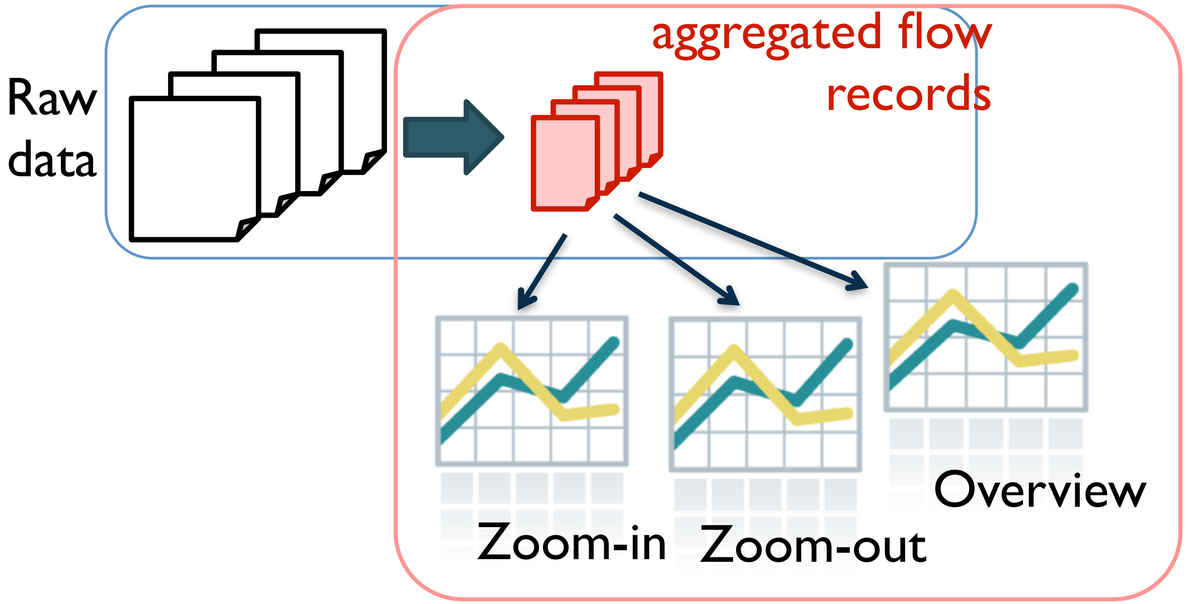
- 柔軟で効率良い多次元フロー集約ツール
  - 対話型の視覚化の実現
  - 効率：生データから再利用可能な細粒度集約フロー生成
  - 柔軟性：細粒度集約フローを再集約して、ビューに必要な集約フローを生成



agurimの概要：２段階フロー集約

一次集約：効率重視のフロー集約

再利用可能な細粒度集約フロー情報の生成



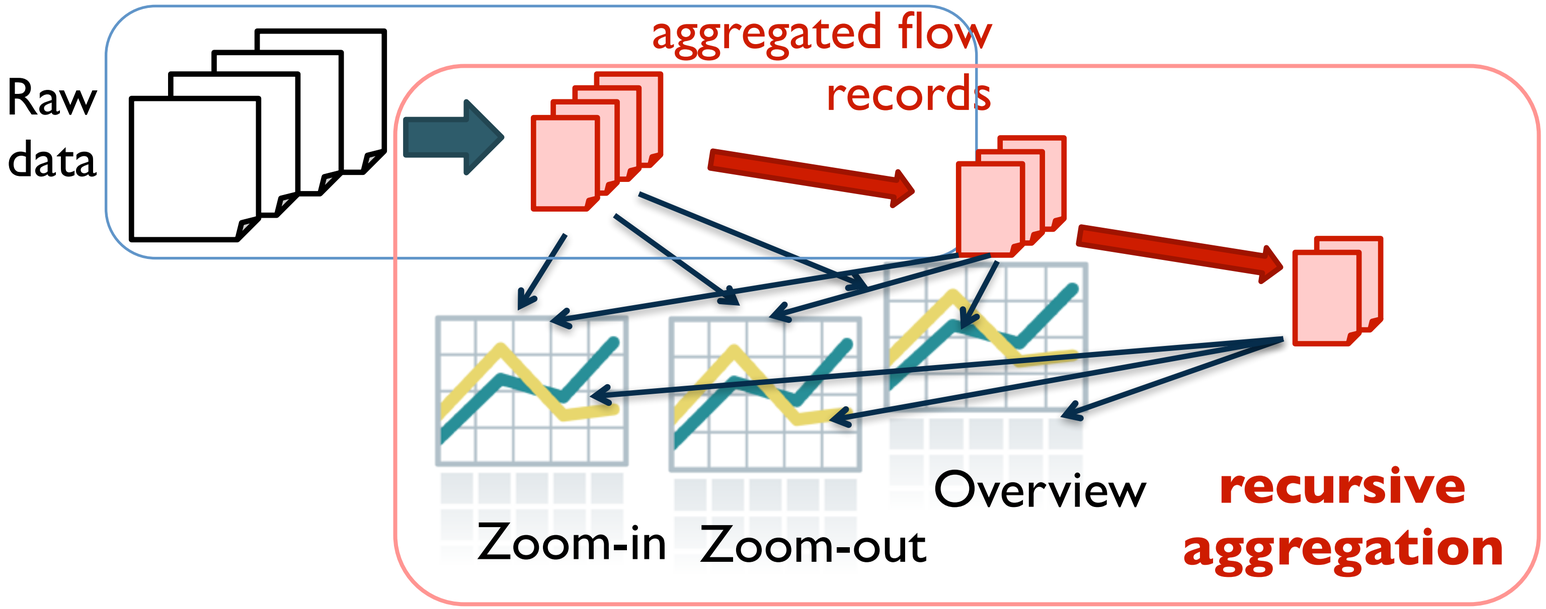
二次集約：表示用の柔軟性重視のフロー集約

プロットに必要な時間空間粒度で集約フロー情報を再集約

agurimの概要：多段階フロー集約

一次集約：効率重視のフロー集約

再利用可能な細粒度集約フロー情報の生成



二次集約：表示用の柔軟性重視のフロー集約

プロットに必要な時間空間粒度で集約フロー情報を再集約

再帰的集約：再集約結果をさらに集約

多次元フロー集約アルゴリズム

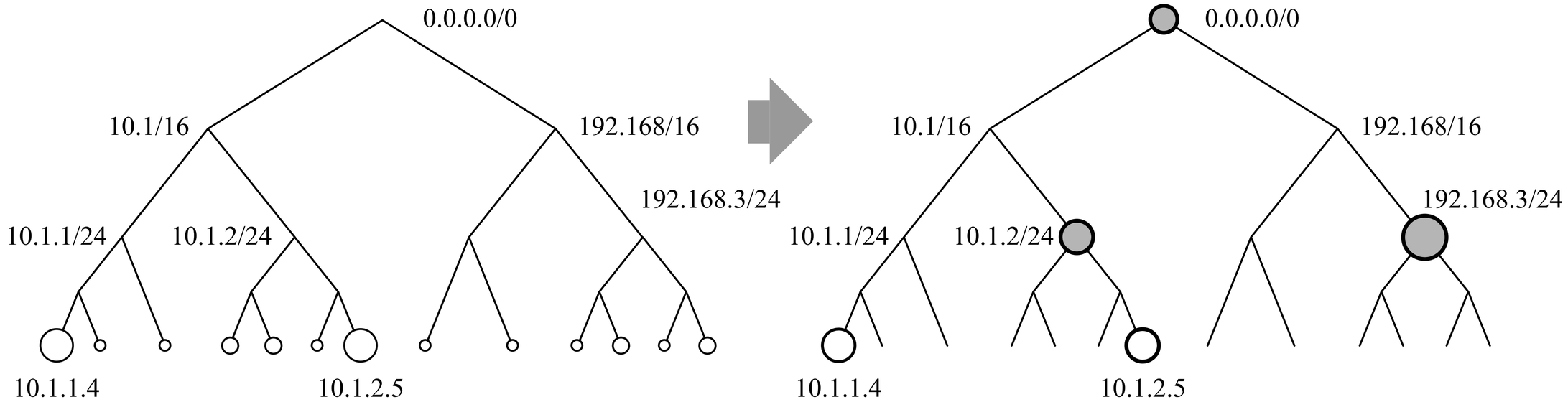
- 最近開発したアルゴリズム
  - 空間分割を使った効率的な手法

Hierarchical Heavy Hitters (HHHs)

- 階層ヘビーユーザ問題として多くの既存研究
  - 例えば (src, dst) IPアドレス
    - *(1.2.3.4, *)* → one-to-many: e.g., scanning
    - *(*, 5.6.7.8)* → many-to-one: e.g., DDoS
    - *(1.2.3.0/24, 4.5.6.0/28)* → subnet-to-subnet
  - 5タプルを使った多次元への拡張

HHH: 一次元の場合

- HHH: カウントが閾値を超える集約フロー: $c \geq \varphi N$
  - φ: 閾値　N: 総入力数 (パケット数やバイト数)
- HHHは深さ優先探索で一意に発見可能
  - 小さなフローを閾値を超えるまで集約する



HHH: 多次元の場合

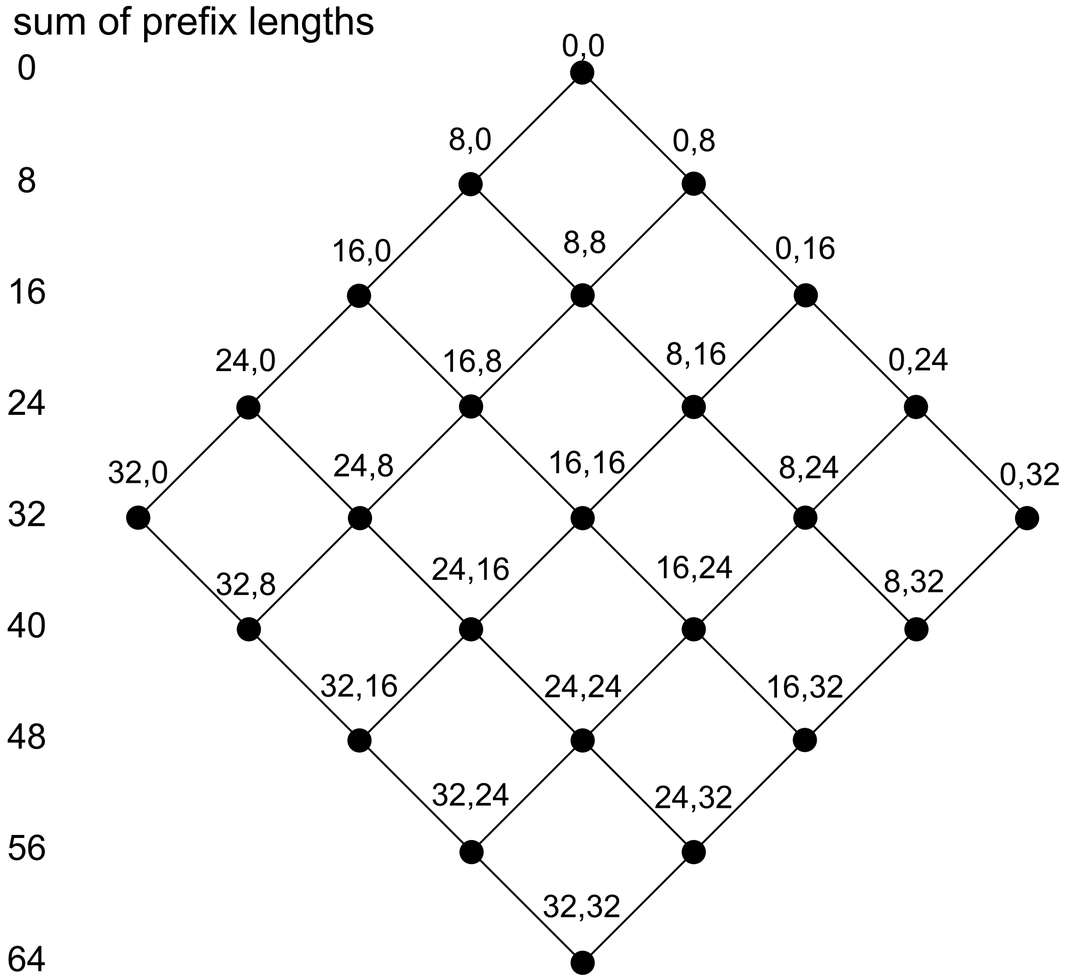
- 各ノードは複数の親ノードを持つ
  - 集約の組み合わせが多数存在
  - 一次元より大幅に難しい
- 二次元のIPv4アドレスペアの場合
  - バイト粒度だと 5×5=25
  - ビット粒度だと 33×33=1089

src: 1.2.3.4　　dst: 5.6.7.8

[1.2.3.4/32,5.6.0.0/16] [1.2.3.0/24,5.6.7.0/24] [1.2.0.0/16,1.2.3.4/32]

[1.2.3.4/32,5.6.7.0/24] [1.2.3.0/24,5.6.7.8/32]

[1.2.3.4/32,5.6.7.8/32]



Lattice for IPv4 prefix length pair with 8-bit granularity

技術課題

- 性能
  - ビット粒度の集約はコスト高
- 運用との整合性
  - 例えば、[32, *]と[16, 16]の優先度
  - 冗長で巨大な集約フロー: (128/4 と128/2など)
- 再集約
  - ズームイン・アウトのインタラクティブな操作

本研究の成果

- ビット粒度で効率の良いフロー集約アルゴリズム
  - 運用目的に合い、再集約も可能
- オープンソースのツール実装、オープンデータセット

- 広義には、既存の定義を見直すことによって、困難な問題に対する簡潔な解法が存在することを示したこと

HHHの定義に関して

- ディスカウントHHH ⬅ 本提案でも採用
  - 子孫HHHのカウントを除外する

$$c_i' = \sum_j c_j' \ \ where\ \{ j \in child(i) \mid c_j' < \varphi N \}$$

- ロールアップルール: 親ノードへのカウントの集約方法
  - オーバーラップ: すべてのHHHを検出するためにダブルカウントを許可
  - スプリット: カウントの保存 ⬅ 順序的に最初の親ノードにロールアップ
- 集約順序
  - プリフィックス長の和 ⬅ この順序を見直す

既存のアルゴリズム

- 構造の工夫
  - cross-producting, grid-of-trie, rectangle-search
- 理論解析
  - ストリーミング近似アルゴリズムと誤差範囲
- 既存手法はすべてボトムアップ

- 提案手法: トップダウンな決定的アルゴリズム
  - 複雑な構造も近似も必要としない

既存アルゴリズム：cross-producting

- まず、各次元独立にHHHを抽出し要素数を絞る
  - 各次元のHHH間のN次元配列を作る
- 入力を再走査、各次元で最長プリフィックスマッチにより対応する配列要素を求めカウント
- 最後に、小さい配列要素を集約

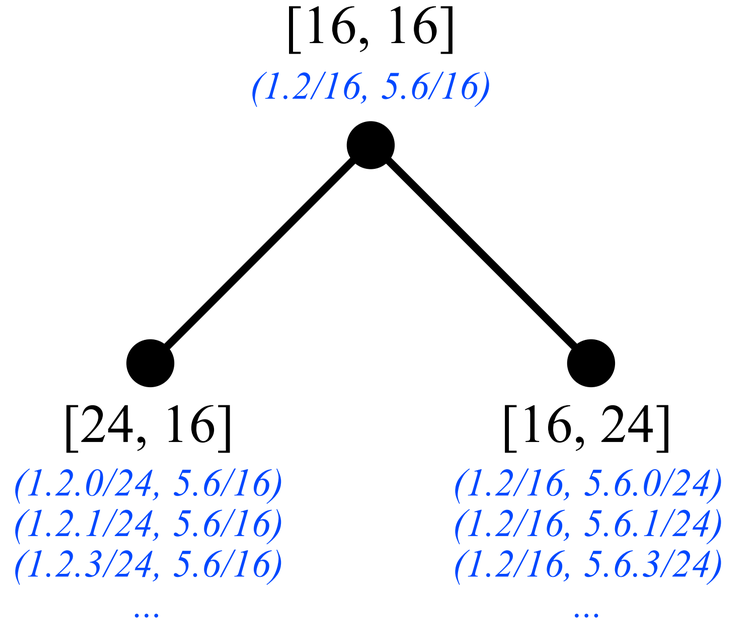
- 処理に２パス必要

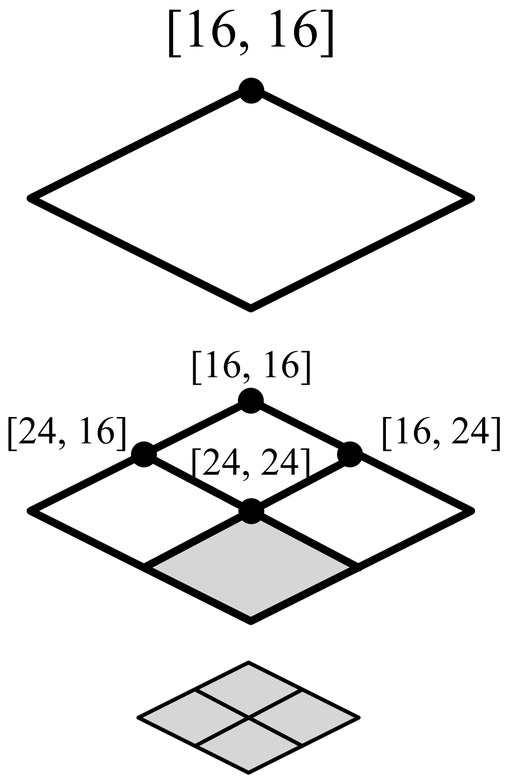
既存アルゴリズム：ストリーミングアルゴリズム

- 頻出アイテム検出近似アルゴリズム
  - n+αのカウンターでn個の最頻出要素を発見する
  - 最大誤差の上限が計算できる ➡ 理論解析
- HHHへの応用
  - 各集約レベル(格子ノード)に適用
- そのままだとネットワーク運用目的に使うには適さない

HHH再考

- アイデア: *child(i)* を再定義して空間分割を可能に
  - *child(i)*: bin-tree から quadtree へ

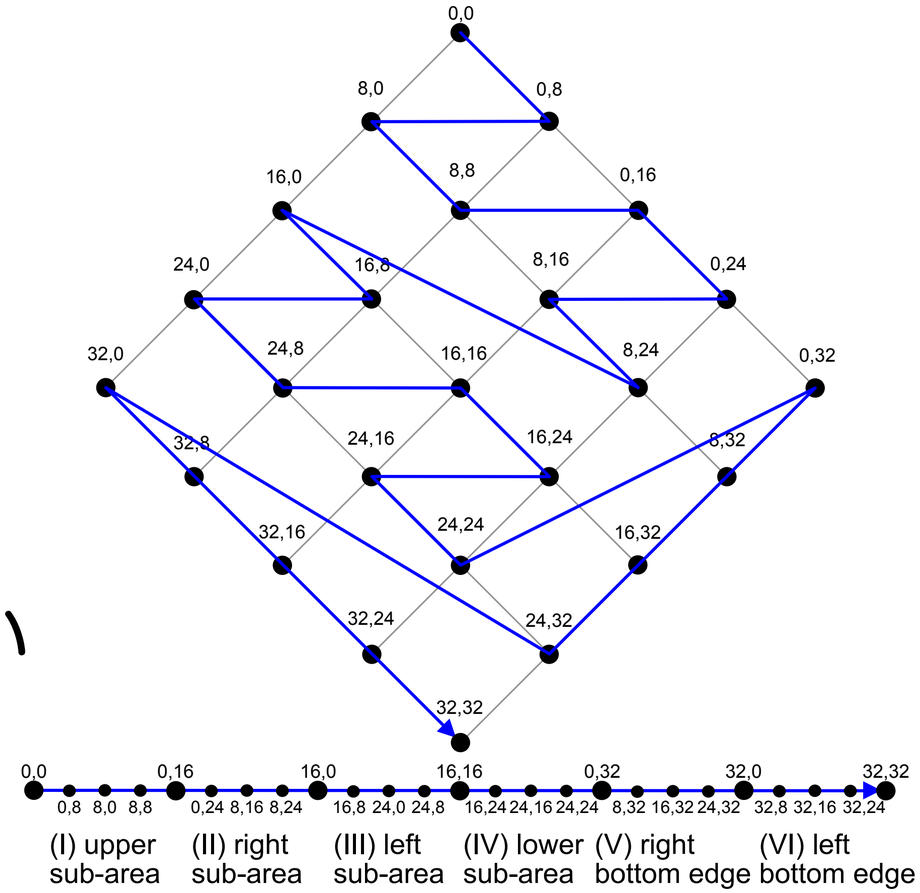




bottom-up aggregation **top-down space partitioning**

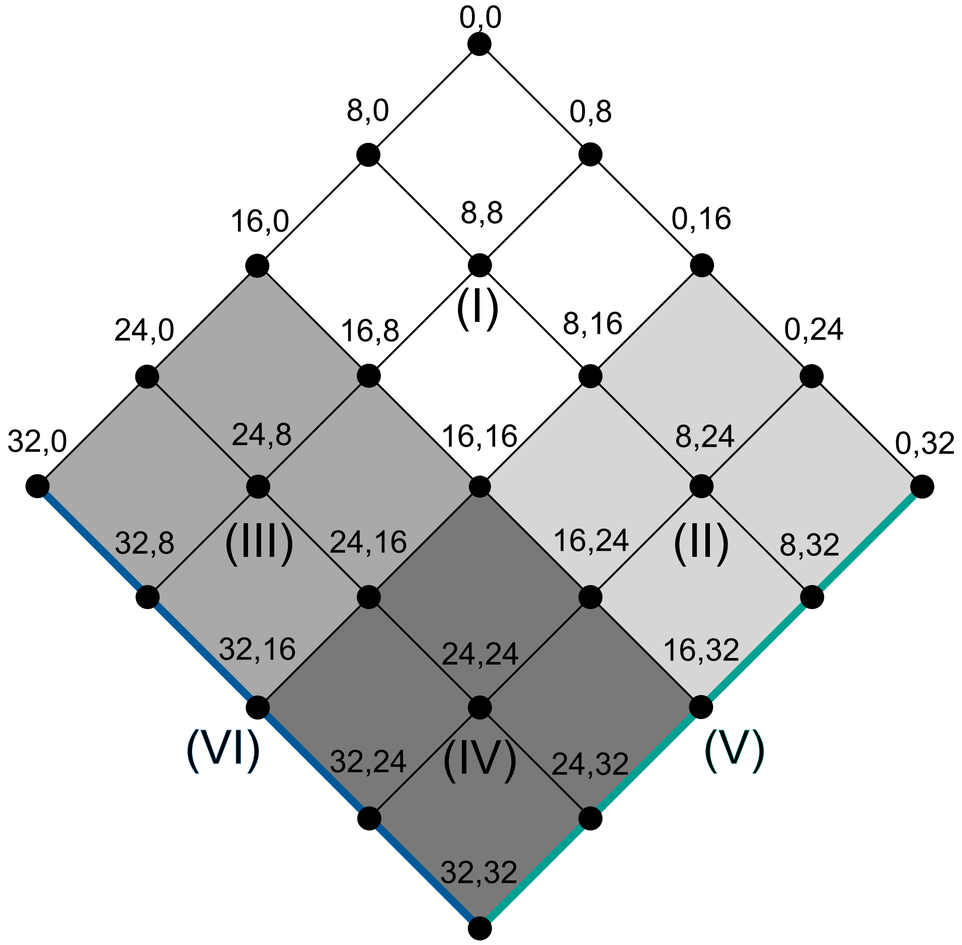
Z順序 [Morton1966]

- 空間充填曲線
  - ビットインターリービング (l_0, l_1)
- 次元間の最大値による順序
- 標準的なZ順序との違い
  - [0..32]は全5ビット空間に足りない
  - 結果、/32 の順序が高くなる



再帰的空間分割

- リージョン (VI) から (I) の順に探索
  - 2つの底辺
    - (VI) 左下エッジ
    - (V) 右下エッジ
  - 4つの四分木空間
    - (IV) 下四分木空間
    - (III) 左四分木空間
    - (II) 右四分木空間
    - (I) 上四分木空間

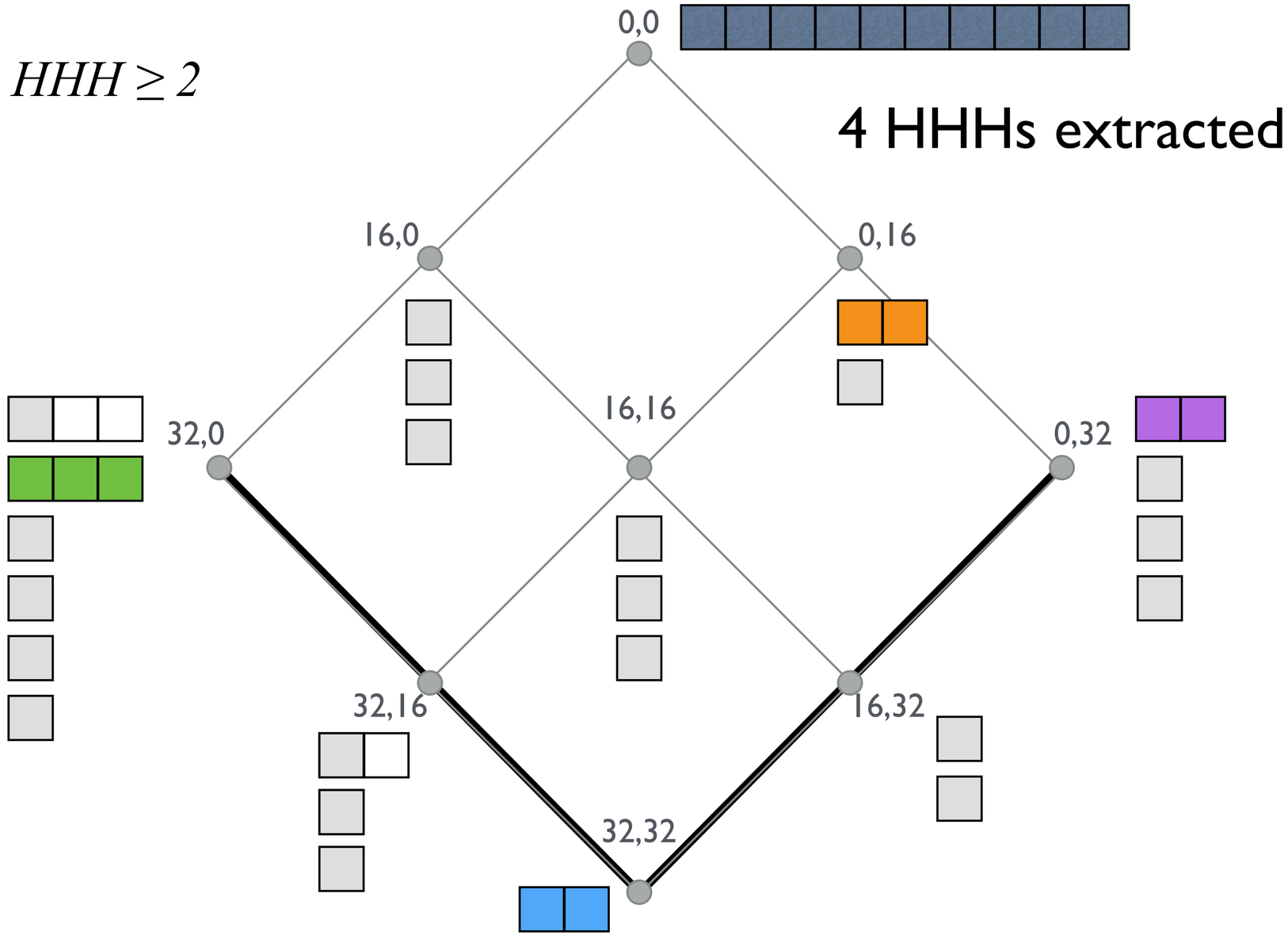


Recursive Lattice Search (RLS)

- アイデア: Z順序を使って再帰的に集約フローを分割
- 利点
  - 再帰の対象フローは閾値以上のフローのみ
  - 分割対象は親フローの要素フローのみ
- 欠点
  - 最初の次元が優先される偏向

RLSの動作例

10 inputs, HHH ≥ 2



評価

- シミュレータ: SpaceSaving [Mitzenmacher2012] のコードを流用
  - agurimのRLS実装をポート
  - データ: mawi パケットトレース 2016-10-20
- 対順序敏感性: (src,dst) vs. (dst,src)
  - 出力はほとんど同じ: 実用上問題なし
- SS (ストリーミングアルゴリズム、オーバラップロールアップ)との比較
  - そもそも定義が変わっているため、主要な違いを示す
  - 出力: よりコンパクト、必要な要素を含む
  - 性能:ビット粒度の集約で約100倍高速

対順序敏感性 (src,dst) vs. (dst,src)

region	no	aggregated by (src,dst) src	dst	$c'/N(\%)$
VI	(1)	112.31.100.1/32	163.229.97.230/32	16.5
V	(2)	64.0.0.0/2	202.203.3.13/32	5.2
	(3)	128.0.0.0/1	202.203.3.13/32	5.8
	(4)	*	202.26.162.46/32	6.0
III	(5)	163.229.96.0/23	*	5.0
	(6)	203.179.128.0/20	*	6.8
II	(7)	*	202.203.3.0/24	5.9
	(8)	*	203.179.140.0/23	5.7
	(9)	*	163.229.128.0/17	5.1
I	(10)	0.0.0.0/1	202.192.0.0/12	5.3
	(11)	202.192.0.0/12	*	6.7
	(12)	*	202.0.0.0/7	7.6
	(13)	**128.0.0.0/4**	*****	**5.0**
	(14)	**128.0.0.0/2**	*	**6.0**
	(15)	*	**128.0.0.0/2**	**5.4**
-		*	*	**2.0**
				100.0
		aggregated by (dst,src)		
	(1)-(12)	identical to (src,dst)		
I	(13)	**128.0.0.0/2**	**0.0.0.0/2**	**5.7**
	(14)	*	**128.0.0.0/3**	**5.3**
	(15)	**128.0.0.0/1**	*	**6.4**
-		*	*	**1.0**

- (1)-(12): 完全一致
- (13)-(15): 僅かな違い
- src-dst 順の影響は僅か

RLS vs. SS 出力の比較

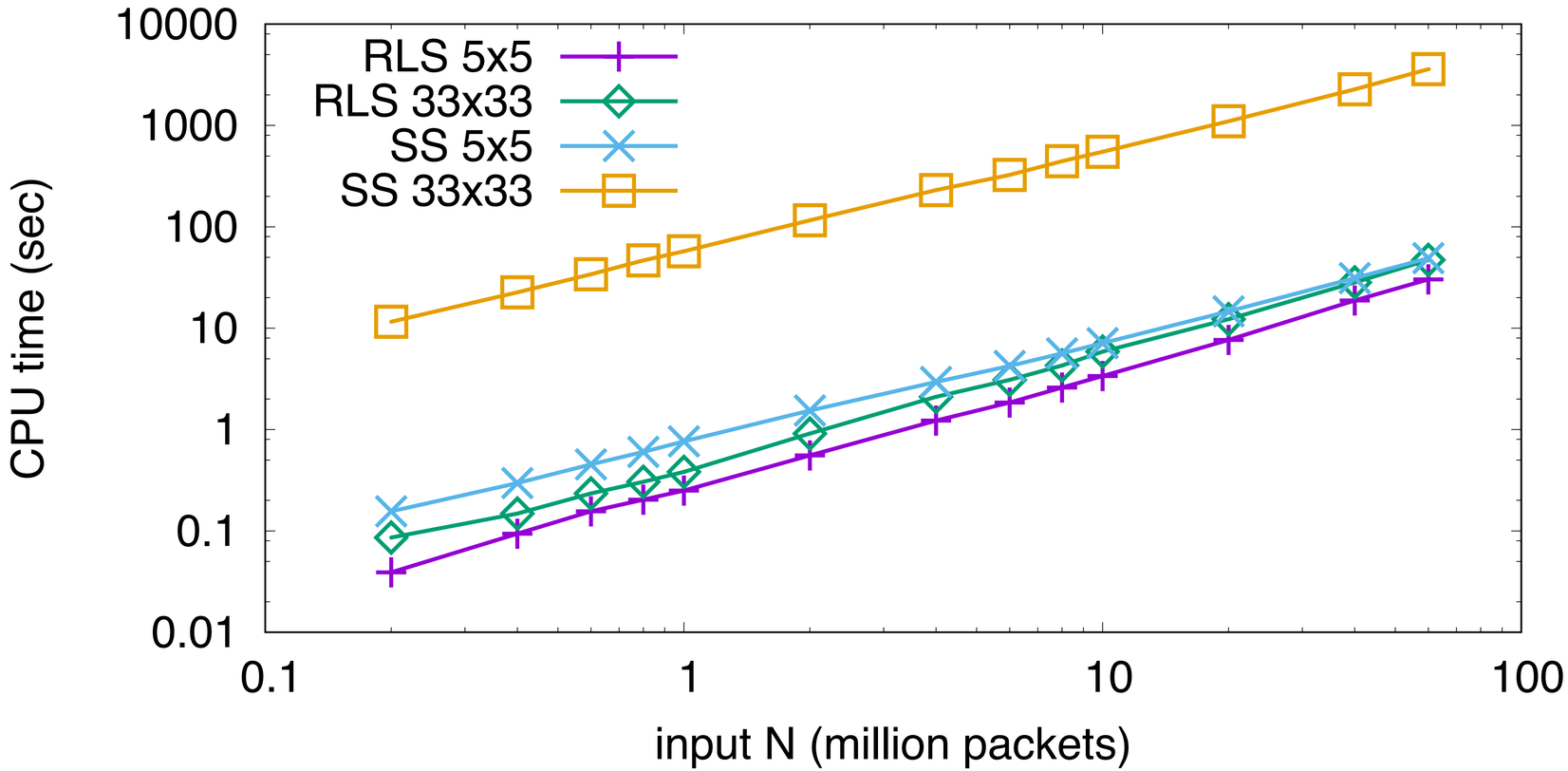
- HHHの数: RLS:15, SS:52
- SSのHHH: 冗長な情報多い
  - ダブルカウントや広域フロー
  - 40の省略フロー, 35 (I), 4 (II), 1 (III)
- RLSは簡素な出力

region	no	aggregated by (src,dst) src	dst	$c'/N(\%)$
VI	(1)	112.31.100.1/32	163.229.97.230/32	16.5
V	(2)	64.0.0.0/2	202.203.3.13/32	5.2
	(3)	128.0.0.0/1	202.203.3.13/32	5.8
	(4)	*	202.26.162.46/32	6.0
III	(5)	163.229.96.0/23	*	5.0
	(6)	203.179.128.0/20	*	6.8
II	(7)	*	202.203.3.0/24	5.9
	(8)	*	203.179.140.0/23	5.7
	(9)	*	163.229.128.0/17	5.1
I	(10)	0.0.0.0/1	202.192.0.0/12	5.3
	(11)	202.192.0.0/12	*	6.7
	(12)	*	202.0.0.0/7	7.6
	(13)	**128.0.0.0/4**	*****	**5.0**
	(14)	**128.0.0.0/2**	*	**6.0**
	(15)	*	**128.0.0.0/2**	**5.4**
-		*	*	**2.0**
				100.0

no	RLS(%)	SS(%)	missing SS HHHs with their $c'/N(\%)$
(1)	16.5	16.5	-
(2)	5.2	5.2	-
(3)	5.8	5.8	-
(4)	6.0	6.0	-
(5)	5.0	5.0	-
(6)	6.8	6.8	-
(7)	**5.9**	**16.9**	-
(8)	5.7	5.7	-
(9)	5.1	5.1	-
(10)	**5.3**	**-**	(96/**3**,202.203/**16**):5.4 (0/**2**,202.203/**16**):5.6 (112/**4**,202.192/**12**):5.2 (64/**2**,202.192/**12**):9.0
(11)	6.7	6.7	-
(12)	**7.6**	**-**	(0/**1**,203.179.128/**20**):6.0 (128/**2**,202.203/**16**):5.5 (192/**4**,202/**8**):5.1 (*****,202.192/**12**):25.5 (16/**4**,202/**7**):5.4 (128/**1**,202.128/**9**):10.6 (64/**2**,202/**7**):15.5 (128/**1**,202/**7**):17.7
(13)	**5.0**	**5.2**	-
(14)	**6.0**	**-**	(163.229/**16**,0/**1**):6.0 (144/**4**,128/**1**):5.3 (128/**2**,96/**3**):5.0 (128/**3**,0/**1**):5.3 (160/**3**,128/**1**):7.0 (128/**2**,0/**2**):5.7 (128/**2**,0/**1**):11.4
(15)	**5.4**	**33.1**	(128/**1**,160/**6**):5.0 (192/**4**,128/**2**):5.2 (0/**1**,128/**2**):22.7 (*****,128/**3**):7.1
-	**2.0**	**-**	(202/**7**,0/**2**):5.4 (192/**8**,128/**1**):5.6 (202/**8**,0/**1**):5.7 (202/**7**,128/**1**):6.0 (192/**3**,200/**5**):10.5 (128/**1**,112/**6**):5.1 (112/**5**,128/**1**):21.8 (200/**5**,*****):17.0 (192/**4**,128/**1**):13.6 (128/**1**,16/**4**):6.2 (*****,200/**5**):42.4 (64/**3**,128/**1**):6.0 (96/**3**,128/**1**):29.7 (128/**1**,64/**2**):10.4 (0/**1**,128/**1**):46.7 (128/**1**,*****):53.3 (*****,128/**1**):78.3

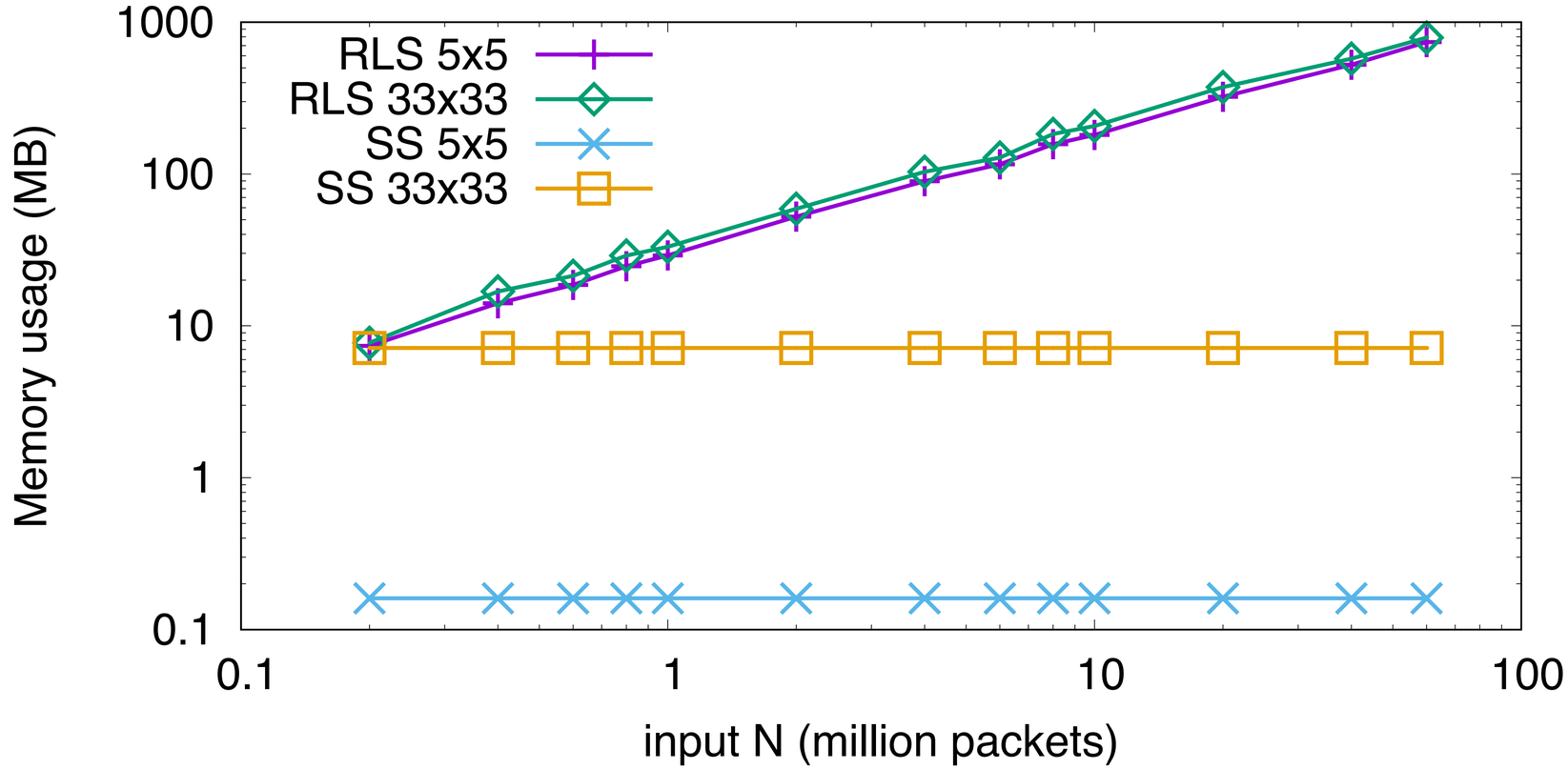
CPU時間: RLS vs. SS

- RLS: 粒度が細かくなると有利
  - ビット粒度だと100倍高速



メモリ利用: RLS vs. SS

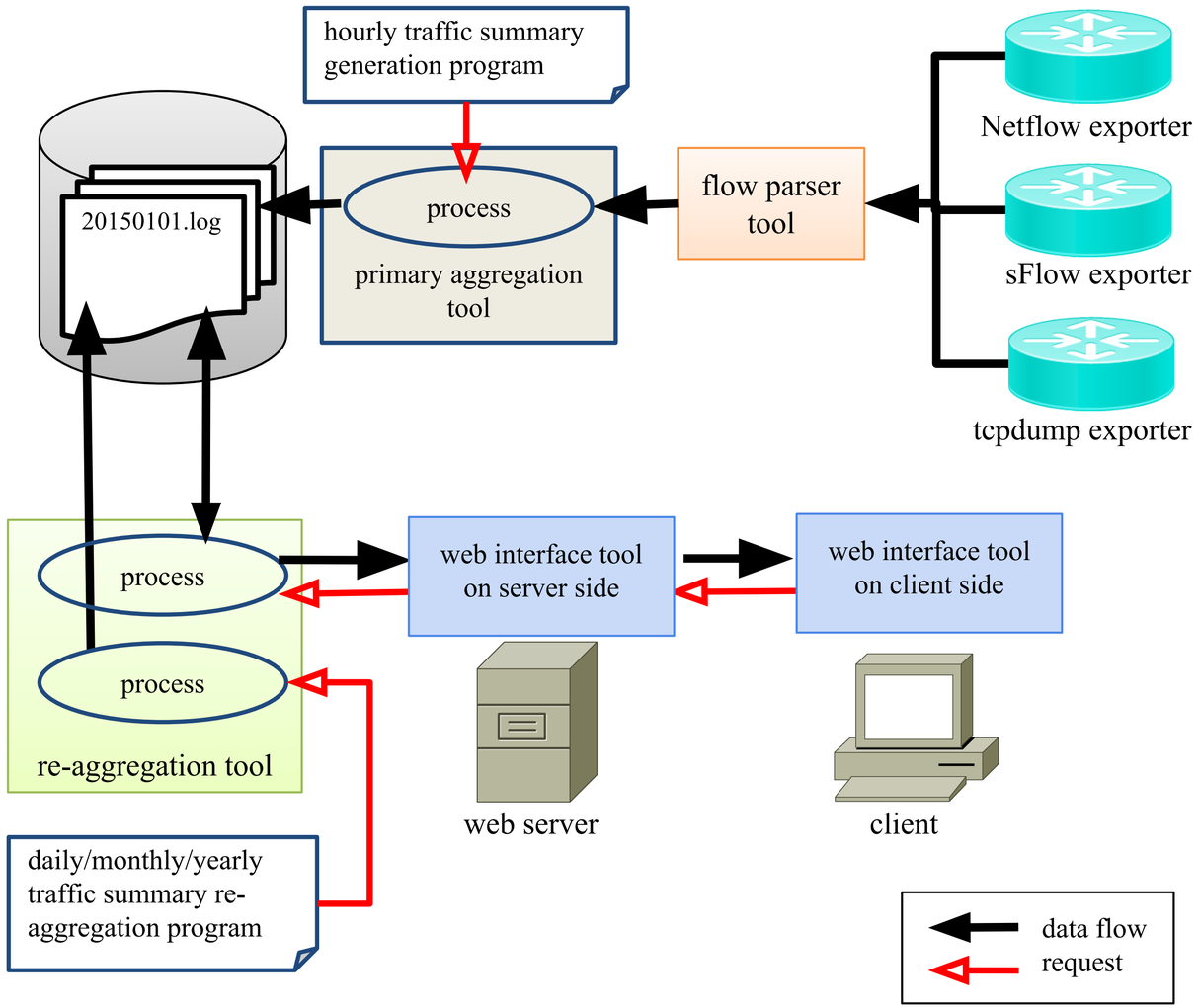
- RLS: 入力に比例 (いまどきのPCなら問題小)
- SS: 入力に関わらず一定



agurimのRLS実装

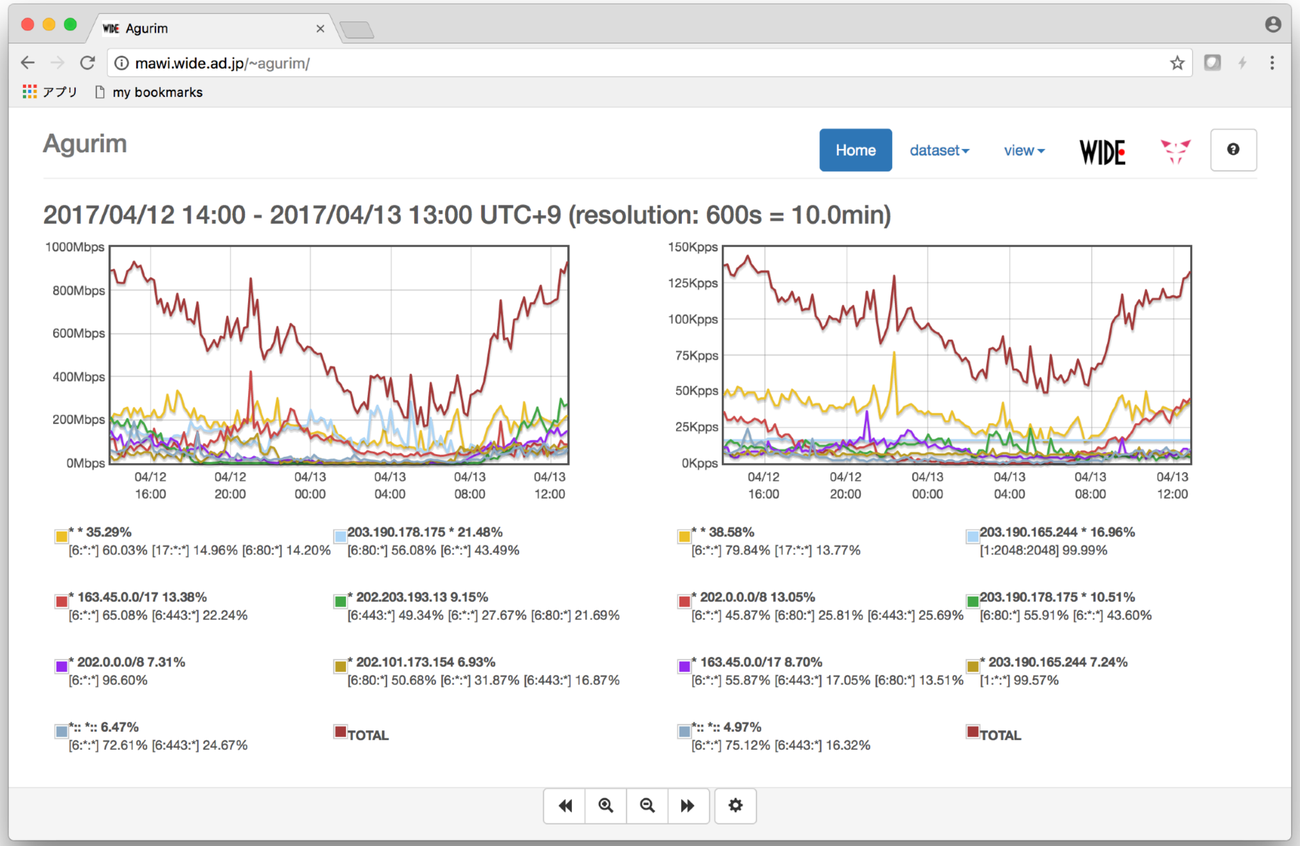
- 2段構成 HHH
  - 主属性 (src-dst addrs), 副属性 (src-dst ports)
- プロトコル最適化
  - 部分的に再帰の深さを制御して粒度調整
- マルチコアCPUを利用したリアルタイム処理

agurimシステム構成



agurim Web UI

- オープンデータ: 2013年からのWIDEトランジットリンク



http://mawi.wide.ad.jp/~agurim/

コードとデータの情報

- ソースコード公開 (2015/05-)
  - aguri3: 一次集約ツール
  - agurim: 二次集約ツールおよびWeb UI
    - http://mawi.wide.ad.jp/~agurim/about.html
- データ公開 (2015/05-)
  - 2013/02からのWIDEのトランジットトラフィック
  - IPアドレスは匿名化
    - http://mawi.wide.ad.jp/~agurim/

まとめ

- フロー集約：トラフィック監視や異常検出のための要素技術
- agurim：フロー集約のプロトタイプ実装
  - 多次元フロー集約により柔軟な集約粒度変更を実現
  - プロトタイプの性能：クリックから1秒以内での描画達成
- Recursive Lattice Search
  - 空間分割による多次元フロー集約アルゴリズム
    - Kenjiro Cho. “Recursive Lattice Search: Hierarchical Heavy Hitters Revisited”. ACM IMC 2017, London, UK, November 2017.
    - Midori Kato, Kenjiro Cho, Michio Honda, Hideyuki Yokuda. “Monitoring the Dynamics of Network Traffic by Recursive Multi-dimensional Aggregation”. OSDI2012 MAD Workshop. Hollywood, CA. October 2012.