Internet Exchanges and Peering

Scribe Notes

1 Introduction

1.1 Shortcomings of the Border Gateway Protocol (BGP)

In the last lecture we discussed the details of the Border Gateway Protocol (BGP), the Internet's current interdomain routing protocol. BGP lacks sufficient flexibility for performing various traffic engineering and traffic management operations. It's generally only possible to:

- Route on destination IP addresses.
- Influence immediately neighboring ASes.
- Indirectly control how switches/routers forward traffic (through mechanisms like AS path prepending, local preference, etc.).
- Use basic packet forwarding techniques (difficult to introduce new in-network services like arbitrarily routing flows through "middleboxes").

In this lecture we discuss two papers that aim to achieve more flexible forms of interdomain routing. There are several wide-area use-cases that would be extremely valuable to network operators if only they had better ways to implement them. For example:

- Application-specific peering: providing the ability for ASes to exchange traffic only for specific applications, such as video streaming.
- Fine-grained approaches to routing: for example, redirection through middleboxes.
- Server load balancing: directing client requests to different data centers depending on where they originate from.
- Inbound traffic engineering: splitting incoming traffic over multiple incoming peering links.

It's not that the services above are impossible with BGP, but they are extremely clunky, inconvenient, and error-prone (analogous to writing assembly code for developing a computer vision app).

1.2 Software-Defined Networking (SDN)

1.2.1 Overview

Routers have two time scales: the control plane and data plane. Generally, the control plane finds routes between two points, while the data plane enforces these routes by forwarding packets to the next hop. Historically, router boxes were designed with a CPU for the control plane, and a custom ASIC to handle fast-forwarding in the data plane. In a network with many of these routers, the routing algorithm (ex. BGP) would be run on the CPUs within each of these routers in a distributed fashion (in the last lecture we

discussed establishing a BGP session over TCP between border routers – this BGP session is really between the CPUs of the routers). The problem with this approach is that distributed algorithms are difficult to reason about (determining convergence, optimizations, etc.).

Software-Defined Networking (SDN) offers a solution. The main idea behind SDN is separating the control plane from the router box by moving the computing power required for route computation to a decoupled fleet of servers (note that a CPU would still be present in the routers, but only for basic bookkeeping tasks). The benefit of this approach centers on the servers having a universal view of the network rather than each router only having partial information. This global view allows for the development of more sophisticated algorithms for more optimal routing.

The SDN approach works as long as communication between the control and data plane is relatively infrequent. The cost of shuttling data between the control and data planes (occurring on link failures or aggressive traffic engineering, i.e. re-optimizing every microsecond) would outweigh the benefits of this design. Furthermore, for SDN to scale, the control plane must have a standardized interface for configuring router ASICs across multiple vendors. This was only accomplished some years after SDN was initially proposed through the development of OpenFlow.

1.2.2 Applications for Interdomain Routing

Broadly speaking, SDN addresses the shortcomings of BGP; not by proposing new features, but by offering much greater ease of use. SDN offers a convenient way for network operators to handle the use cases listed in Section 1.1. With SDN, switches/routers can have more direct control over packet handling and can match packets based on multiple header fields (as opposed to only the destination IP address), allowing for many different actions to be performed on packets beyond simply forwarding them. Furthermore, entire networks can be controlled (as opposed to just an AS's immediate neighbors) with just a single program.

2 SDX Paper

This is an academic paper providing an overview of concepts and a proposed design for a Software-Defined Internet Exchange (SDX). The researchers use a basic deployment (university researchers typically do not have their own AS to experiment with), consisting of a virtual AS and simple emulation-based experiments.

2.1 Internet Exchange Point (IXP)

An IXP is the digital analogue of the old telephone exchange points, where several peer ASes physically connect with each other to exchange routing information and data (note that just because two ASes connect to the same IXP doesn't mean they have a peering or transit

agreement in place). IXPs are convenient alternatives to the otherwise numerous and isolated, pair-wise physical connections that would have to be made between peering ASes. ASes must still negotiate peering agreements, but the mechanical connection is accomplished through IXPs.

Within IXPs, all ASes border routers are connected to each other over a layer-two network; providing the abstraction of a LAN. The border routers are also connected to a BGP Route Server which compiles the path advertisements from each router and broadcasts the combined information to all ASes.

2.2 Software Defined Internet Exchange (SDX) Design

The paper provides the abstraction of an SDX through the use of a virtual SDN switch which connects each AS to its peer ASes. Each AS can write forwarding policies as if it is the only participant at the SDX, yet each AS cannot influence how the other ASes forward packets on their own virtual switches.

Forwarding policies on the virtual switches are written using the Pyretic language, providing a convenient way for ASes to write policies for a variety of use cases (defaulting to BGP in the absence of a policy):

- Application-specific peering: forwarding packets to a peer AS based on destination ports.
- Inbound traffic engineering: forwarding packets to an AS's own ingresses based on source IP addresses.
- Wide-area server load balancing: forwarding packets to various destination IPs based on client IP prefixes.

The paper also converts the central BGP Route Server to an SDX-enabled Route Server, which ultimately combines policies across ASes (into single match-action rules), ensures policies are consistent with agreements/BGP route advertisements, compiles the code and distributes to all AS border routers.

In conclusion, the added flexibility and ease of use proposed by the SDX paper largely comes from a high-level language to express policies (do not have to deal with low-level forwarding rules) and a compiler that automates this transformation.

3 Espresso Paper

This is an industry paper detailing a system built over a few years, and discusses the problems and advances faced along the way. The researchers' deployment is considerably more sophisticated than that of the SDX paper (thanks to the Google backing), but the paper ends up being low on insight and conceptual ideas are difficult to extract.

3.1 Peering Edge

Peering agreements between internet companies and ISPs are a relatively recent development. Historically there had been a separation between content providers and ISPs. As internet companies, like Google, grew in size, they replicated servers across multiple geographical locations to be closer to clients (minimizing object fetch times over the network and therefore total webpage latency). These local servers are called 'metros' which peer with nearby ISPs – and this border between metros and ISPs is called the 'peering edge'.

Google was able to build out this peering edge by repurposing available content caching servers to run an entire BGP stack, using it for advertising routes and peering in an application-specific manner with the ISPs. Espresso's main design goal is inbound traffic engineering: regulating the amount of traffic coming into each of Google's metros.

3.2 Espresso Design

The authors' main design requirements were:

- High feature velocity: must be efficient for network operators to adopt/implement changes and features.
- Incremental deployment: given Google's scale, must support a gradual rollout.
- Reliability: 99.999% global availability
- Interoperability: ability to work with existing routers, supporting all standard Internet protocols (important for interdomain routing).

To meet the requirements, the authors employed the following design principles:

- Hierarchical control plane: there is a local controller which manages the local metro (handling faults quickly, improving reliability, etc), while a global controller computes the universal-view route computation using a greedy algorithm. Note that the control plane does not have to be a single server – it can even be a supercomputing cluster. SDN's main idea is just that this control plane topology be decoupled from the data plane.
- Fail static: the data plane maintains the last known good state so that the control plane may be unavailable for short periods of time (ex. due to upgrades, bug fixes) without impacting packet forwarding. This principle ensures high availability at scale.
- Software programmability: simple hardware primitives are used so that new features can be introduced without waiting for vendors to release updates, allowing networks to evolve with changing application requirements and high feature velocity.
- Testability and manageability

In conclusion, the convenience resulting from Espresso's design is largely based on the idea that the authors built their own stack and developed everything in-house – resulting in higher feature velocity (few weeks instead of few years) and better route convergence.