Robustness of Class-Based Path-Vector Systems

Vijay Ramachandran
http://www.cs.yale.edu/~vijayr

Joint work with
Aaron Jaggard
http://www.math.tulane.edu/~adj

Supported by the DoD URI program
under ONR grant N00014-01-1-0795
Overview

Formal-model approach to analyzing the behavior of inter-domain routing protocols

Robustness of Class-Based Path-Vector Systems

One of several design goals for routing protocols: Convergence even in the presence of link and node failures

A generalization of the hierarchical inter-AS relationships introduced by Gao & Rexford [TON '01]

A formal model of path-vector routing protocols, which communicate destinations’ reachability information on a hop-by-hop basis depending on domain policies
Inter-Domain Path-Vector Routing
[BGPv4—RFC 1771]
Routing Divergence
[Griffin, Shepherd, Wilfong—ICNP ‘99]
Hierarchical-BGP Example

[Gao, Rexford—TON '01]

A

Tier-1 Provider

BCD

ISP

BD

ISP

ABCD

ACD

BCD

ISP

BD

ISP

BD

Peering

Customer-Provider

Multihomed Customer

CBD

CD
Class-Based Systems
[GJR—SIGCOMM ’03 / JR—ICNP ’04]

An application of the PVPS framework:

\[(PV, PL, K)\]

Path-Vector System:
Level: non-decreasing, shared
Local pref.: unconstrained, opaque
Rank: (level, -pref)

Policy Language:
Nodes can classify neighbors and change attributes according to preference and scoping rules.

Global Constraint:
No customer-provider cycles.

Properties [GJR03]:
This system’s expressiveness, autonomy, and transparency implies a non-trivial global constraint.
**HBGP Class Description**

[JR '04 Ex. 2.2]

**Classes**

\[ C = \{ C_1 = 'customer,', \quad C_2 = 'peer,,' \quad C_3 = 'provider' \} \]

**Consistency**

\[
\begin{pmatrix}
1 & 0 & 0 & 1 \\
2 & 0 & 1 & 0 \\
3 & 1 & 0 & 0
\end{pmatrix}
\]

**Preference**

\[
\begin{pmatrix}
1 & 0 & -1 & -1 \\
2 & 1 & 0 & -1 \\
3 & 1 & 1 & 0
\end{pmatrix}
\]

**Scoping**

\[
\begin{pmatrix}
1 & 1 & 1 \\
2 & 1 & 0 & 0 \\
3 & 1 & 0 & 0
\end{pmatrix}
\]
Scoping and preference rules can preclude a dispute wheel.

Level attribute of paths at rim must be equal.
Dispute Predicate
[JR '04, Def. 3.14]

\[ P(C_a, C_b) \iff (M_{ab} = 1) \lor (\exists C_c \forall m^{-1}(C_b) : W_{ca} \neq -1) \]
Robustness Constraint
[JR '04, Thm. 3.15]

- In an instance:
  - any cycles where all pairs of adjacent edges have class assignments satisfying $P$ are potential dispute wheels.
  - without any such cycles, there can be no dispute wheel, so the instance is robust.
- Generation of constraint naively takes $O(|C|^3)$ time.
Constraint Enforcement (Centralized)  
[JR '04, Sec. 4.1]

Running time is $O(\Delta|E|)$. 

$P( c(B,C), c(B,D) )$

$P( c(B,D), c(B,C) )$

$P( c(C,B), c(C,B) )$

$P( c(B,C), c(B,C) )$

$P( c(C,D), c(C,B) )$

$P( c(B,D), c(B,C) )$

$P( c(C,B), c(C,D) )$

$P( c(B,C), c(B,D) )$

$P( c(B,D), c(B,C) )$

$P( c(C,B), c(C,D) )$
Constraint Enforcement (Distributed)
[JR '04, Sec. 5.1]

Depref \( w \) or filter from \( w \) or filter to \( v \)
Distributed Algorithm Summary

[JR '04, Sec. 5.1]

- Decides whether one directed signaling edge (because of its class assignments) can participate in a dispute-wheel rim.

- The tail node sends a token which is forwarded through the network along pairs of edges satisfying $P$.

- If the tail receives its own token, it knows the edge is problematic, as the token has traveled through a cycle violating the constraint.
Distributed Algorithm Properties
[JR '04, Sec. 5.1]

- **Termination**: 
  - A token traverses each edge only once.
  - All token-traversal paths end.

- **Number of messages**: 0 or 3 per edge

- **Privacy**: 
  - Don't need to reveal starting node.
  - If a node receives 1 message, it receives all 3.

- **Correctness**: Cycles are potential dispute wheels and tail node can tweak policies.
The algorithms can be used to check robustness of networks using arbitrary next-hop preferences.

[Sob03] gives a centralized algorithm that is sometimes faster (when $C < \Delta$) but produces more false positives.

[GW00] gives a distributed algorithm that works while routing but increases routing-message size.
Conclusions and Future Work

- Completed work on the application of [GJR03] to class-based systems
  - Generate constraint from class description
    - If constraint holds, instance is robust.
    - If not, instance may have a legal, policy-induced oscillation.
  - Algorithms to enforce the constraint (centralized and distributed)
- Constraints for more general cases?
- Better distributed algorithm?