NETTLE: A LANGUAGE FOR CONFIGURING BGP NETWORKS

Andreas Voellmy

OGST - April 7, 2009

Prof. Dr. Otto Wilhelm Thomé, Flora von Deutschland, Österreich und der Schweiz. 1885, Gera, Germany

BGP: FLEXIBLE & DANGEROUS!

- BGP is the Internet's interdomain routing protocol; It is designed to be flexible and allow a variety of policies to be expressed by networks.
- This flexibility also makes BGP complex, and misconfiguration is common: Mahajan et al estimate that **50% of network outages are due to misconfigurations**;
- Furthermore, BGP routers typically play a crucial role in a network's connectivity, and misconfiguration can have serious consequences.

Wow, AS7007!

- *From:* Stephen A Misel
- *Date:* Fri Apr 25 13:20:40 1997

```
I happened to be in one of our 7505 routers this afternoon when POP -- all
of a sudden most of the internet disappeared!  I immediately thought it was
me, but looked around and saw this AS7007 broadcasting MY routes!   It
wasn't for all of our network space -- We have several /18's here, and it
seemed only the first /24 of each CIDR was affected.  When I found a
workstation at the end of the /18, we got the whois info for 7007 --
Florida Internet Exchange, and called them.

They claimed to have a customer broadcasting some bad routing information
and unplugged their router.  A few moments later, the internet stabilized
and I started seeing real routes.

Correct me if I'm wrong, but:

     (1)  We're going to read about this in EVERY computer magazine, newspaper
and TV as "the end of the internet?"

     (2)  Access lists by backbone providers *should* have prevented this.

     (3)  Does or does not the RADB and other routing registries (MCI's, etc)
prevent this?

I bet this hole will be patched up real soon!


Steve
- - - - - - - - - - - - - - - - - -
```

DSL'S TO THE RESCUE!

- Our overall goal is to **help operators configure BGP according to their intentions**, reducing misconfigurations and improving productivity.
- *Domain-specific languages* (*DSL*) help programmers construct correct programs by providing a language that matches the way domain experts think about their domain.
- A domain-specific embedded language (DSEL) is a DSL embedded in a host language; this technique reduces the cost of implementation and allows the DSL to inherit the general features of the host language.
- We have built Nettle, a DSEL in Haskell, in which BGP configurations for a whole network can be described, and a compiler which translates a Nettle program into router configuration files for the eXtensible Open Router Platform (XORP).

BGP KNOBS AND CONTROLS

- BGP provides lots of "controls" and "knobs"
- Nettle makes those "controls" available in Haskell
- We can now compose "controls" to make new "controls"

THIS TALK

- Intro to BGP: understanding BGP's "controls".
- Intro to Nettle: how we embed BGP's controls in Haskell.
- Three examples: defining high-level controls.

COMPUTER NETWORKS

- A *computer network* consists of a set of *nodes*, each having an *address*, and a set of *links* connecting nodes.
- **Forwarding** is the process of sending packets to the next hop node
- **Routing** is the process that establishes the paths along which forwarded packets flow. Routing results in each node having a **forwarding table**.

Destination	Outgoing interface
A	1
B	2
C	2

FORWARDING ON THE INTERNET

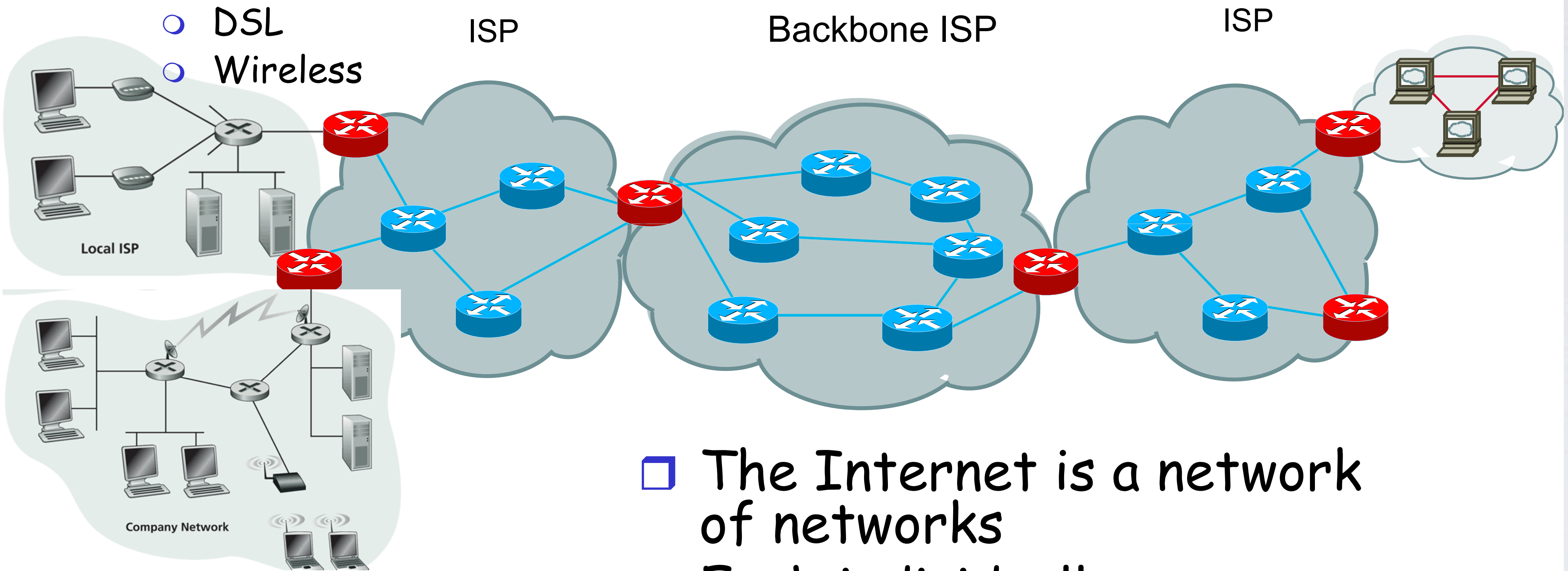
- *IP addresses* are 32-bit values, typically written as 4 bytes, as in a.b.c.d,
- An *address prefix*, is written a.b.c.d/e, and denotes the subset of IP addresses.
- Forwarding is by the "longest match", i.e. *most specific*

Prefix	Outgoing
0.0.0.0/0	1
1.1.0.0/16	2
1.1.1.0/24	3

Address	Longest Match
1.2.0.0	0.0.0.0/0
1.1.2.0	1.1.0.0/16
1.1.1.2	1.1.1.0/24

Internet Physical Infrastructure

Residential access
- Cable
- Fiber
- DSL
- Wireless

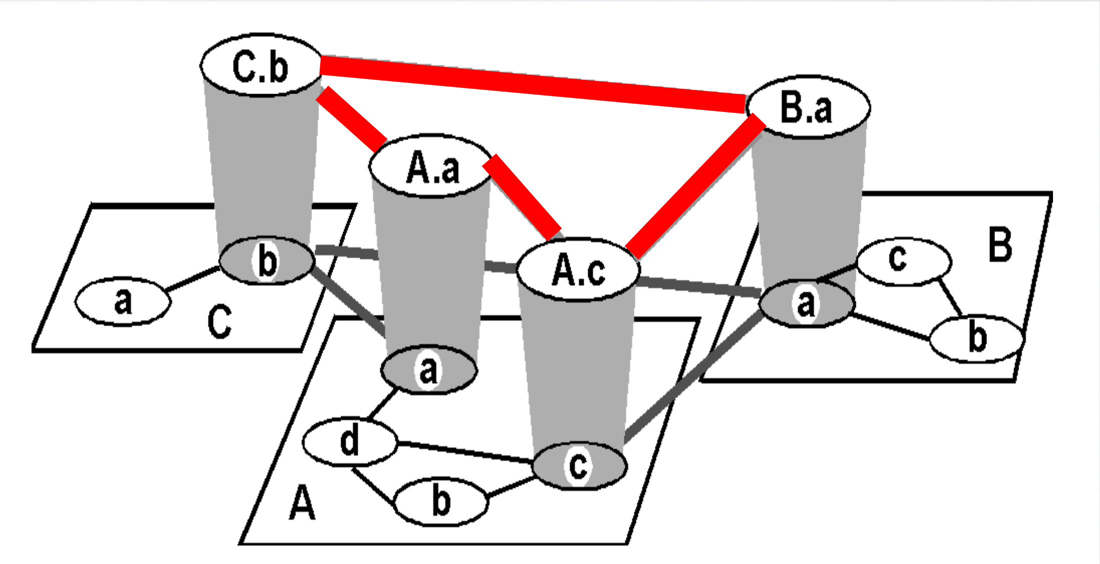


Campus access, e.g.,
- Ethernet
- Wireless

- The Internet is a network of networks
- Each individually administrated network is called an Autonomous System (AS)

BGP ROUTES

- BGP nodes *announce* routes to each other. These routes carry *attributes*, some of which are:
  - *Address Prefix*
  - *Next hop address*
  - *AS Path*, a sequence of AS numbers
  - *Community attributes*



EXAMPLE POLICIES

- Do not advertise routes heard from one provider to another -- *no transit*
- For a particular customer, only accept routes with the subnet they have been assigned -- *no hijacks*
- Prefer this route to my customer, but prepend my AS number several times when advertising it to others -- *customer wants me to use this route, but to discourage others (but not prevent entirely) from using it*.

BGP'S CONTROLS

- BGP nodes then repeat the following:
  - Collect announcements from neighbors;
  - **Choose some of the announced routes to consider - *Use Filter***
  - **Assigns numeric preference to remaining routes - *Preference Policy***
  - For each prefix announced, select the best one according to the decision process
  - Install best routes in the forwarding table of the router
  - **Choose some of the best routes to advertise - *Ad Filter***
  - **Advertise these with (potentially) modified attributes - *Ad Modifier***

BGP'S CONTROLS

- Two parts of BGP policy:
  - **Import policy**: which routes should we *use*?
  - **Export policy**: which routes should we offer to neighbors, and how good or bad should we make it look?

- useFilter : Neighbor × Route → Bool
- preference : Route → N
- adFilter : Neighbor × Route → Bool
- adModifier : Neighbor × Route → Route

BGP DECISION PROCESS

- The BGP decision process roughly implements shortest AS-path routing, while allowing networks to override this behavior by assigning non-default local preferences to routes.
- Of the routes known, a BGP node selects the *best* route for each prefix, by applying the *BGP decision process*, which is a lexicographic order on the following attributes:
  - highest local preference
  - shortest AS path length
  - ...
  - lowest router ID (used as a tie-breaker)

FORWARDING, REVISITED

Route	Prefix	NextHop	Pref
1	0.0.0.0/0	10.10.10.10	100
2	1.1.0.0/16	20.20.20.20	80
3	1.1.0.0/16	30.30.30.30	90

- To forward a packet to an address outside of the network:
  - Find the most specific BGP routes matching the address,
  - Use the best one.

Address	Route
1.1.1.1	3
1.2.0.0	1

PROTOCOL INTERACTION

- Networks running BGP also run an internal routing protocol and these protocols interact by *injecting* routes into from one protocol to the other. This process is called *route redistribution*.
- Some routes are known statically, and these may need to be injected into BGP. We view statically known routes as being computed by a *static* routing process.

NETTLE TUTORIAL

$$nettleProg = routingNetwork\ bgpNet\ staticNet\ redistPolicy$$

BGP NETWORK

$$bgpNet = bgpNetwork\ myASNumber\ bgpConns\ prefs\ usefilter\ adfilter\ admodifier$$

ROUTERS

$$
\begin{aligned}
&r1 = router\ r1xorp \\
&\quad \textbf{where}\ r1xorp &&= xorpRouter\ xorpBgpId\ xorpInterfaces \\
&\qquad\quad xorpInterfaces &&= [\,ifaceEth0\,] \\
&\qquad\quad ifaceEth0 &&= xorpInterface\ \texttt{"eth0"}\ \texttt{"data"}\ [\,virtualIfEth0Eth0\,] \\
&\qquad\quad virtualIfEth0Eth0 &&= \textbf{let}\ block \quad\ \ = address\ 200\ 200\ 200\ 2\ /\!\!/\ 30 \\
&&&\qquad\quad\ bcastAddr = address\ 200\ 200\ 200\ 3 \\
&&&\quad\ \textbf{in}\ vif\ \texttt{"eth0"}\ (vifAddrs\ (vifAddr\ block\ bcastAddr))
\end{aligned}
$$

BGP CONNECTIONS

$$\textit{conn1} = \textit{externalConn}\ \textit{r1}\ (\textit{address}\ 100\ 100\ 1\ 0)\ (\textit{address}\ 100\ 100\ 1\ 1)\ 3400$$
$$\textit{conn2} = \textit{internalConn}\ \textit{r1}\ (\textit{address}\ 130\ 0\ 1\ 4)\ \textit{r3}\ (\textit{address}\ 130\ 0\ 1\ 6)$$

ROUTE PREDICATES

$$nextHopEq\ (address\ 128\ 32\ 60\ 1) \vee taggedWith\ (5000 ::: 120)$$

$$\begin{aligned} &prefixInSet\ [address\ 128\ 32\ 60\ 0\ /\!/\ 24, address\ 63\ 100\ 0\ 0\ /\!/\ 16] \\ &\quad \wedge taggedWithAtLeastOneOf\ [5000 ::: 120, 7500 ::: 101] \end{aligned}$$

$$asSeqIn\ (repeat\ (i\ 7000)\ \rhd\ repeat\ any\ \rhd\ (i\ 3370\ |||\ i\ 4010)\ \rhd\ repeat\ (i\ 6500))$$

NEW PREDICATES

$$pathIs\ xs = asSeqIn\ \$\ foldr\ (\lambda a\ r \rightarrow repeat\ (i\ \ a)\ \ \triangleright\ \ r)\ empty\ xs$$

USE & AD FILTERS

$$reject\ (prefixEq\ (address\ 128\ 32\ 0\ 0\ /\!\!/\ 16))$$

$$\begin{aligned}
&usefilter\ c = \\
&\quad \textbf{if}\ c \equiv c1 \\
&\quad \textbf{then}\ reject\ ((prefixEq\ (address\ 128\ 32\ 0\ 0\ /\!\!/\ 16) \wedge taggedWith\ (5000 ::: 120)) \\
&\qquad\qquad\qquad \vee\ taggedWith\ (12345 ::: 100)) \\
&\quad \textbf{else}\ reject\ (asSeqIn\ (repeat\ any\ \rhd\ repeat\ (i\ 7000)\ \rhd\ repeat\ any))
\end{aligned}$$

ROUTE PREFERENCES

$$
\begin{aligned}
&cond\ (taggedWith\ (5000 ::: 120))\ 120\\
&\quad \$\ cond\ (taggedWith\ (5000 ::: 100))\ 100\\
&\quad \$\ cond\ (taggedWith\ (5000 ::: 80))\ 80\\
&\quad \$\ always\ 100
\end{aligned}
$$

ROUTE MODIFIERS

$$tag\ (5000 ::: 130)$$

$$prepend\ 65000$$

$$tag\ (5000 ::: 130)\ \rhd\ prepend\ 65000$$

$$cond\ (taggedWith\ (5000 ::: 120))\ (prepend\ 65000)\ (always\ (tag\ 1000 ::: 99))$$

$$\begin{aligned} adMod\ c\ &\mid c \equiv c1 &&= always\ (prepend\ 65000) \\ &\mid c \equiv c2 &&= always\ (prepends\ 2\ 65000\ \rhd\ tag\ (65000 ::: 120)) \\ &\mid otherwise &&= always\ ident \end{aligned}$$

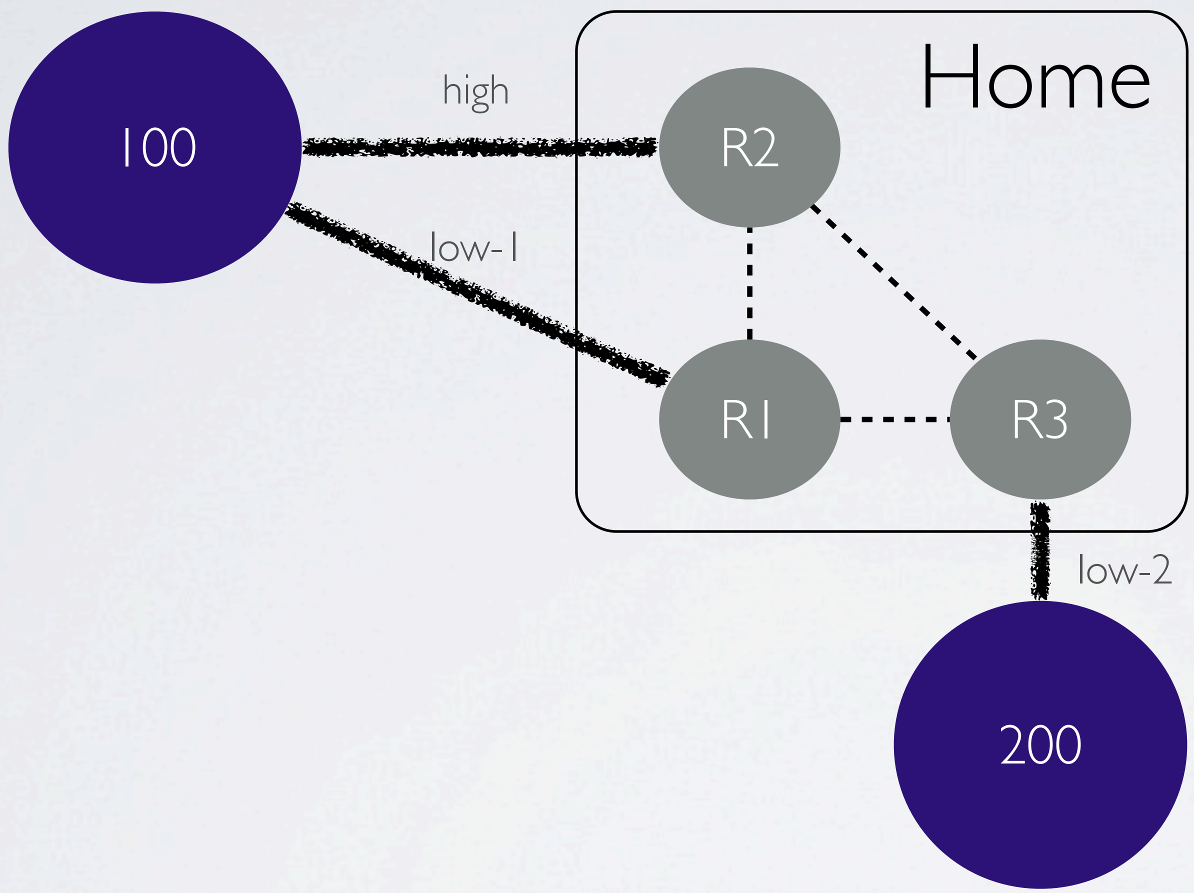
COMPILING

$$compile\ nettleProg\ r1$$

3 EXAMPLES

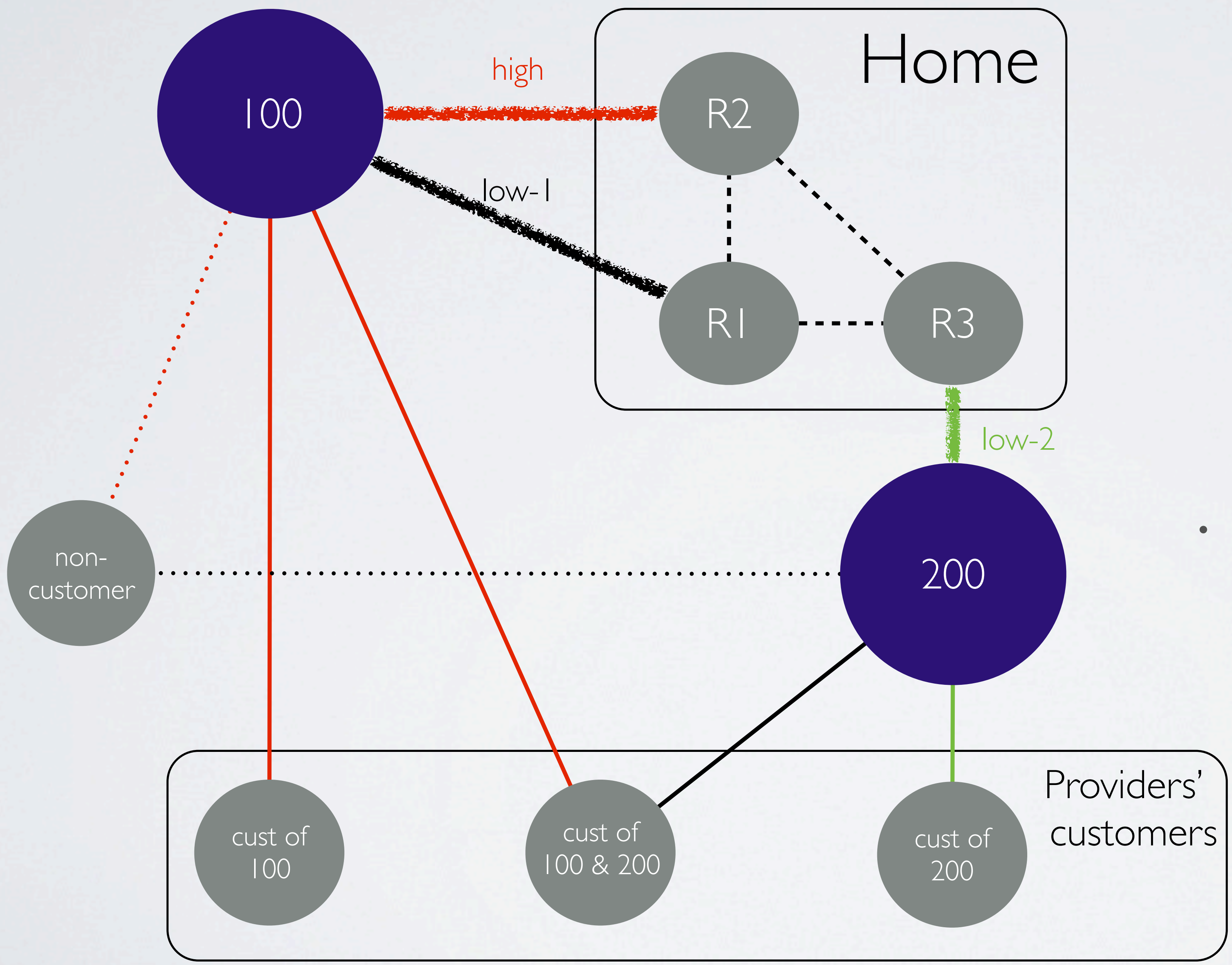
- Multi-homed “stub” network
- Hierarchical BGP
- Provider policies allowing customer-controlled policy

EXAMPLE: MULTI-HOMED NETWORK



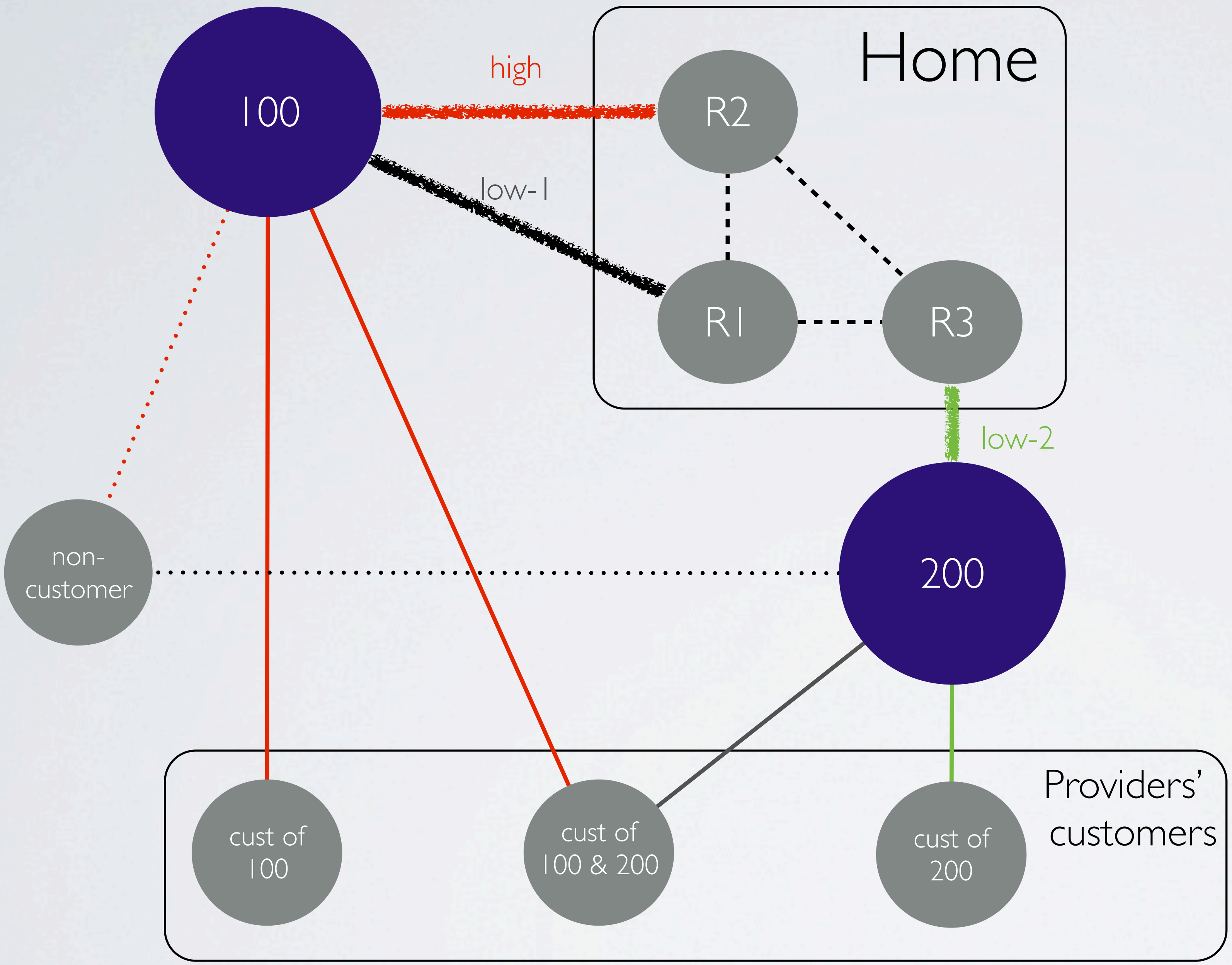
- Example from Zhang and Bartell, "BGP Design and Implementation"
- Objectives:
  - Balance traffic over high and low bandwidth links.
  - Use multiple links to provide robustness under link failures.
- Default routing only is too coarse.
- Request "default and partial" routes from providers.

EXAMPLE: MULTI-HOMED NETWORK



- Routes are of these types:
  - Customer of 100
  - Customer of 200
  - Customer of both 100 & 200
  - Neither customer of 100 nor customer of 200.
- To balance traffic we want
  - most traffic to flow over high, including non customers and customers of 100
  - traffic to customers of 200 (and not of 100) to flow over low-2.

EXAMPLE: MULTI-HOMED NETWORK



- Plan for link failures
  - When one link fails:
    - If low-2 fails, use high
    - If low-1 fails, traffic is unaffected
    - If high fails, balance over low-1 and low-2
  - When two links fails, use the remaining one.

EXAMPLE: MULTI-HOMED NETWORK

We can write the preference policy simply...

```
cond (nextHopEq (peerAddr connHigh)) 120 (always 100)
```

but this obscures the intentions of the policy.
We would like to express this more directly.

EXAMPLE: MULTI-HOMED NETWORK

- Balance on the most direct links available having the highest bandwidth

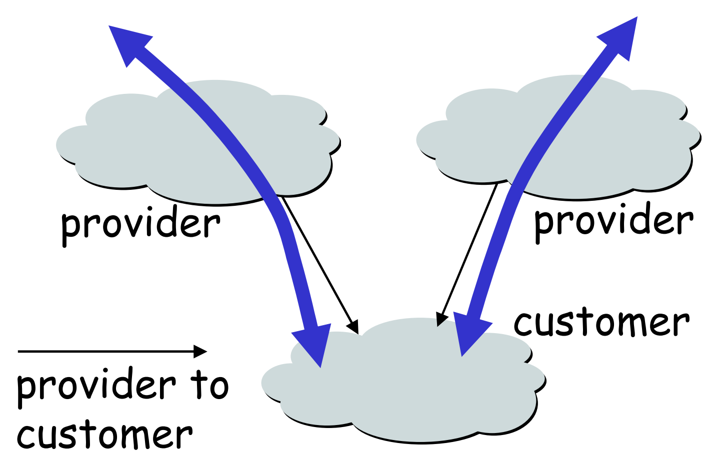
$$balanceByBandwidth :: [[BGPConnection]] \rightarrow Cond\ BGPT\ Preference$$

$$\begin{array}{l} \textbf{import}\ Nettle.MultiHomed \\ prefs = balanceByBandwidth\ [[connHigh], [connLow100, connLow200]] \end{array}$$

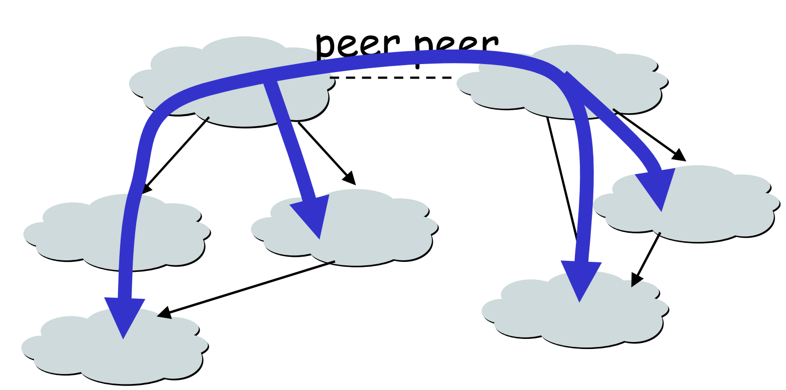
EXAMPLE 2: HIERARCHICAL BGP

- Gao and Rexford found that business relationships between ASes falls into two types:
  - customer-provider
  - peer-peer

- *Customer provider relationship*
  - a provider is an AS that connects the customer to the rest of the Internet
  - customer pays the provider for the transit service
  - e.g., Yale is a customer of AT&T and QWEST



- *Peer-to-peer relationship*
  - mutually agree to exchange traffic between their respective **customers**
  - there is no payment between peers



EXAMPLE 2: HIERARCHICAL BGP

- These two types strongly constrain BGP policy, both usage and advertising:
  - route preferences: customer > peer > provider
  - customer routes are advertised to all neighbors, whereas peer and provider routes may only be shared with customers
- Writing the routing configurations to satisfy these constraints is tedious and error-prone.
- We can generate these policies using Nettle!

HBGP IN NETTLE

$$\textbf{data}\ \mathit{PeerType} = \mathit{Customer} \mid \mathit{Peer} \mid \mathit{Provider}$$

$$\begin{aligned}
&\mathit{hbgpAdFilter} :: [(\mathit{ASNumber}, \mathit{PeerType})] \\
&\quad \rightarrow \mathit{BGPConnection}\ r \\
&\quad \rightarrow \mathit{Filter}\ \mathit{BGPT} \\
&\mathit{hbgpPrefs} \quad :: [(\mathit{ASNumber}, \mathit{PeerType})] \\
&\quad \rightarrow \mathit{PartialOrder}\ \mathit{ASNumber} \\
&\quad \rightarrow \mathit{Cond}\ \mathit{BGPT}\ \mathit{Preference}
\end{aligned}$$

HBGP IN NETTLE

$$home = 100; cust1 = 200; cust2 = 300; cust3 = 400$$
$$cust4 = 500; peer1 = 600; peer2 = 700; prov = 800$$

$$\begin{aligned} peerTyping = [&(cust1, Customer), (cust2, Customer), (cust3, Customer), \\ &(cust4, Customer), (peer1, Peer), (peer2, Peer), (prov, Provider)] \end{aligned}$$

$$\begin{aligned} &prefs = hbgpPrefs\ peerTyping\ basicPrefs \\ &\quad \textbf{where}\ basicPrefs = [(cust1, cust2), (cust1, cust4), (cust3, cust4), \\ &\qquad\qquad\qquad\qquad (peer2, peer1), (prov, prov)] \end{aligned}$$

$$adFilter = hbgpAdFilter\ peerTyping$$

COMPILING HBGP

```
term impterm13 {
 from {
  as-path: "^200|(200 [0-9][0-9]*( [0-9][0-9]*)*)$"
 }
 then {
  localpref: 105
 }
}
...
term impterm13 {
 from {
  as-path: "^300|(300 [0-9][0-9]*( [0-9][0-9]*)*)$"
 }
 then {
  localpref: 104
 }
}
...
```

COMPILING HBGP

```
term expterm7 {
 to {
  neighbor: 130.6.1.1
  as-path: "^700|(700 [0-9][0-9]*( [0-9][0-9]*)*)$"
 }
 then {
  reject
 }
}
term expterm8 {
 to {
  neighbor: 130.6.1.1
  as-path: "^800|(800 [0-9][0-9]*( [0-9][0-9]*)*)$"
 }
 then {
  reject
 }
}
```

EXAMPLE 3: CUSTOMER-ADJUSTABLE POLICY

- Allow customers to adjust preference of announced routes.

Community	Preference
1000:80	80
1000:100	100
1000:120	120

$$\begin{aligned}
&\mathit{cond}\ (\mathit{taggedWith}\ 1000 ::: 80)\ 80\\
&\quad \$\ \mathit{cond}\ (\mathit{taggedWith}\ 1000 ::: 100)\ 100\\
&\quad \$\ \mathit{cond}\ (\mathit{taggedWith}\ 1000 ::: 120)\ 120\\
&\quad \$\ \mathit{always}\ 100
\end{aligned}$$

CUSTOMER-ADJUSTABLE POLICY

- Allow customers to control advertising:
  - Suppress by AS number - community tag indicates which peers not to advertise to based on their AS Numbers.

Community	Suppress when advertising to
1000:5500	5500
1000:5600	5600
...	...

$$reject\ (\lambda connection \rightarrow taggedWith\ 1000 ::: (asNumber\ \ connection))$$

CUSTOMER-ADJUSTABLE POLICY

- Prepending by AS Number - community tag indicates how many times to prepend when advertising a route to a specific neighbor

Community	AS	Times to Prepend
1001:5500	5500	1
1002:5500	5500	2
...		
1001:5600	5600	1

CUSTOMER-ADJUSTABLE POLICY

- We can write functions these policies in Nettle, and then apply them by applying these functions:

$$adjustablePrefs\ homeASNum\ [80, 100, 120]\ (always\ 100)$$

$$adjustablePrepending\ homeASNum\ [1, 2, 3, 4, 5]\ [5500, 5600]\ (always\ ident)$$