BST—BGP Scalable Transport

Kedar Poduri
Cengiz Alaettinoglu
Van Jacobson

NANOG 27
Phoenix, AZ
February 9, 2003

BGP gets the job done but has some problems

- Convergence is slow (minutes to tens of minutes)
- It's configuration intensive and configuration errors are often disastrous
- Scaling is problematic: Overhead grows as the square of the core size while peering capacity grows only linearly.
- Reliability is poor: protocol depends on a large number of point-to-point connections (n^2 for a core of size n) and failure of any of them causes large-scale ripple effects on the entire core.

There's talk of replacing BGP but it took fifteen years to get this one working and no one wants to go through that pain again.

There may be a painless way to fix many of the problems

"BGP" is built from two separable pieces:

1. the BGP protocol and

2. the TCP transport used to carry protocol messages between peers.

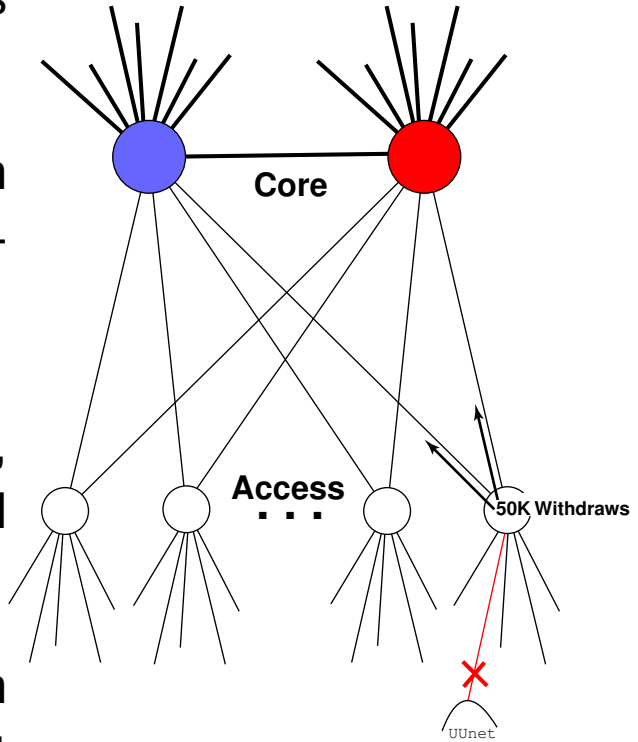
It's relatively easy to add additional transport(s) to BGP.

- To quantify: adding BST to GateD required changing ~200 lines of GateD code.

If designed carefully, the new transport will have *no* effect on the protocol or its behavior — the same bits get delivered to the same peers in the same order, just a little faster and more reliably (think of mailing a contract via FedEx vs USPS).

Where can transport help?

Take a typical PoP with core routers in a full IBGP mesh and acting as router reflectors for the PoP's access routers.



- Say one of the access routers loses a peering with UUnet. That router sends withdraws for ~50K prefixes to its local core routers (~500KB of data).

- Since other UUnet peerings are in different pops, the core routers have to send those withdraws to all the other core routers (100–200 of them).

So each core router in the PoP sends 500KB on each of 200 different TCP connections – 100MB total or at least a minute on a 100Mb/s ethernet.

- Transport can't fix the 500KB of withdraws but a better transport can prevent inflating that 500KB into 100MB.

“Better” transport?

Reading RFC1771 (BGP4) or looking at the bits on the wire will tell you that the *same* messages are sent to all 200 peers.

> Rather than sending 200 copies of the same thing, a *multipoint transport* could take care of delivering one copy to all interested parties.

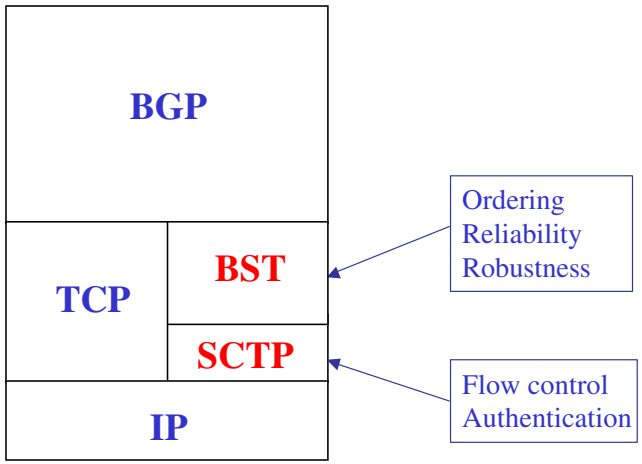
Multipoint transport can be done two different ways:

- use multicast (e.g., PGM for IBGP I-D from Dino Farinacci & Tony Speakman, 1999)

- use application level replication & flooding like IS-IS or OSPF.

Either approach works. For ease of deployment (to keep things self-contained) we picked the second.

BST Architecture



The top half of the shim uses an SRM variant[1] that does application-level flooding (unicast or link level multicast).

The bottom half uses SCTP[2] which supports multihomed associations and TCP-friendly congestion control.

---

[1] S. Floyd, V. Jacobson, S. McCanne, "A Reliable Multicast Framework for Light-weight Sessions and Application Level Framework for Light-weight Sessions and Application Level Framing", Proc ACM SIGCOMM 95, Aug 1995 pp. 342-356.

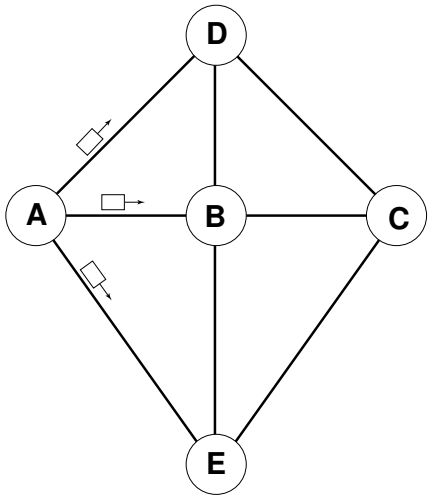
RFC 2887 — Multicast Design Space for Bulk Data Transfer

NORM — Nack-Oriented Reliable Multicast (draft-ietf-rmt-pi-norm-05)

[2] RFC2960 — Stream Control Transmission Protocol (Proposed Standard)

Why Flooding?

IGPs use flooding because it is the fastest, most reliable way to do a one-to-many dissemination.



- Doesn't require topology information.
- Doesn't concentrate traffic: at most one copy of any message per link, independent of network size.
- Highly reliable delivery: automatically takes advantage of all available spatial diversity.

Robust distributed agreement resulting from flooding allows implementation of any reliability model (graceful degradation, hot standby, cold standby/hot swap, k-for-n redundancy, etc.) and most service models (load balance, reserve capacity, pre-emptive allocation, etc.).

Next target: Simplifying IBGP Configuration

Replacing *n* parallel TCP connections with a single multipoint connection cuts down on traffic while improving reliability and convergence.

✘ But, by itself a multipoint transport doesn't solve BGP's configuration issues: every peer still has to be configured with the address of every other peer in the mesh and every configuration has to be updated when a router is added or removed from the mesh.

However, the multi-point delivery model *virtualizes* peers. I.e., a peer can send to a single address that reaches all peers without needing to know their individual identities. This mechanism can be used to bootstrap autoconfiguration, much like an IGP autoconfigures.

Autoconfiguration details

- peers are configured with the logical address to be used for the multipoint transport (like cisco's "peer-group" but address is group, not group leader).
  - ↪ logical address also needed for standard BGP (TCP based) initial capabilities negotiation so peers learn they both speak the new protocol and are part of the same group.
- "Hellos" are sent to this logical address (via unicast or link-level multicast) so adjacent peers learn about each other.

Autoconfig (cont.)

- Received “Hellos” are validated via a shared secret (same machinery and config info needed for RFC2385 BGP MD5 option).
  - ↪ Note that cryptographic peer and message validation is *much* cheaper to do with multipoint protocol (HMAC computed once per message, not once per message per peer).
- If config allows autodiscovery (via something like Juniper’s “dynamic peers”) then peering is automatically initiated.
- Once adjacent peerings are established, flooded messages from them tell about other peers which triggers the same validation and auto peering.

Next targets: unlimited scalability, fault tolerance, dynamic load balancing and simpler EBGP configuration

Multipoint protocol virtualizes receivers but not senders. If sending is bound to a particular box's IP address:

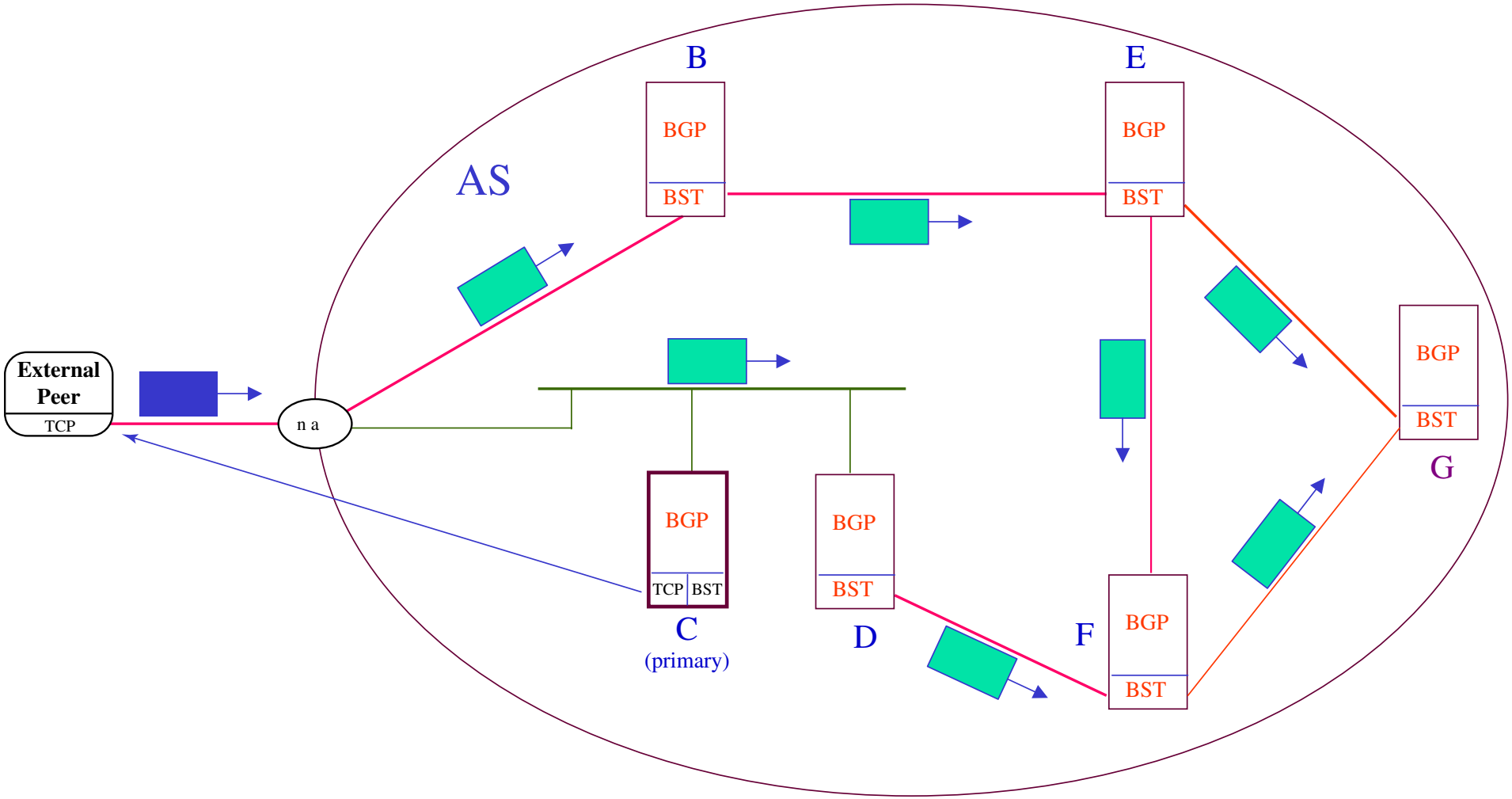
- EBGP peerings have to be architected
- Internal mesh has to expand when you add peering capacity (n^2 scaling issue)
- it's difficult to do failover and load balancing.

All these problems can be solved by using the multipoint protocol to virtualize the sender as well.

Virtualizing one end of an external / legacy peering

- collect a bunch of “BGP engines” (possibly with private addresses) into a mesh using the multipoint transport.
- Where the mesh connects to the outside world, put stateless “encapsulators” that take valid TCP BGP packets addressed to a particular unicast address (e.g., x.y.z.179), encapsulates them with a multipoint header and floods them through the mesh.
- If the flooded packet contains a SYN (new peering session) one of the engines elects itself as “primary” for that session.
  - ↪ If fault tolerance is wanted, another elects itself as secondary.
- Primary establishes peering, sending packets destined for external peer both to the mesh via multipoint protocol and directly to peer via unicast TCP.
  - ↪ Since secondary hears all packets from both sides of peering, it builds same Adj-Rib-In as primary and can take over if primary fails.

Virtualizing a peering (cont.)



Is this really practical?
How much of this is theory?

No theory. It's all practical.

- We've got a working prototype (using GateD as the hosting BGP implementation) that does everything described here.
  - ↪ prototype provides a validation of the transport-based approach and a quantification of how much impact the additional transport has on an existing BGP implementation.
- The engineer that built it (Kedar Poduri) will be demoing it at tonight's Beer-and-Gear.
- Please come by, take a look and ask questions.

So why are we here?

We want to convince you that BGP should be evolved, not replaced

- BGP embodies 15 years of painful lessons learned.
- BGP is a pretty good piece of work. It has problems but they *all* can be fixed by small, simple, evolutionary steps.

The evolution should be grounded in running code

- The internet used to be built on rough concensus and running code but there seems to be no emphasis on the latter today.
- “Running code” means actual operation in the Internet, not something that compiles or just works in the lab.