Don’t Mind the Gap:
Bridging Network-wide Objectives and Device-level Configurations

Ryan Beckett (Princeton, MSR)
Ratul Mahajan (MSR)
Todd Millstein (UCLA)
Jitu Padhye (MSR)
David Walker (Princeton)
Configuring Networks is Error-Prone

~60% of network downtime is caused by human error

- Yankee group 2002

50-80% of outages from configuration changes

- Juniper 2008
Configuring Networks is Error-Prone

YouTube/Pakistan incident: Could something similar whack your site?

Configuring BGP properly is key to avoidance, 'Net registry official says

By Carolyn Duffy Marsan
Network World | Mar 10, 2008 1:00 AM PT

In light of Pakistan Telecom/YouTube incident, Internet registry official explains how you can avoid having your web site victimized by such an attack.

When Pakistan Telecom blocked YouTube's traffic one Sunday evening in February, the ISP created an international incident that wreaked havoc on the popular video site for more than two hours.

RIPE NCC, the European registry for Internet addresses, has conducted an analysis of what happened during Pakistan Telecom's hijacking of YouTube's traffic and the steps that YouTube took to stop the attack.

We posed some questions to RIPE NCC's Chief Scientist Daniel Karrenberg about the YouTube incident. Here's what he had to say:

How frequently do hijacking incidents like the Pakistan Telecom/YouTube incident happen?

Misconfigurations of IBGP (internal BGP, the protocol used between the routers in the same Autonomous System) happen regularly and are usually the result of an error. One such misconfiguration caused the Pakistan Telecom/YouTube incident. It appears that the Pakistan Telecom/YouTube incident was not an "attack" as some have labeled it, but a configuration error. (See Columnist Johna Till Johnson's take on the topic.)

What is significant about the YouTube incident?
Configuring Networks is Error-Prone

By Carolyn Duffy Marsan

Network World | Mar 10, 2008 1:00 AM PT

In light of Pakistan Telecom/YouTube incident, you ask yourself: How can this happen to your web site victimized by such an unfriendly hijacking?

When Pakistan Telecom blocked YouTube and other sites in Pakistan last month, it raised an international incident that weakened the Internet's credibility as a reliable platform for content delivery. The incident demonstrated the flaws of RIPE NCC, the European registry for Internet addresses, which was unable to react in a timely manner to the hijacking incident caused by Pakistan Telecom/YouTube incident.

We posed some questions to RIPE NCC. Here's what he had to say:

How frequently do hijacking incidents occur?

Misconfigurations of IBGP (Interior BGP) Autonomous System) happen regularly. A common misconfiguration caused the Pakistan Telecom/YouTube incident was no checking of the BGP route under which the hijacking took place.

What is significant about the YouTube/Pakistan incident?

Since the incident was not resolved by the local Internet exchange, it was triggered by a hijack that took place. The hijack was carried out by a third party who hijacked the traffic through Pakistan Telecom. The Pakistan Telecom/YouTube incident was not resolved until the traffic was rerouted through a different path.

Figure 1: Supply chain portal with limited availability during Time Warner Cable outage

I recall one right away that users were unable to reach the supply chain portal indicating the issue was in the Time Warner network. In this case, I'd expect to see a brief service disruption while all traffic moved through their other upstream ISP AT&T. However, the availability issues continued for the entire duration of the outage. I was surprised to see such an issue, so took a look at the path calculation view to figure out exactly where traffic was getting dropped. As a result of the event, normally, two locations (Java and Dallas) were split back after the outage, which was not the case (Figure 2).
Configuring Networks is Error-Prone
Configuring Networks is Error-Prone
Fundamental Tradeoff?

<table>
<thead>
<tr>
<th>Control Mechanism</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distributed</td>
</tr>
<tr>
<td>Centralized</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distributed</td>
</tr>
<tr>
<td>Centralized</td>
</tr>
</tbody>
</table>
Fundamental Tradeoff?

<table>
<thead>
<tr>
<th>Control Mechanism</th>
<th>Distributed</th>
<th>Centralized</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distributed</td>
<td>OSPF</td>
<td>RIP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Scalability</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Configuration
Fundamental Tradeoff?

- Distributed Control Mechanism
- Centralized Configuration

- OSPF
- RIP
- BGP

- Scalability
- Robustness
- Complexity

100,000s of lines of config
Fundamental Tradeoff?

<table>
<thead>
<tr>
<th>Control Mechanism</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distributed</td>
<td>Distributed</td>
</tr>
<tr>
<td>Centralized</td>
<td>Centralized</td>
</tr>
</tbody>
</table>

- **OSPF**
  - Scalability
  - Robustness
  - Complexity

- **RIP**

- **BGP**

- **SDN**
  - Scalability
  - Robustness
  - Complexity
Fundamental Tradeoff?

<table>
<thead>
<tr>
<th>Distributed</th>
<th>Centralized</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OSPF</strong></td>
<td><strong>SDN</strong></td>
</tr>
<tr>
<td><strong>RIP</strong></td>
<td></td>
</tr>
<tr>
<td><strong>BGP</strong></td>
<td></td>
</tr>
</tbody>
</table>

Control Mechanism:
- Distributed
- Centralized

Configuration:
- Distributed
- Centralized

Ideal:
- Scalability
- Robustness
- Complexity

SDN:
- Scalability
- Robustness
- Complexity
Propane Overview
Propane System

I) Language for expressing network-wide objectives with:

- Path **constraints** and **preferences** in case of failures
- Uniform abstractions for **intra-** and **inter-**domain routing
Propane System

2) Compiler for a purely distributed implementation
Propane System

2) Compiler for a purely distributed implementation

- Generate BGP configs for each router
- Compiler guarantees policy-compliance for all failures
Example: A DC network with traditional configs

Goals

- Local prefixes reachable only internally
- Global prefixes reachable externally
- Aggregate global prefixes as GP
- Prefer leaving through Peer_1 over Peer_2
- Prevent transit traffic between peers
Example: A DC network with traditional configs

**Goals**
- Local prefixes reachable only internally
- Global prefixes reachable externally
- Aggregate global prefixes as GP
- Prefer leaving through Peer$_1$ over Peer$_2$
- Prevent transit traffic between peers

**Configuration Attempt**
- Don’t export from G, H to external
- Aggregate externally as GP
**Example: A DC network with traditional configs**

### Goals
- Local prefixes reachable only internally
- Global prefixes reachable externally
- Aggregate global prefixes as GP
- Prefer leaving through Peer\(_1\) over Peer\(_2\)
- Prevent transit traffic between peers

### Configuration Attempt
- Don’t export from G, H to external
- Aggregate externally as GP
Example: A DC network with traditional configs

Goals

- Local prefixes reachable only internally
- Global prefixes reachable externally
- Aggregate global prefixes as GP
- Prefer leaving through Peer₁ over Peer₂
- Prevent transit traffic between peers

Configuration Attempt

- Don’t export from G, H to external
- Aggregate externally as GP
- X,Y block routes through each other
Example: A DC network with traditional configs

**Goals**

- Local prefixes reachable only internally
- Global prefixes reachable externally
- Aggregate global prefixes as GP
- Prefer leaving through Peer\(_1\) over Peer\(_2\)
- Prevent transit traffic between peers

**Configuration Attempt**

- Don’t export from G, H to external
- Aggregate externally as GP
- X,Y block routes through each other
**Example: A DC network with traditional configs**

**Goals**
- Local prefixes reachable only internally
- Global prefixes reachable externally
- Aggregate global prefixes as GP
- Prefer leaving through Peer₁ over Peer₂
- Prevent transit traffic between peers

**Configuration Attempt**
- Don’t export from G, H to external
- Aggregate externally as GP
- X,Y block routes through each other
Example: A DC network with traditional configs

Goals

- Local prefixes reachable only internally
- Global prefixes reachable externally
- Aggregate global prefixes as GP
- Prefer leaving through Peer\(_1\) over Peer\(_2\)
- Prevent transit traffic between peers

Configuration Attempt

- Don’t export from G, H to external
- Aggregate externally as GP
- X,Y block routes through each other

Aggregation-Induced Black Hole!
**Example: A DC network with traditional configs**

**Goals**
- Local prefixes reachable only internally
- Global prefixes reachable externally
- Aggregate global prefixes as GP
- Prefer leaving through Peer\(_1\) over Peer\(_2\)
- Prevent transit traffic between peers

![Diagram](image-url)
Example: A DC network with Propane

```plaintext
define Destination =
{GP1  =>  end(A)
 GP2  =>  end(B)
 LP1  =>  end(E)
 LP2  =>  end(F)
 true  =>  exit(Peer1 >> Peer2)}
```
**Example: A DC network with Propane**

```plaintext
define Destination =
{GP1  =>  end(A)
 GP2  =>  end(B)
 LP1  =>  end(E)
 LP2  =>  end(F)
 true  =>  exit(Peer1 >> Peer2)}

define Locality =
{LP1  |  LP2  =>  internal}
```
**Example: A DC network with Propane**

```plaintext
define Destination =
{GP1 => end(A)
 GP2 => end(B)
 LP1 => end(E)
 LP2 => end(F)
 true => exit(Peer1 >> Peer2)}

define Locality =
{LP1 | LP2 => internal}

define transit(X,Y) =
enter(X|Y) and exit(X|Y)
```
**Example: A DC network with Propane**

```plaintext
define Destination =
{GP1 => end(A)
 GP2 => end(B)
 LP1 => end(E)
 LP2 => end(F)
 true => exit(Peer1 >> Peer2)}

define Locality =
{LP1 | LP2 => internal}

define transit(X,Y) =
 enter(X|Y) and exit(X|Y)

define NoTransit =
{true => !transit(Peer1,Peer2)}
```

---

**Diagram:**

![Diagram of a DC network with Propane prefixes and two peers: Peer1 and Peer2. The network includes global prefixes (GP) and local prefixes (LP). The prefixes are connected with arrows indicating the direction of transit.]
**Example: A DC network with Propane**

```plaintext
define Destination =
{GP1  =>  end(A)
 GP2  =>  end(B)
 LP1  =>  end(E)
 LP2  =>  end(F)
 true  =>  exit(Peer1 >> Peer2)}

define Locality =
{LP1 | LP2 => internal}

define transit(X,Y) =
 enter(X|Y) and exit(X|Y)

define NoTransit =
{true => !transit(Peer1,Peer2)}

define Main =
 Destination & Locality &
 NoTransit & agg(GP, in -> out)
```

Diagram:
- Global Prefixes: GP1, GP2
- Local Prefixes: LP1, LP2
- Peers: Peer1, Peer2
- Nodes: X, Y, A, B, C, D, E, F, G, H

Example graph showing connectivity between global and local prefixes.
Compilation

Propane ➔ Compiler ➔ Topology ➔ BGP Configs
Compilation

Constraints on Policy

Topology

State Machines

Propane

Compiler

BGP Configs
Compilation

Propane → Compiler → Product Graph → Topology

Jointly analyze with topology

BGP Configs
Compilation: A simple Example

end(Y) & (path(A, C, D) >> any)
Compilation: A simple Example

\[ \text{end}(Y) \& (\text{path}(A,C,D) \gg \text{any}) \]

Convert to Regex

\[ XACDY \gg (\Sigma^*) Y \]
Reversed Automata from Policies

Policy:

1. XACDY
2. (Σ*)Y

More preferred paths
Less preferred paths
Reversed Automata from Policies

Policy:

1. XACDYO
2. (Σ*)Y
Reversed Automata from Policies

Reversed automata tracks BGP message flow

Policy:

1. XACDY
2. \((\Sigma^*)Y\)
Constructing the Product Graph (PG)
Constructing the Product Graph (PG)

Topology

Location

Automata

States

(X, 5, I)

0 \rightarrow Y \rightarrow 1 \rightarrow D \rightarrow 2 \rightarrow C \rightarrow 3 \rightarrow A \rightarrow 4 \rightarrow X \rightarrow 5

0 \rightarrow Y \rightarrow \Sigma \rightarrow 0 \rightarrow 1
Constructing the Product Graph (PG)

Topology
Location
Automata
States

\((X, 5, 1)\)

\{1, 2\}

Path preferences
Constructing the Product Graph (PG)
Constructing the Product Graph (PG)
Constructing the Product Graph (PG)

Graph capturing all possible policy-compliant paths through the topology
Constructing the Product Graph (PG)
Constructing the Product Graph (PG)
Constructing the Product Graph (PG)

Accept: Y D C A X

Preferences

\{1, 2\}

\{2\}
Constructing the Product Graph (PG)
Compilation to BGP:

Idea 1: Restrict advertisements to edges
- Encode state in a BGP community tag
- Incoming edges — import filters
- Outgoing edges — export filters

Let BGP find some allowed path dynamically
Compilation to BGP:

D allows import matching regex(Y)
Compilation to BGP:

D exports to C with tag (2,1)
Compilation to BGP:

C allows import from D with tag (2,1)
Compilation to BGP:

C exports to A, B, D, E with tag (3, 1)

C exports to A, B, D, E with tag (3, 1)
Compilation to BGP:

Idea 2: Find preferences

- Direct BGP towards best path
- Under all combinations of failures

Let BGP find the best path dynamically
Compilation to BGP:

Router C
match peer = D ...
match peer = E ...

{2} {1, 2}
Compilation to BGP:

Router C

match peer = D ...
local-pref ← ???

match peer = E ...
local-pref ← ???

{2} {1, 2}
Compilation to BGP:

Efficient algorithm to assign preferences that forces BGP to find the best paths for all possible failures

See the paper for details!
Compilation to BGP:

Implementation: Local preference

Less preferred import

More preferred import

C, -, 1

A, -, 1

B, -, 1

E, -, 1

D, 2, 1

Y, 1, 1

start

{2} {1, 2}
Compilation to BGP:

Implementation: MED/Prepending

Less preferred import

More preferred import
end(Y) & (path(A, C, D) >> any)
Implementation

- Written in 7000 lines of F#
- Generates Quagga configurations
- A number of other analyses & features
Benchmarks

- Configurations from a large cloud provider
- Policy described in English documents
- Datacenter and Backbone policies
Policy Size

Without prefix/peer definitions

- Datacenter policy: 31 lines of Propane
- Backbone policy: 43 lines of Propane

Conventional BGP configurations are 1000s of lines
Compilation Time

- Compile for each prefix *equivalence class*
- Compile for each equivalence class in *parallel*
- 8 core, 3.6 GHz Intel Xeon processor

**Data center (< 9 min)**

**Backbone (< 3 min)**
Configuration Size

Optimizations

- Avoid using community tags when unambiguous
- Reuse community values across peers
- Merge import/export behaviors across peers

Results

- Optimizations yield $50-100\times$ decrease in config size
- Configurations $\sim 1000-10000$ lines per router
Propane: Summary

High-level language

- **Centralized** network programmability
- Constraints specify preferred paths and backups in case of failure
- Uniform abstractions for **Inter**- and **Intra**-domain routing
- Core policy in 30-50 lines of Propane vs. 1000s

Compiler

- **Distributed** implementation via BGP
- Static analysis guarantees policy compliance for all failures
- **Scales** to reasonably sized network topologies