AS-Wide Inter-Domain Routing Policies: Design and Realization

Anja Feldmann
Technisch Universität München

Hagen Böhm
Deutsche Telekom AG

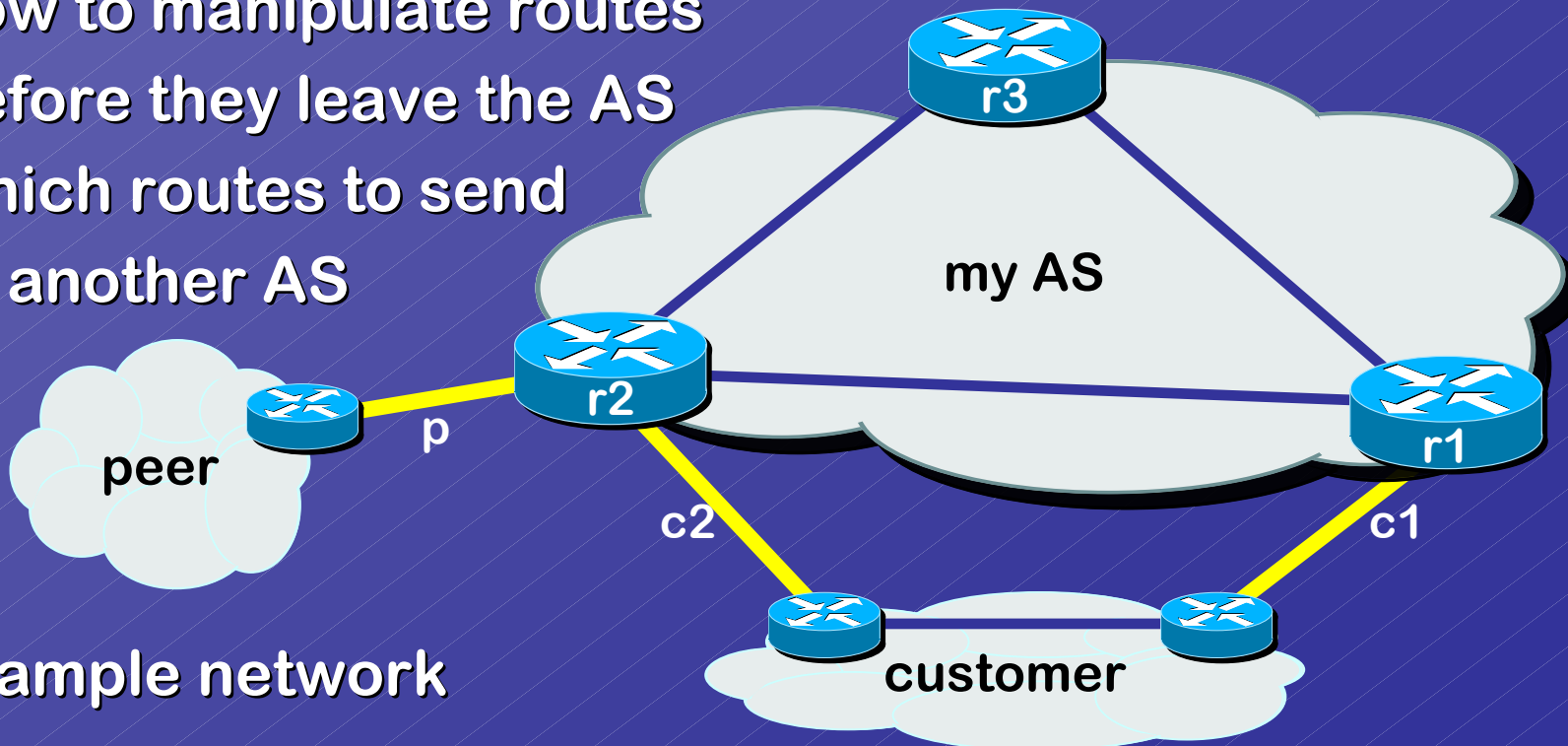
Olaf Maennel
Technische Universität München

Christian Reiser
Technische Universität München

Rüdiger Volk
Deutsche Telekom AG

Routing policy

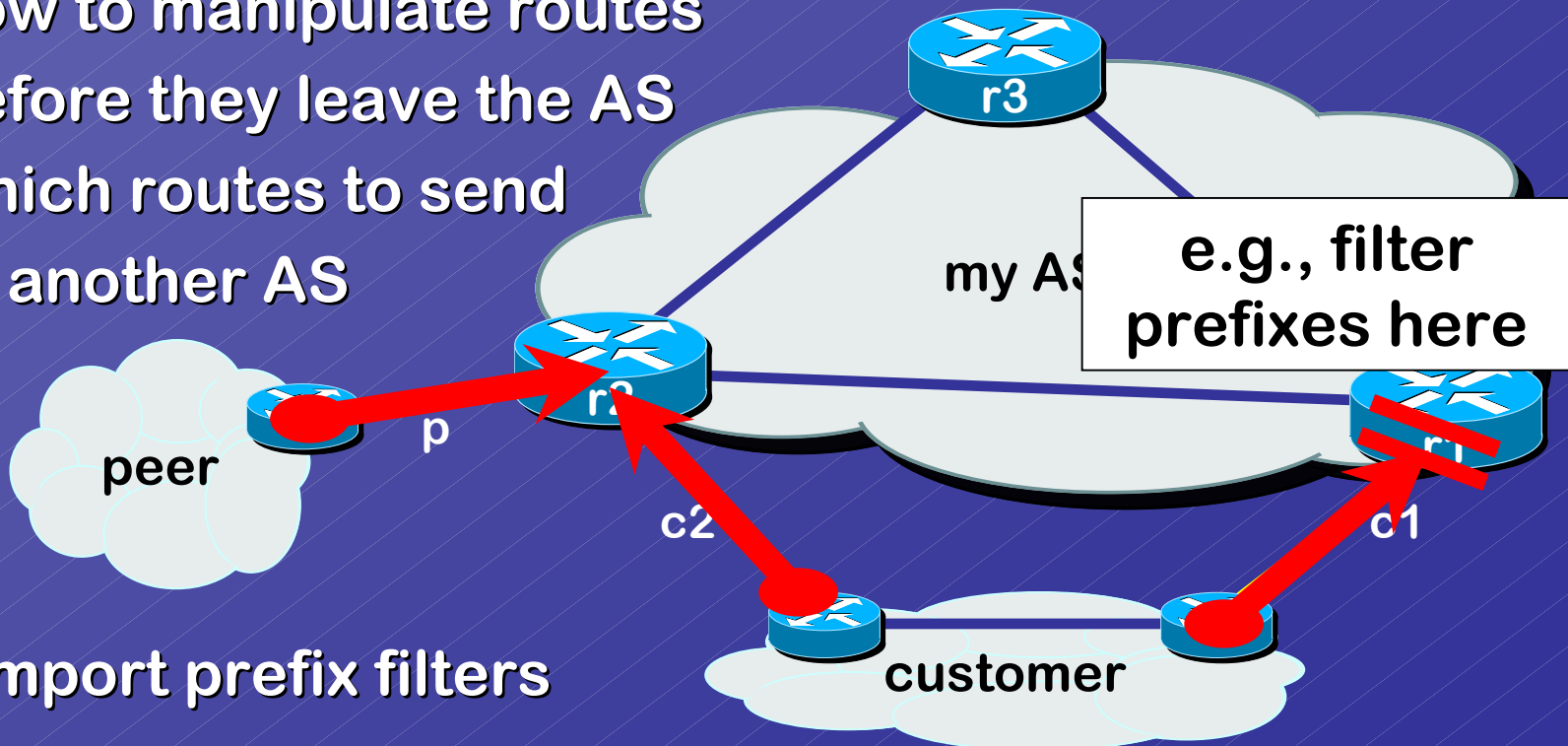
- reflects the goals of the network provider:
  - which routes to accept from other ASes
  - how to manipulate the accepted routes
  - how to propagate routes through network
  - how to manipulate routes before they leave the AS
  - which routes to send to another AS



Example network

Routing policy

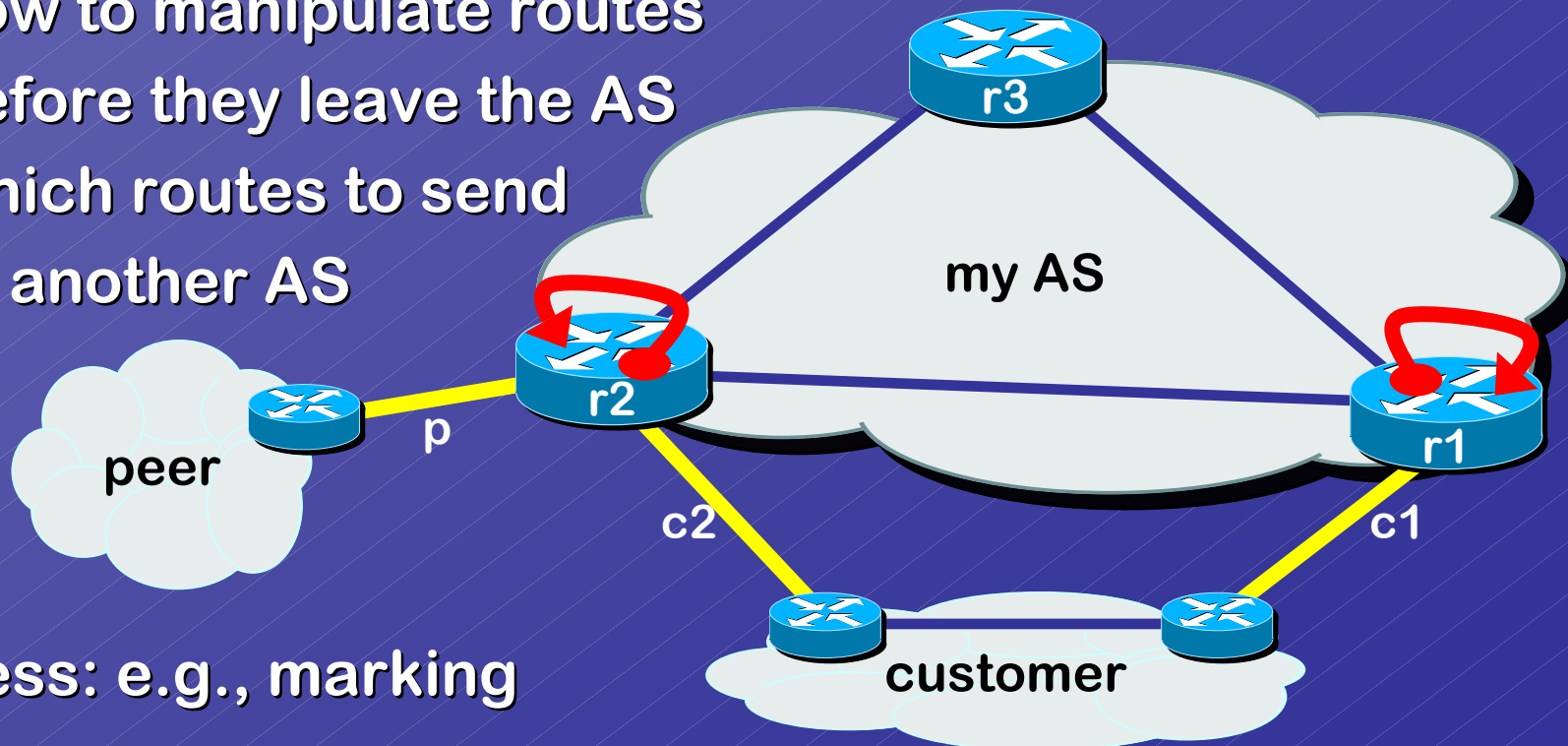
- reflects the goals of the network provider:
  - which routes to accept from other ASes
  - how to manipulate the accepted routes
  - how to propagate routes through network
  - how to manipulate routes before they leave the AS
  - which routes to send to another AS



e.g., import prefix filters

Routing policy

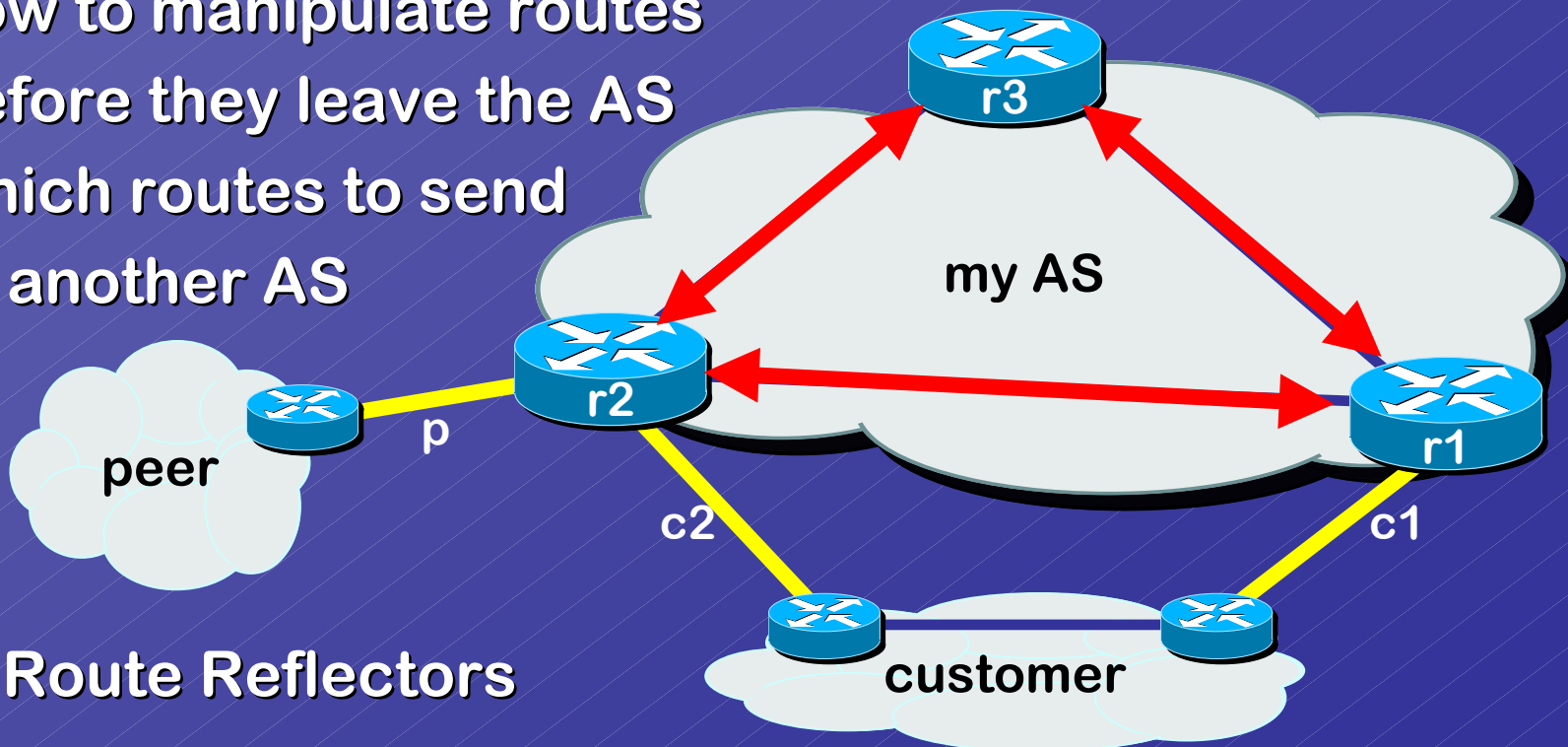
- reflects the goals of the network provider:
  - which routes to accept from other ASes
  - how to manipulate the accepted routes
  - how to propagate routes through network
  - how to manipulate routes before they leave the AS
  - which routes to send to another AS



ingress: e.g., marking

Routing policy

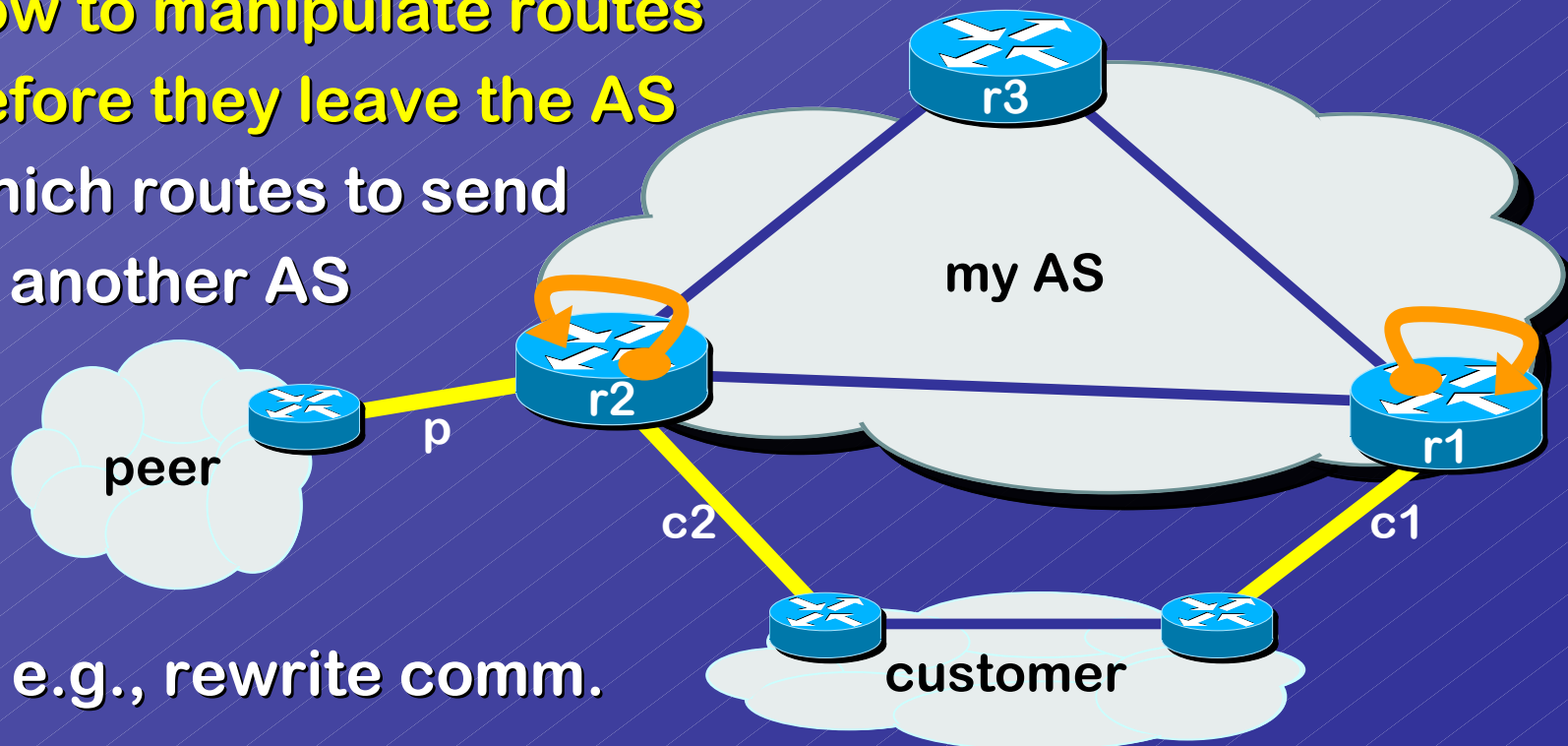
- reflects the goals of the network provider:
  - which routes to accept from other ASes
  - how to manipulate the accepted routes
  - how to propagate routes through network
  - how to manipulate routes before they leave the AS
  - which routes to send to another AS



e.g., Route Reflectors

Routing policy

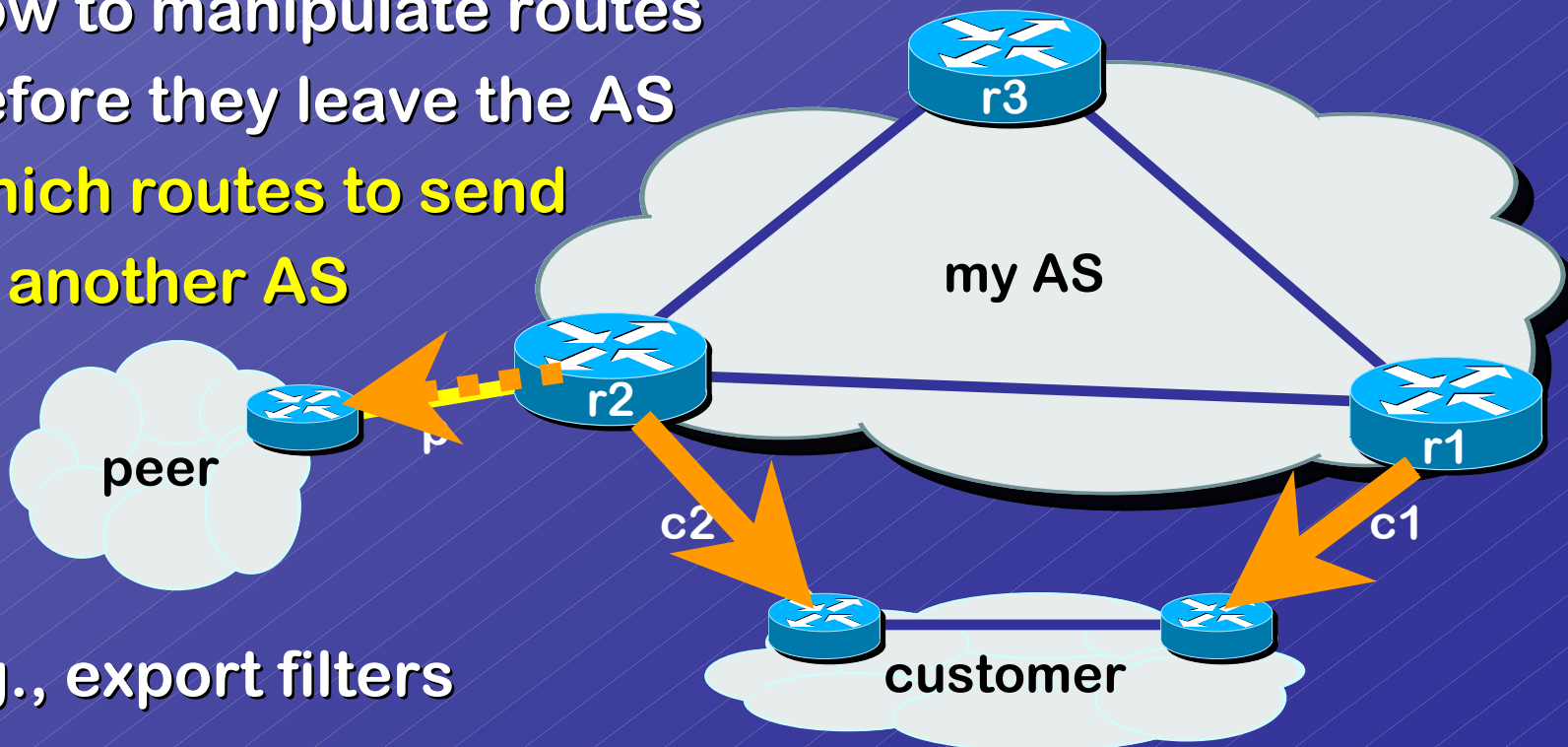
- reflects the goals of the network provider:
  - which routes to accept from other ASes
  - how to manipulate the accepted routes
  - how to propagate routes through network
  - **how to manipulate routes before they leave the AS**
  - which routes to send to another AS



egress: e.g., rewrite comm.

Routing policy

- reflects the goals of the network provider:
  - which routes to accept from other ASes
  - how to manipulate the accepted routes
  - how to propagate routes through network
  - how to manipulate routes before they leave the AS
  - **which routes to send to another AS**



e.g., export filters

Routing policy

- Current state of the art:
  - ill-specified (e.g., policy database is the network itself)
  - undergoes constant adjustments
  - customer specific
  - conglomerate of BGP statements
  - realized by manual configuration of routers

- Goal: increased abstraction level
  - AS-wide policy specification

Routing policy: Examples

Policy/service examples:

- honor business relationships
  (e.g., customers get full-table; peers only customer prefixes)
  (e.g., prefer customer routes over peer routes over upstream routes)
- allow customers a choice of route
  (e.g., on customer request do not export prefix to AS x, etc.)
- enable customer traffic engineering
  (e.g., prepend x times to all peers or to specified AS)
- enable DDoS defense for customers
  (e.g., blackholing by rewriting the next hop)
- …

➢ An AS-wide routing policy consists of
  – atomic units: policies as well as services

Routing policy: Specific example

- Blackholing
  - trigger: community
  - mechanism: rewrite next-hop of route
  - safeguard: no-export community
- Implementation
  - on ingress
    - allow blackhole community for specific customers
    - filter blackhole community for other BGP neighbors
    - rewrite next hop
  - on egress
    - filter blackhole route at egress

Expressing an atomic unit

- **Identify a BGP session set:**
  - network specific:
    via parameters, selected services, etc.
  - related to network elements
  - *vendor independent*
- **Apply some filter:**
  - BGP specific: prefix, community, AS lists
  - *vendor specific*
- **Perform some action:**
  - BGP specific: rewrite parameters,
    add community
  - *vendor specific*

Policy realization

- System components:
  - network module
    - description of network elements
  - policy and service module
    - abstract definition of individual routing policies
  - router backend module
    - library of vendor specific code fragments
  - configurator
    - input: above modules
    - output:
      - appropriate vendor specific configlets
      - (alternative) RPSL

Network module

- Task:
  - captures current network components
  - captures current service subscriptions

- Top-level elements:
  - router
  - BGP neighbor
  - BGP session
  - services

Example: <router>

```
<router>
   <name> TEST-R1 </name>
   <loopback> 1.0.0.1 </loopback>
   <location>
     <city> Munich </city>
     <country> DE </country>
     <region> Europe </region>
   </location>
   <system>
     <hw> M320 </hw>
     <sw> JunOS 6.2R2.4 </sw>
   </system>
</router>
```

Example: <bgpneighbor>

```
<bgpneighbor>
   <name> Neighbor P </name>
   <neighborAS> 2 </neighborAS>
   <neighbortype> peer </neighbortype>
   <session> p </session>
   <filter_import> AS2:RS-I </filter_import>
   <filter_export> AS2:RS-E </filter_export>
   <services></services>
</bgpneighbor>
```

Example <bgpsession>

```
<bgpsession>
   <name> p </name>
   <myrouter> TEST-R2 </myrouter>
   <remoteIPaddr> 1.1.1.1 </remoteIPaddr>
   <services></services>
</bgpsession>

<bgpsession> <name> C </name>
   ...
   <services>
    <egress_med>
        <case><filter> SET-2000 </filter>
            <med_value> 2000 </med_value></case>
        <case><filter> SET-3500 </filter>
            <med_value> 3500 </med_value></case>
        <default><med_value> 100 </med_value></case>
    </egress_med>
   </services>
</bgpsession>
```

Policy and service module

- Task:
  - define each unit of the AS-wide policy
    - using an intermediate abstraction
    - independent of any other part of the policy
- Realization:
  - express each unit by
    - selecting session sets and then
    - applying sets of BGP operations
      (the BGP operations are referenced by names!)
- Top-level elements:
  - policy
  - service

Example routing policy

```
<enforced_policies>
   <name> peering </name>
</enforced_policies>
<availabe_services>
   <name> blackhole </name>
</available_services>

<service>
   <name> blackhole </name>
   <parameter> <blackhole_set/> </parameter>
   <sessionset>
      <direction> ingress </direction>
      <condition> IF($service.blackhole) </condition>
      <task>
         <fragment> ingress_blackhole_community </fragment>
         <fragment> ingress_blackhole_accept </fragment>
      </task>
      <default>
         <fragment>ingress_blackhole_community_deny</fragment>
      </default>
   </sessionset>
</service>
...
```

Backend module

- Task:
  - capsules vendor specifics
  - provide library of named fragments
  - each fragment capsules a set of BGP statements

- Realization:
  - for each vendor provide vendor specific realization
  - access of network (BGP session / router) values:
    - via “variables” (replaced in session context):
      - e.g., `$session.neighbortype`
      - e.g., `community_filter_name(“ios”, “blackhole”)`

Back end module <fragment>

```
<fragment>
  <name>ingress_blackhole_community</name>
  <IOS>
    <bgp>
      neighbor $remoteIPaddr route-map $routemapname_in in
    </bgp>
    <routemap>
     <map>
      route-map $routemapname_in permit $routemappriority
        filter($blackhole_set)
        match community community_filter_name("blackhole")
        set ip next-hop 172.17.17.172
        set community no-export community_value("blackhole")
      <routemapaction>
        continue
      </routemapaction>
     </map>
    </routemap>
    …
```

Back end module <fragment>

```
    …
    <list>
      ip community-list expanded community_filter_name(
        "blackhole") permit community_value("blackhole")
      <type>community</type>
    </list>
  </IOS>
  <JUNOS>
    …
  </JUNOS>
  <RPSL>
    …
  </RPSL>
</fragment>
```

Configurator

- Task:
  - parse three databases (one for each module)
  - check consistency
  - combine individual atomic unit in appropriate routing policy for each BGP session
  - generate router configuration pieces
    (e-BGP part of router config, including filter lists for Cisco, Juniper, and RPSL)
- Realization:
  - CISCO: "continue" and/or folding
  - JUNIPER: "next policy"
  - RPSL: "refine"
- Status:
  - prototype operational at Deutsche Telekom

Configurator output (1.)

```
bgp 1
  neighbor 2.1.1.2 remote-as 2
  neighbor 2.1.1.2 route-map c1_routemap_in in
  neighbor 2.1.1.2 route-map c1_routemap_out out
!
route-map c1_routemap_out deny 100
  match ip address prefix-list martians
route-map c1_routemap_out permit 200
  set comm-list out_fltr_communities delete
  continue 300
route-map c1_routemap_out permit 300
  match community export_all
!
route-map c1_routemap_in deny 500
...
ip community-list expanded in_fltr_communities permit _1:.*_
ip community-list expanded in_fltr_communities permit 64900:.*
...
!
ip prefix-list c1-import permit 2.1.1.0/22 ge 24 le 24
ip prefix-list c1-blackhole permit 2.1.1.0/24 ge 32 le 32
ip prefix-list c1-blackhole permit 2.1.1.0/24
ip prefix-list martians permit ...
```

Configurator output (2.)

```
route-map c1_routemap_out deny 10
  match ip address prefix-list martians
route-map c1_routemap_out permit 15
  match community export_all
  set comm-list out_fltr_communities delete
!

route-map c1_routemap_in deny 10
  match ip address prefix-list martians
route-map c1_routemap_in permit 76
  match ip address prefix-list c1-import-c1-blackhole
  match community blackhole
  set comm-list in_fltr_communities delete
  set ip next-hop 172.24.42.172
  set community 1:1 no-export additive
route-map c1_routemap_in permit 77
  match ip address prefix-list c1-blackhole
  match community blackhole
  set comm-list in_fltr_communities delete
  set ip next-hop 172.24.42.172
  set community 1:1 no-export additive
...
!
```

Summary

Benefits of the system for an ISP:

- explicit specification of the AS-wide routing policy independent of the network!
- separation of the routing policy in atomic units
- easy introduction of new services
- easy to add customers, routers, etc…
- easy to take advantage of new router features
- respects knowledge domains
- automatically generates appropriate router configlets

Requirements

- Abstraction
  - policies should be expressible via high-level language primitives.
- Customizable
  - parameters depend on peer (e.g., prefix filters from IRR)
- Modularity / Separability
  - policies needs to be independent from vendor code (e.g., refine-statements in RPSL, cisco “continue”, juniper “next policy”)
- Extensibility
  - add new or change policies or services
  - should be possible to take advantage of new router features
- Debugability
  - e.g., prefix-filter, and community-lists should have same name (number) on all routers.
- Testability
  - automatically generate the outcome of policy combinations for exploration in tests (work-in-progress)