PC Rev 1

Fibbing, coding, vectoring
April 21, 2016
Consider this simple network
(implemented with Cisco routers)
An IGP control-plane computes shortest paths on a shared weighted topology.
IGP shortest paths are translated into forwarding paths on the data-plane.

In the control-plane, the shortest path is from A to X. In the data-plane, the traffic flows from D2 to X and then to D1.
In Fibbing, operators can ask the controller to modify forwarding paths
The Fibbing controller injects information on \textit{fake nodes and links} to the IGP control-plane.
Informations are flooded to all IGP routers in the network

requirement (C,A,B,X,D2)

node V1, link (V1,C), map (V1,C) to (C,A)
Fibbing messages *augment* the topology seen by all IGP routers.
Augmented topologies translate into new control-plane paths

requirement \((C, A, B, X, D2)\)

node \(V_1\), link \((V_1, C)\), map \((V_1, C)\) to \((C, A)\)
Augmented topologies translate into new *data-plane* paths

requirement
(C,A,B,X,D2)

node V1, link (V1,C), map (V1,C) to (C,A)
Fibbing can program
arbitrary per-destination paths

Theorem  Any set of forwarding DAGs can be enforced by Fibbing
Fibbing can program arbitrary per-destination paths

Theorem

Any set of forwarding DAGs can be enforced by Fibbing

paths to the same destination do not create loops
By achieving full per-destination control, Fibbing is highly flexible.

Theorem: Any set of forwarding DAGs can be enforced by Fibbing.

- fine-grained traffic steering (middleboxing)
- per-destination load balancing (traffic engineering)
- backup paths provisioning (failure recovery)
Central Control over Distributed Routing

fibbing.net

1 Manageability
2 Flexibility
3 Scalability
4 Robustness
We implemented a Fibbing controller
We also propose algorithms to compute augmented topologies of limited size.
For our Fibbing controller, we propose algorithms to be run in sequence.
Consider the following example, with a drastic forwarding path change.

original shortest-path
“down and to the right”

desired shortest-path
“up and to the right”
Simple adds one fake node for every router that has to change next-hop.
Merger iteratively merges fake nodes
(starting from Simple’s output)
Merger iteratively merges fake nodes
(starting from Simple’s output)
This way, Merger programs multiple next-hop changes with a single fake node.
This way, Merger programs multiple next-hop changes with a single fake node

Previous SDN solutions (e.g., RCP) cannot do the same
Simple and Merger achieve different trade-offs in terms of time and optimization efficiency

We ran experiments on Rocketfuel topologies, with at least 25% of nodes changing next-hops

- Simple runs in milliseconds
  Merger takes 0.1 seconds

- Merger reduces fake nodes by up to 50%
  and up to 90% with cross-destination optimization
We implemented the machinery to listen to OSPF and augment the topology.
Experiments on real routers show that Fibbing has very limited impact on routers

<table>
<thead>
<tr>
<th># fake nodes</th>
<th>router memory (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 000</td>
<td>0.7</td>
</tr>
<tr>
<td>5 000</td>
<td>6.8</td>
</tr>
<tr>
<td>10 000</td>
<td>14.5</td>
</tr>
<tr>
<td>50 000</td>
<td>76.0</td>
</tr>
<tr>
<td>100 000</td>
<td>153</td>
</tr>
</tbody>
</table>

DRAM is cheap

>> # real routers

>> # real routers
Experiments on real routers show that Fibbing has very limited impact on routers.

<table>
<thead>
<tr>
<th># fake nodes</th>
<th>router memory (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>0.7</td>
</tr>
<tr>
<td>5000</td>
<td>6.8</td>
</tr>
<tr>
<td>10000</td>
<td>14.5</td>
</tr>
<tr>
<td>50000</td>
<td>76.0</td>
</tr>
<tr>
<td>100000</td>
<td>153</td>
</tr>
</tbody>
</table>

DRAM is cheap

CPU utilization always under 4%
Experiments on real routers show that Fibbing does not impact IGP convergence.

Upon link failure, we registered *no difference* in the (sub-second) IGP convergence with

- no fake nodes
- up to 100,000 fake nodes and destinations
Experiments on real routers show that Fibbing achieves fast forwarding changes.

<table>
<thead>
<tr>
<th># fake nodes</th>
<th>installation time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 000</td>
<td>0.9</td>
</tr>
<tr>
<td>5 000</td>
<td>4.5</td>
</tr>
<tr>
<td>10 000</td>
<td>8.9</td>
</tr>
<tr>
<td>50 000</td>
<td>44.7</td>
</tr>
</tbody>
</table>
| 100 000      | 89.50                       | 894.50 μs/entry
Network Coding – Background


Allowing routers to mix the bits in forwarding messages can increase network throughput (Achieves multicast capacity).
Chronology of Research

• Li et al. – Showed that linear codes are sufficient to achieve maximum capacity bounds (2003)
• Koetter and Medard – Polynomial time algorithms for encoding and decoding (2003)
• Ho et al. – Extended previous results to a randomized setting (2003)
• Studies on wireless network coding began in 2003 as well! (Shows that it was a high interest research area)
• More work on wireless network coding with multicast models (2004)
• Lun et al. – Problem of minimizing communication cost in wireless networks can be formulated linearly (2005) – Used multicast model as well!
  So all the previous work was theoretical and assumes multicast traffic.
• Authors introduced the idea of opportunistic coding for wireless environments in 2005
  Why is it different?
  They address the common case of unicast traffic, bursty flows and other practical issues.
Current Paper

• Explores the utility of network coding in improving the throughput of wireless networks.
• Authors extend the theory of their opportunistic coding architecture (COPE) by application in a practical scenario.
• Presents the first system architecture for wireless network coding.
• Implements the design, creating the first deployment of network coding in a wireless network.
• Studies the performance of COPE.
COPE

• What does being opportunistic mean?
  Each node relies on local information to detect and exploit coding opportunities when they arise, so as to maximize throughput.

• COPE inserts an *opportunistic* coding shim between the IP and MAC layers.

• Enables forwarding of multiple packets in a single transmission.

• Based on the fact that intelligently mixing packets increases network throughput.
Design Principles:

– COPE embraces the broadcast nature of the wireless channel.
– COPE employs network coding.
Inside COPE

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native Packet</td>
<td>A non-encoded packet</td>
</tr>
<tr>
<td>Encoded or XOR-ed Packet</td>
<td>A packet that is the XOR of multiple native packets</td>
</tr>
<tr>
<td>Nexthops of an Encoded Packet</td>
<td>The set of nexthops for the native packets XOR-ed to generate the encoded packet</td>
</tr>
<tr>
<td>Packet Id</td>
<td>A 32-bit hash of the packet’s IP source address and IP sequence number</td>
</tr>
<tr>
<td>Output Queue</td>
<td>A FIFO queue at each node, where it keeps the packets it needs to forward</td>
</tr>
<tr>
<td>Packet Pool</td>
<td>A buffer where a node stores all packets heard in the past $T$ seconds</td>
</tr>
<tr>
<td>Coding Gain</td>
<td>The ratio of the number of transmissions required by the current non-coding approach, to the number of transmissions used by COPE to deliver the same set of packets.</td>
</tr>
<tr>
<td>Coding+MAC Gain</td>
<td>The expected throughput gain with COPE when an 802.11 MAC is used, and all nodes are backlogged.</td>
</tr>
</tbody>
</table>

COPE incorporates three main techniques:

– Opportunistic Listening
– Opportunistic Coding
– Learning Neighbor State
Opportunistic Listening

• Nodes are equipped with omni-directional antennae
• COPE sets the nodes to a promiscuous mode.
• The nodes store the overheard packets for a limited period T (0.5 s)
• Each node also broadcasts reception reports to tell its neighbors which packets it has stored.
Opportunistic Coding

Rule:
“*A node should aim to maximize the number of native packets delivered in a single transmission, while ensuring that each intended next-hop has enough information to decode its native packet.*”

(a) B can code packets it wants to send

<table>
<thead>
<tr>
<th>Coding Option</th>
<th>Is it good?</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1 + P2</td>
<td>Bad Coding (C can decode but A can’t)</td>
</tr>
<tr>
<td>P1 + P3</td>
<td>Better Coding (Both A and C can decode)</td>
</tr>
<tr>
<td>P1 + P3 + P4</td>
<td>Best Coding (Nodes A, C, and D can decode)</td>
</tr>
</tbody>
</table>

(b) Next hops of packets in B’s queue

(c) Possible coding options
Issues:

– Unneeded data should not be forwarded to areas where there is no interested receiver, wasting capacity.
– The coding algorithm should ensure that all next-hops of an encoded packet can decode their corresponding native packets.

Rule: To transmit $n$ packets $p_1 \ldots p_n$ to $n$ next-hops $r_1 \ldots r_n$, a node can XOR the $n$ packets together only if each next-hop $r_i$ has all $n - 1$ packets $p_j$ for $j \neq i$. 
Learning Neighbor State

- A node cannot solely rely on reception reports, and may need to guess whether a neighbor has a particular packet.
- To guess intelligently, we can leverage routing computations. The ETX metric computes the delivery probability between nodes and assigns each link a weight of $1/(\text{delivery\_probability})$.
- In the absence of deterministic information, COPE estimates the probability that a particular neighbor has a packet, as the delivery probability of the link between the packet’s previous hop and the neighbor.

```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
Understanding COPE’s Gains

Coding Gain
- Defined as the ratio of no. of transmissions required without COPE to the no. of transmissions used by COPE to deliver the same set of packets.
- By definition, this number is greater than 1. (4/3 for Alice-Bob Example)
- Theorem: In the absence of opportunistic listening, COPE’s maximum coding gain is 2, and it is achievable.

Coding Gain achievable = \[ \frac{2N}{N+1} \]

This value tends to 2 as...
In the presence of opportunistic listening

**Achievable Coding Gain**

- "X" topology
  - 2 flows intersecting at \( n_2 \).
  - **Achievable Coding Gain**
    - \( = 1.33 \)

- Cross topology
  - 4 flows intersecting at \( n_2 \).
  - **Achievable Coding Gain**
    - \( = 1.6 \)
Understanding COPE’s Gains

Coding + MAC Gain

– It was observed that throughput improvement using COPE greatly exceeded the coding gain.
– Since it tries to be fair, the MAC layer divides the bandwidth equally between contending nodes.
– COPE allows the bottleneck nodes to XOR pairs of packets and drain them quicker, increasing the throughput of the network.
– For topologies with a single bottleneck, the Coding + MAC Gain is the ratio of the bottleneck’s draining rate with COPE to its draining rate without COPE.
• **Theorem**: In the absence of opportunistic listening, COPE’s maximum Coding + MAC gain is 2, and it is achievable.

Node can XOR at most 2 packets together, and the bottleneck can drain at almost twice as fast, bounding the Coding + MAC Gain at 2.

• **Theorem**: In the presence of opportunistic listening, COPE’s maximum Coding + MAC gain is unbounded.

For N edge nodes, the bottleneck node XORs N packets together, and the queue drains N times faster.

Wheel topology; many flows intersecting at the center node.
• Theoretical gains:

<table>
<thead>
<tr>
<th>Topology</th>
<th>Coding Gain</th>
<th>Coding+MAC Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice-and-Bob</td>
<td>1.33</td>
<td>2</td>
</tr>
<tr>
<td>“X”</td>
<td>1.33</td>
<td>2</td>
</tr>
<tr>
<td>Cross</td>
<td>1.6</td>
<td>4</td>
</tr>
<tr>
<td>Infinite Chain</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Infinite Wheel</td>
<td>2</td>
<td>$\infty$</td>
</tr>
</tbody>
</table>

• Important to note that:
  – The gains in practice tend to be lower due to non-availability of coding opportunities, packet header overheads, medium losses, etc.,
  – But COPE does increase actual information rate of the medium far above the bit rate.
Can’t acknowledge a packet until you can decode. Usually, decoding requires a number of packets. Code / acknowledge over small blocks to avoid delay, manage complexity.
## Compare to ARQ

**Context:** Reliable communication over a (wireless) network of packet erasure channels

<table>
<thead>
<tr>
<th>ARQ</th>
<th>Network Coding</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Retransmit lost packets</td>
<td>• Transmit linear combinations of packets</td>
</tr>
<tr>
<td>• Low delay, queue size</td>
<td>• Achieves min-cut multicast capacity</td>
</tr>
<tr>
<td>• Streaming, not blocks</td>
<td>• Extends to broadcast links</td>
</tr>
<tr>
<td>• Not efficient on broadcast links</td>
<td>• Congestion control requires feedback</td>
</tr>
<tr>
<td>• Link-by-link ARQ does not achieve network multicast capacity.</td>
<td>• Decoding delay: block-based</td>
</tr>
</tbody>
</table>
Goals

• Devise a system that behaves as close to TCP as possible, while masking non-congestion wireless losses from congestion control where possible.
• Stream-based, not block-based.
• Low delay.
• Focus on wireless setting.
  – Where network coding can offer biggest benefits.
  – Not necessarily a universal solution.
Main Idea : Coding ACKs

• What does it mean to “see” a packet?
  • Standard notion: we have a copy of the packet.
    – Doesn’t work well in coding setting.
    – Implies must decode to see a packet.

• New definition: we have a packet that will allow us to decode *once enough* useful packets arrive.
  – Packet is useful if linearly independent.
  – When enough useful packets arrive can decode.
Coding ACKs

- For a message of size $n$, need $n$ useful packets.
- Each coded packet corresponds to a degree of freedom.
- Instead of acknowledging individual packets, acknowledge newly arrived degrees of freedom.
Coding ACKs

Original message: \( p_1, p_2, p_3 \ldots \)

Coded Packets

| \( c_1 \) | \( 4 \ 2 \ 5 \ 0 \ 0 \ 0 \ 0 \) |
| \( c_2 \) | \( 3 \ 1 \ 2 \ 5 \ 0 \ 0 \ 0 \) |
| \( c_3 \) | \( 1 \ 2 \ 3 \ 4 \ 1 \ 0 \ 0 \) |
| \( c_4 \) | \( 3 \ 3 \ 1 \ 2 \ 1 \ 0 \ 0 \) |
| \( c_5 \) | \( 1 \ 2 \ 5 \ 4 \ 5 \ 0 \ 0 \) |

4\( p_1 \) + 2\( p_2 \) + 5\( p_3 \)
Coding ACKs

Original message: \( p_1, p_2, p_3 \ldots \)

Coded Packets

\[
\begin{array}{c|ccccccc}
\text{c}_1 & 4 & 2 & 5 & 0 & 0 & 0 & 0 \\
\text{c}_2 & 3 & 1 & 2 & 5 & 0 & 0 & 0 \\
\text{c}_3 & 1 & 2 & 3 & 4 & 1 & 0 & 0 \\
\text{c}_4 & 3 & 3 & 1 & 2 & 1 & 0 & 0 \\
\text{c}_5 & 1 & 2 & 5 & 4 & 5 & 0 & 0 \\
\end{array}
\]

When \( c_1 \) comes in, you’ve “seen” packet 1; eventually you’ll be able to decode it. And so on…
Coding ACKs

Original message: $p_1, p_2, p_3...$

Coded Packets

<table>
<thead>
<tr>
<th>$c_1$</th>
<th>42500000</th>
<th>14530000</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_2$</td>
<td>31250000</td>
<td>01326000</td>
</tr>
<tr>
<td>$c_3$</td>
<td>12341000</td>
<td>00162000</td>
</tr>
<tr>
<td>$c_4$</td>
<td>33121000</td>
<td>00015000</td>
</tr>
<tr>
<td>$c_5$</td>
<td>12545000</td>
<td>00001000</td>
</tr>
</tbody>
</table>

Use Gaussian elimination as packets arrive to check for a new seen packet.
Formal Definition

• A node has *seen* a packet $p_k$ if it can compute a linear combination $p_k + q$ where $q$ is a linear combination of packets with index larger than $k$.

• When all packets have been seen, decoding is possible.
Layered Architecture

SOURCE SIDE

Application

TCP

IP

MAC / PHY

RECEIVER SIDE

Application

TCP

IP

MAC / PHY

Physical medium

Data

ACK

Eg. HTTP, FTP

Transport layer: Reliability, flow and congestion control

Network layer (Routing)

Medium access, channel coding
TCP using Network Coding
The Sender Module

• Buffers packets in the current window from the TCP source, sends linear combinations.

• Need for redundancy factor $R$.
  – Sending rate should account for loss rate.
  – Send a constant factor more packets.
  – Open issue: determine $R$ dynamically?
Redundancy

• Too low $R$
  – TCP times out and backs off drastically.

• Too high $R$
  – Losses recovered – TCP window advances smoothly.
  – Throughput reduced due to low code rate.
  – Congestion increases.

• Right $R$ is $1/(1-p)$, where $p$ is the loss rate.
BGP-4

- **BGP** = **Border Gateway Protocol**
- Is a **Policy-Based** routing protocol
- Is the **de facto EGP** of today’s global Internet
- Relatively simple protocol, but configuration is complex and the entire world can see, and be impacted by, your mistakes.

**1989 : BGP-1 [RFC 1105]**
- Replacement for EGP (1984, RFC 904)

**1990 : BGP-2 [RFC 1163]**

**1991 : BGP-3 [RFC 1267]**

**1995 : BGP-4 [RFC 1771]**
- Support for Classless Interdomain Routing (CIDR)
BGP Operations (Simplified)

Establish session on TCP port 179

Exchange all active routes

Exchange incremental updates

While connection is ALIVE exchange route UPDATE messages
Two Types of BGP Neighbor Relationships

- External Neighbor (eBGP) in a different Autonomous Systems
- Internal Neighbor (iBGP) in the same Autonomous System

iBGP is routed (using IGP!)
iBGP Mesh Does Not Scale

- N border routers means \( N(N-1)/2 \) peering sessions
- Each router must have \( N-1 \) iBGP sessions configured
- The addition a single iBGP speaker requires configuration changes to all other iBGP speakers
- Size of iBGP routing table can be order \( N \) larger than number of best routes (remember alternate routes!)
- Each router has to listen to update noise from each neighbor

Currently four solutions:
(0) Buy bigger routers!
(1) Break AS into smaller ASes
(2) BGP Route reflectors
(3) BGP confederations
Route Reflector

- Route reflectors can pass on iBGP updates to clients
- Each RR passes along ONLY best routes
  - ORIGINATOR_ID and CLUSTER_LIST attributes are needed to avoid loops
BGP Confederations

From the outside, this looks like AS 1

Confederation eBGP (between member ASes) preserves LOCAL_PREF, MED, and BGP NEXTHOP.
Four Types of BGP Messages

- **Open**: Establish a peering session.
- **Keep Alive**: Handshake at regular intervals.
- **Notification**: Shuts down a peering session.
- **Update**: Announcing new routes or withdrawing previously announced routes.

announcement = prefix + attributes values
# BGP Attributes

<table>
<thead>
<tr>
<th>Value</th>
<th>Code</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ORIGIN</td>
<td>[RFC1771]</td>
</tr>
<tr>
<td>2</td>
<td>AS_PATH</td>
<td>[RFC1771]</td>
</tr>
<tr>
<td>3</td>
<td>NEXT_HOP</td>
<td>[RFC1771]</td>
</tr>
<tr>
<td>4</td>
<td>MULTI_EXIT_DISC</td>
<td>[RFC1771]</td>
</tr>
<tr>
<td>5</td>
<td>LOCAL_PREF</td>
<td>[RFC1771]</td>
</tr>
<tr>
<td>6</td>
<td>ATOMIC_AGGREGATE</td>
<td>[RFC1771]</td>
</tr>
<tr>
<td>7</td>
<td>AGGREGATOR</td>
<td>[RFC1771]</td>
</tr>
<tr>
<td>8</td>
<td>COMMUNITY</td>
<td>[RFC1997]</td>
</tr>
<tr>
<td>9</td>
<td>ORIGINATOR_ID</td>
<td>[RFC2796]</td>
</tr>
<tr>
<td>10</td>
<td>CLUSTER_LIST</td>
<td>[RFC2796]</td>
</tr>
<tr>
<td>11</td>
<td>DPA</td>
<td>[Chen]</td>
</tr>
<tr>
<td>12</td>
<td>ADVERTISER</td>
<td>[RFC1863]</td>
</tr>
<tr>
<td>13</td>
<td>RCID_PATH / CLUSTER_ID</td>
<td>[RFC1863]</td>
</tr>
<tr>
<td>14</td>
<td>MP_REACH_NLRI</td>
<td>[RFC2283]</td>
</tr>
<tr>
<td>15</td>
<td>MP_UNREACH_NLRI</td>
<td>[RFC2283]</td>
</tr>
<tr>
<td>16</td>
<td>EXTENDED COMMUNITIES</td>
<td>[Rosen]</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>255</td>
<td>reserved for development</td>
<td></td>
</tr>
</tbody>
</table>

From IANA: [http://www.iana.org/assignments/bgp-parameters](http://www.iana.org/assignments/bgp-parameters)

- **Most important attributes**
- Not all attributes need to be present in every announcement
Attributes are Used to Select Best Routes

Given multiple routes to the same prefix, a BGP speaker must pick at most one best route.

(Note: it could reject them all!)
Route Selection Summary

- Highest Local Preference
- Shortest ASPATH
- Lowest MED
- i-BGP < e-BGP
- Lowest IGP cost to BGP egress
- Lowest router ID

Enforce relationships

traffic engineering

Throw up hands and break ties
BGP Route Processing

Open ended programming. Constrained only by vendor configuration language

Receive BGP Updates
- Apply Policy = filter routes & tweak attributes
  - Based on Attribute Values
  - Best Routes
    - Apply Policy = filter routes & tweak attributes
      - Transmit BGP Updates

Apply Import Policies
Best Route Selection
Best Route Table
Apply Export Policies

Install forwarding Entries for best Routes.

IP Forwarding Table
BGP Next Hop Attribute

Every time a route announcement crosses an AS boundary, the Next Hop attribute is changed to the IP address of the border router that announced the route.
Join EGP with IGP For Connectivity

Forwarding Table

<table>
<thead>
<tr>
<th>destination</th>
<th>next hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.0.2.0/30</td>
<td>10.10.10.10</td>
</tr>
</tbody>
</table>

EGP

<table>
<thead>
<tr>
<th>destination</th>
<th>next hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>135.207.0.0/1</td>
<td>192.0.2.1</td>
</tr>
<tr>
<td>192.0.2.0/30</td>
<td>10.10.10.10</td>
</tr>
</tbody>
</table>

Next Hop = 192.0.2.1
Implementing Customer/Provider and Peer/Peer relationships

Two parts:

- Enforce transit relationships
  - Outbound route filtering
- Enforce order of route preference
  - provider < peer < customer
Import Routes

- provider route
- peer route
- customer route
- ISP route
Export Routes

- Provider route
- Peer route
- Customer route
- ISP route

Filters block
How Can Routes be Colored?

BGP Communities!

A community value is 32 bits

By convention, first 16 bits is ASN indicating who is giving it an interpretation

Community Attribute = a list of community values. (So one route can belong to multiple communities)

Two reserved communities
no_export = 0xFFFFFFFF01: don’t export out of AS
no_advertise 0xFFFFFFFF02: don’t pass to BGP neighbors

RFC 1997 (August 1996)

Very powerful BECAUSE it has no (predefined) meaning

Used for signally within and between ASes
Communities Example

- **1:100**
  - Customer routes

- **1:200**
  - Peer routes

- **1:300**
  - Provider Routes

- **Import**
  - To Customers
    - 1:100, 1:200, 1:300
  - To Peers
    - 1:100
  - To Providers
    - 1:100

- **Export**
So Many Choices

Which route should Frank pick to 13.13.0.0/16?

Frank's Internet Barn

AS 1

AS 2

AS 3

AS 4

peer
peer
provider
customer

13.13.0.0/16
LOCAL PREFERENCE

Higher Local preference values are more preferred

Local preference used ONLY in iBGP

AS 1

AS 2

AS 3

AS 4

13.13.0.0/16

local pref = 80

local pref = 90

local pref = 100

local pref = 80

local pref = 90

local pref = 100