Alloy For Configuration

Lecture 6

CS 598D, Spring 2010
Princeton University

Sanjai Narain
narain@research.telcordia.com
908 337 3636

Outline

- Alloy is a great tool for learning about first-order logic
- The difference between model finding and model checking. Alloy is a model finder
- Alloy by example
- Plan for solving configuration problems with Alloy
- Fault-Tolerant VPN requirements in English
- Their formalization in Alloy
- Solving VPN configuration problems
- Limitations of approach

Model Finder Vs. Model Checker

- An **interpretation** of a formula is a value of the free variables in it
- Consider the formula $\forall x.\ \alpha(x) \supset \beta\ (x)$
- Its free variables are α and β that range over unary relations over a non-empty domain
- A **model** of a formula is an interpretation in which the formula is true
- Over the domain of living things, the value
  - α=human, β=mortal is a model of the above formula
  - But not α=mortal, β=human
- A formula is **valid in a domain** if it is true for all interpretations over that domain
- A formula is **valid** if it is valid in all domains, e.g.

$$\underline{\forall x.\ \alpha(x) \supset \beta\ (x)} \wedge \underline{\forall x\ .\beta(x) \supset \gamma\ (x)} \supset \underline{\forall x.\ \alpha(x) \supset \gamma\ (x)}$$

$$p \supset (q \supset p)$$

- A model finder finds a model of a formula in a given domain
- Alloy works by translating a first-order logic formula into a Boolean one; using a SAT solver to find a model of the Boolean formula; translating that model back into a model of the original formula
- Alloy only finds models in *finite* domains otherwise the Boolean formula would not be finite
- In the domain {1,..,k}, $\forall x.\ \alpha(x) \supset \beta\ (x)$ is translated into $\alpha(1) \supset \beta\ (1) \wedge \ldots \wedge \alpha(k) \supset \beta\ (k)$
- A model checker checks whether a formula is true in an interpretation
  - Typically, formulas are temporal logic ones and interpretations are state machines
  - But, they don't have to be

Alloy By Example

Requirements in English

- For every router x there is an interface y whose chassis is x
- No two non-equal interfaces on the same router are placed on the same subnet
- These are first-order logic requirements because they quantify over individual variables

Requirements in Alloy

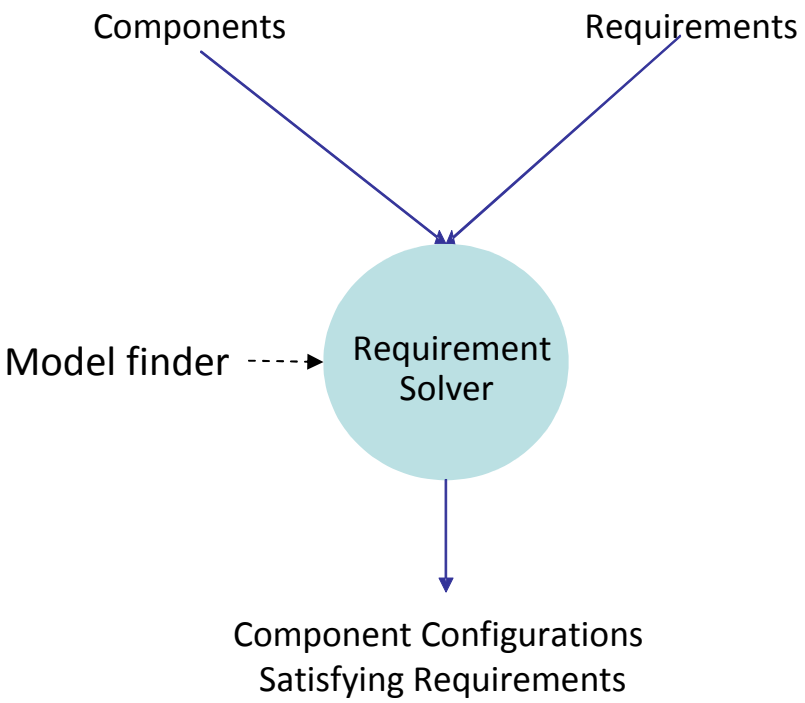
```
sig router {}
sig subnet{}
sig interface {chassis: router, network: subnet}

pred spec ()
    {all x:router | some y:interface | y.chassis = x}
    {no disj x1,x2:interface |
            x1.chassis=x2.chassis &&
            x1.network = x2.network}
```

run spec for <u>1 router, 2 subnet, 2 interface</u> (scope)

Model

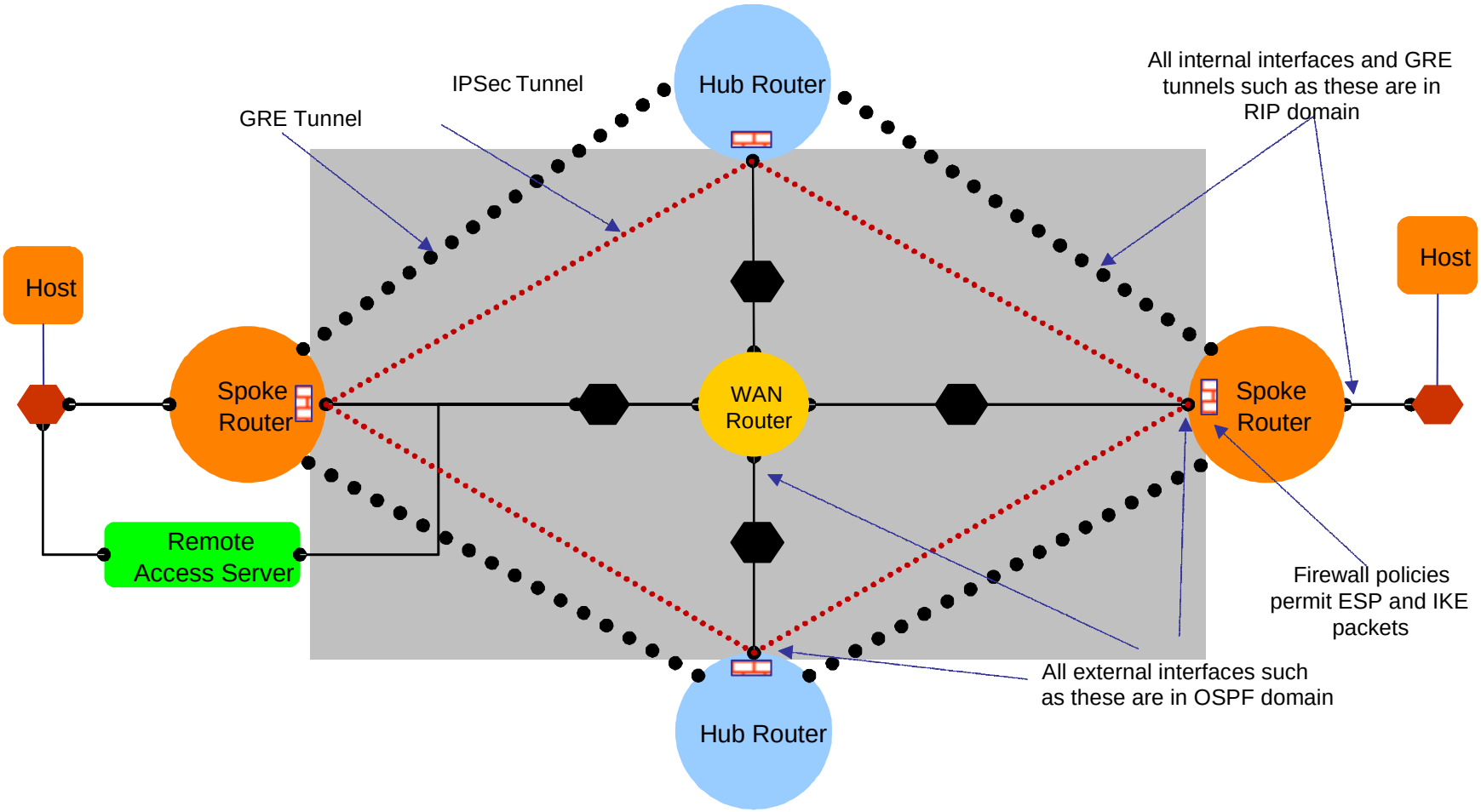
```
chassis : = {interface_0 -> router_0, interface_1 ->
    router_0}
network : = {interface_0 -> subnet_1, interface_1 ->
    subnet_0}
```

Plan



- **Configuration Synthesis:** Take output of Solver
- **Requirement Strengthening:** Solve for conjunction of current and new requirements
- **Component Addition.** Find model of requirement in current scope extended with new component
- **Requirement Verification.** To show S cannot hold when R does show $R \wedge S$ is unsolvable
- **Configuration error detection and repair were rudimentary:**
  - UNSAT-core concept was unknown (at Telcordia) at this time (2004-5)

Fault-Tolerant VPN



VPN Requirements

- **RouterInterfaceRequirements**
  1. Each spoke router has internal and external interfaces
  2. Each access server has internal and external interfaces
  3. Each hub router has only external interfaces
  4. Each WAN router has only external interfaces

- **SubnettingRequirements**
  5. A router does not have more than one interface on a subnet
  6. All internal interfaces are on internal subnets
  7. All external interfaces are on external subnets
  8. Every hub and spoke router is connected to a WAN router
  9. No two non-WAN routers share a subnet

- **RoutingRequirements**
  10. RIP is enabled on all internal interfaces
  11. OSPF is enabled on all external interfaces

- **GRERequirements**
  12. There is a GRE tunnel between each hub and spoke router
  13. RIP is enabled on all GRE interfaces

- **SecureGRERequirements**
  14. For every GRE tunnel there is an IPSec tunnel between associated physical interfaces that secures all GRE traffic

- **AccessServerRequirements**
  15. There exists an access server and spoke router such that the server is attached in “parallel” to the router

- **AccessControlPolicyRequirements**
  16. Each hub and spoke external interface permits esp and ike packets

Alloy Signatures For VPN

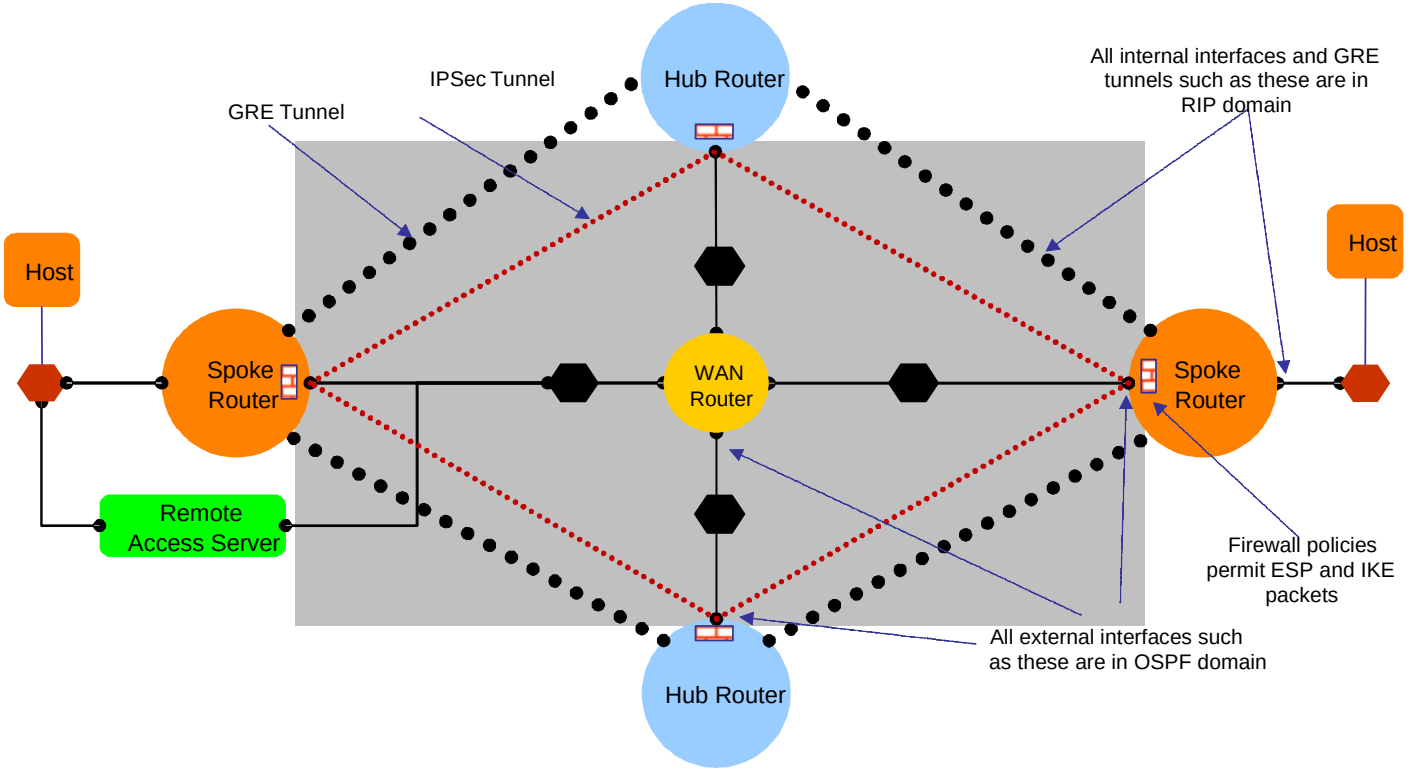
```
sig router {}
sig wanRouter extends router {}
sig hubRouter extends router {}
sig spokeRouter extends router {}
sig accessServer extends router {}
sig legacyRouter extends router {}

sig interface {routing:routingDomain}

sig physicalInterface extends interface {
    chassis: router,
    network: subnet}

sig internalInterface extends physicalInterface {}
sig externalInterface extends physicalInterface {}
sig hubExternalInterface extends externalInterface {}
sig spokeExternalInterface extends externalInterface
    {}
```

```
sig subnet{}
sig internalSubnet extends subnet{}
sig externalSubnet extends subnet{}


sig ipsecTunnel {
    local: externalInterface,
    remote: externalInterface,
    protocolToSecure: protocol}

sig greTunnel {
    localPhysical: externalInterface,
    routing: routingDomain,
    remotePhysical: externalInterface}


sig ipPacket {
    source:interface,
    destination:interface,
    prot:protocol}
```

Specifying GRE Requirement 1 in Alloy

Between every hubExternalInterface x and spokeExternalInterface y there is a greTunnel whose local physical is x and remotePhysical is y, or vice versa

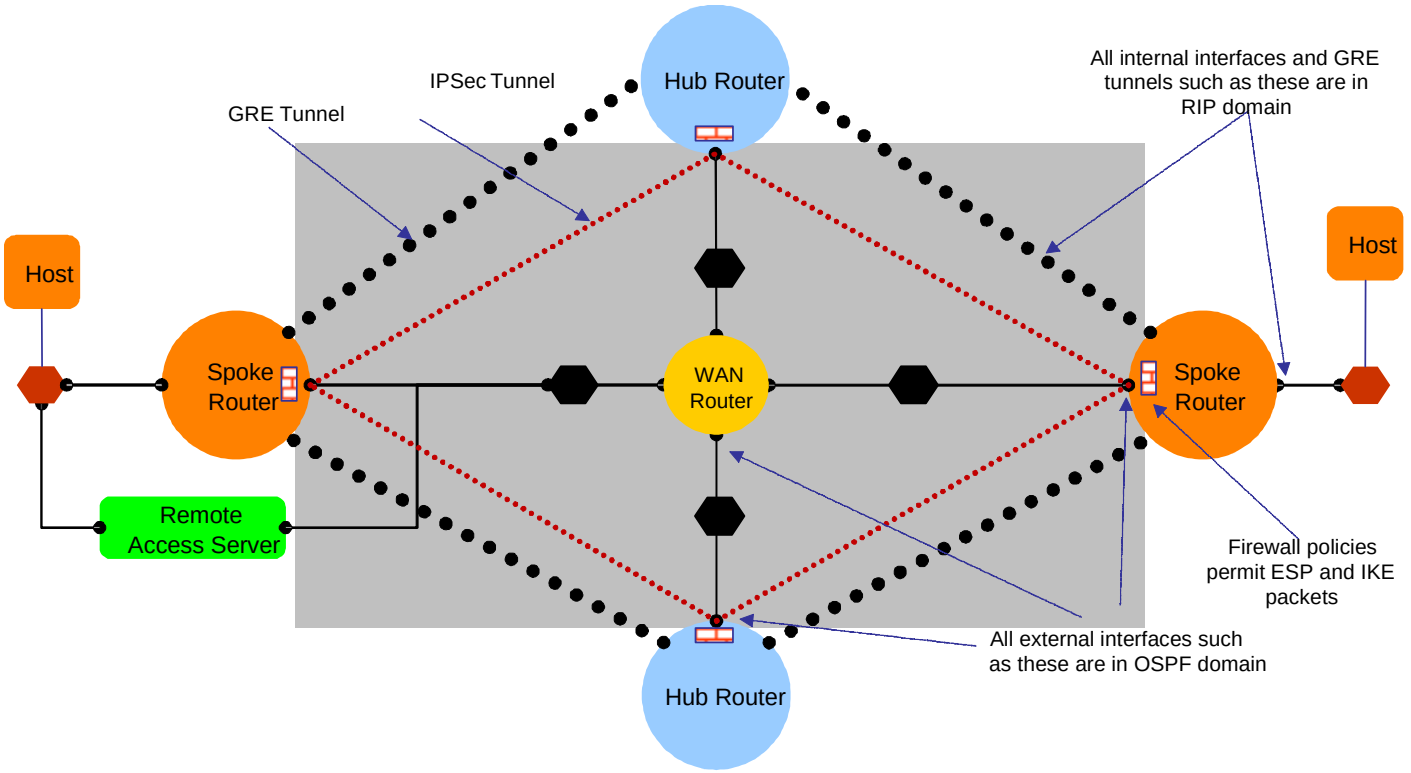
{all x:hubExternalInterface, y:spokeExternalInterface | some g:greTunnel |
(g.localPhysical=x && g.remotePhysical=y) or
(g.localPhysical=y && g.remotePhysical=x)}



Specifying SecureGRE in Alloy

For every greTunnel g there is an ipsecTunnel p that secures the gre protocol and whose endpoints are the same as the physical endpoints of g.

```
{all g:greTunnel |
    some p:ipsecTunnel | p.protocolToSecure=gre &&
    ((p.local = g.localPhysical && p.remote = g.remotePhysical) or
     (p.local = g.localPhysical && p.remote = g.remotePhysical))}
```



Configuration Synthesis: Solve PhysicalSpec For A Scope

Pred PhysicalSpec () {

RouterInterfaceRequirements ()

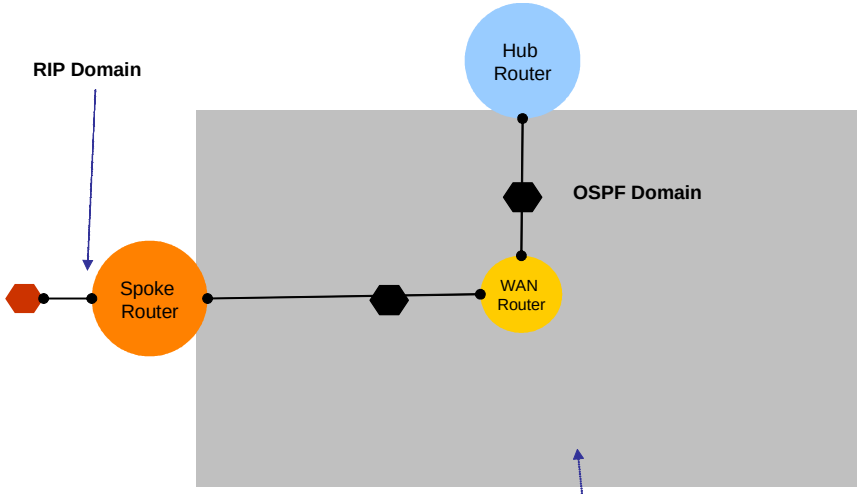
SubnettingRequirements ()

RoutingRequirements ()}

Define a scope:

1 hubRouter, 1 spokeRouter, 1 wanRouter, 1 internalInterface, 4 externalInterface, 1 hubExternalInterface, 1 spokeExternalInterface,

1 ripDomain, 1 ospfDomain, 3 subnet, 0 legacyRouter.



Solution

routing : =

{externalInterface_0 -> ospfDomain_0,
externalInterface_1 -> ospfDomain_0,
hubExternalInterface_0 -> ospfDomain_0,
internalInterface_0 -> ripDomain_0,
spokeExternalInterface_0 -> ospfDomain_0}

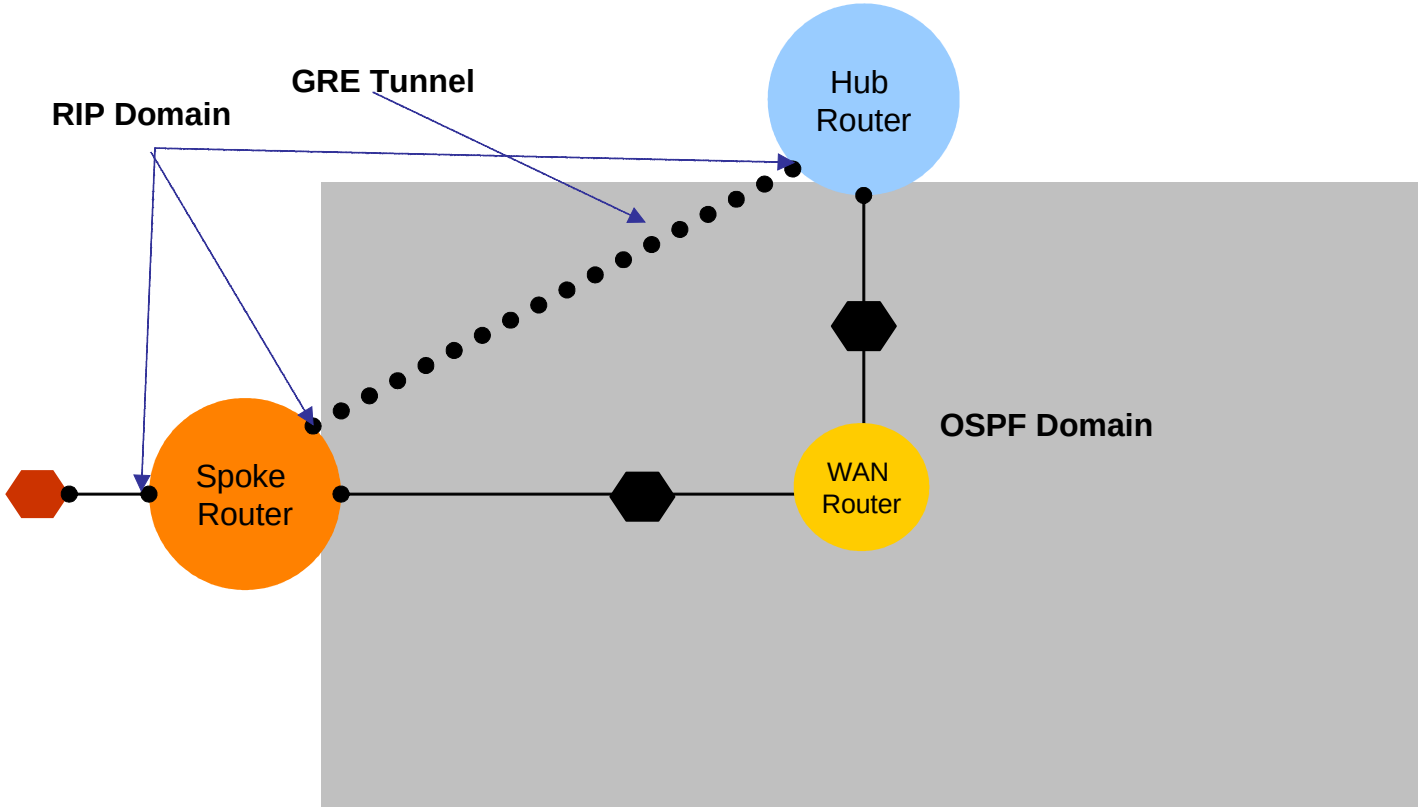
chassis : =

{externalInterface_0 -> wanRouter_0,
externalInterface_1 -> wanRouter_0,
hubExternalInterface_0 -> hubRouter_0,
internalInterface_0 -> spokeRouter_0,
spokeExternalInterface_0 -> spokeRouter_0}

network : =

{externalInterface_0 -> externalSubnet_1,
externalInterface_1 -> externalSubnet_0,
hubExternalInterface_0 -> externalSubnet_0,
internalInterface_0 -> internalSubnet_0,
spokeExternalInterface_0 -> externalSubnet_1}

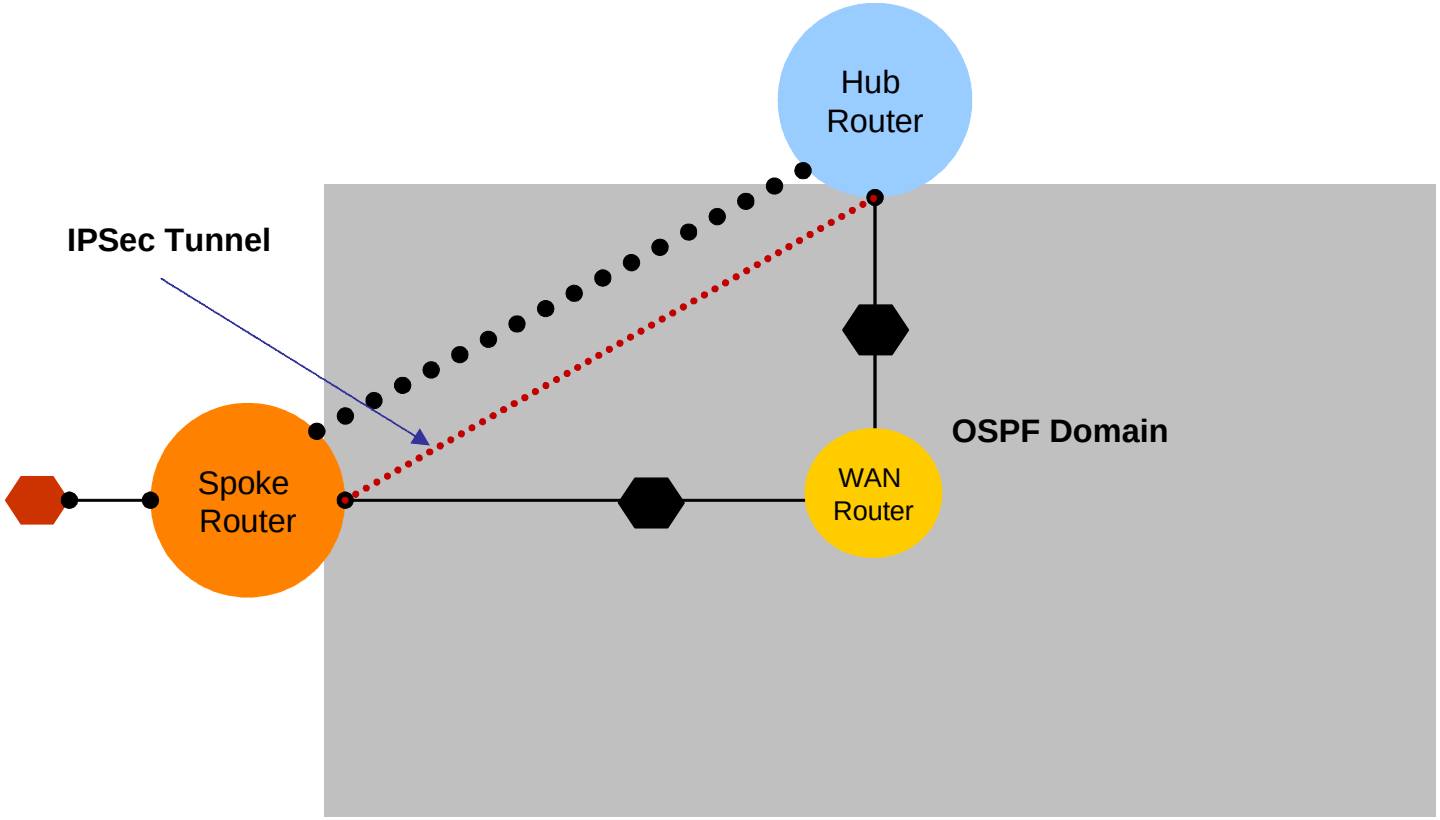
Requirement Strengthening



(PhysicalSpec ∧ GRERequirements)

Alloy automatically sets up the GRE tunnel between the spoke and hub router and enables RIP routing on the GRE tunnel.

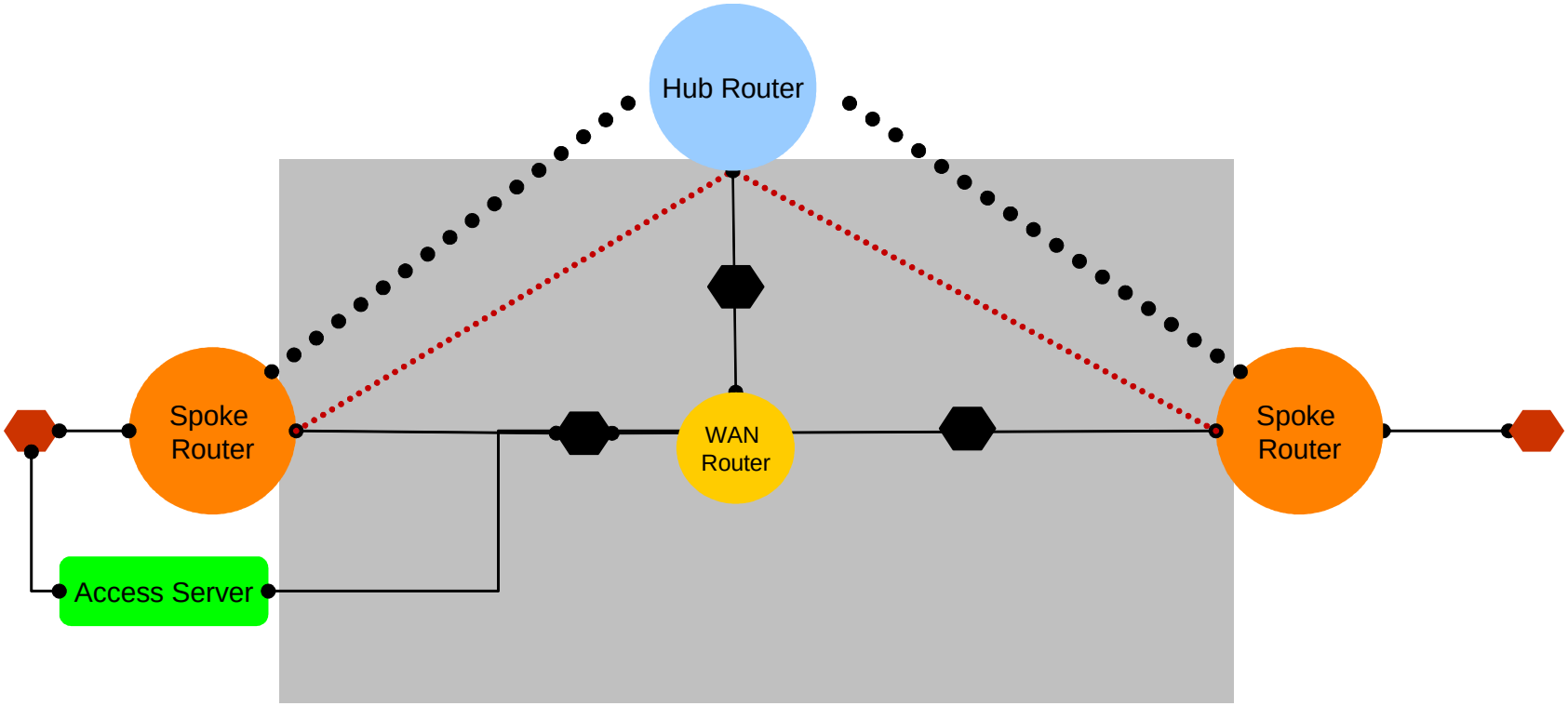
Requirement Strengthening



(PhysicalSpec ∧ GRERequirements ∧ SecureGRERequirements)

Alloy automatically places the IPSec tunnel between the correct physical interfaces to protect the GRE tunnel.

Component Addition

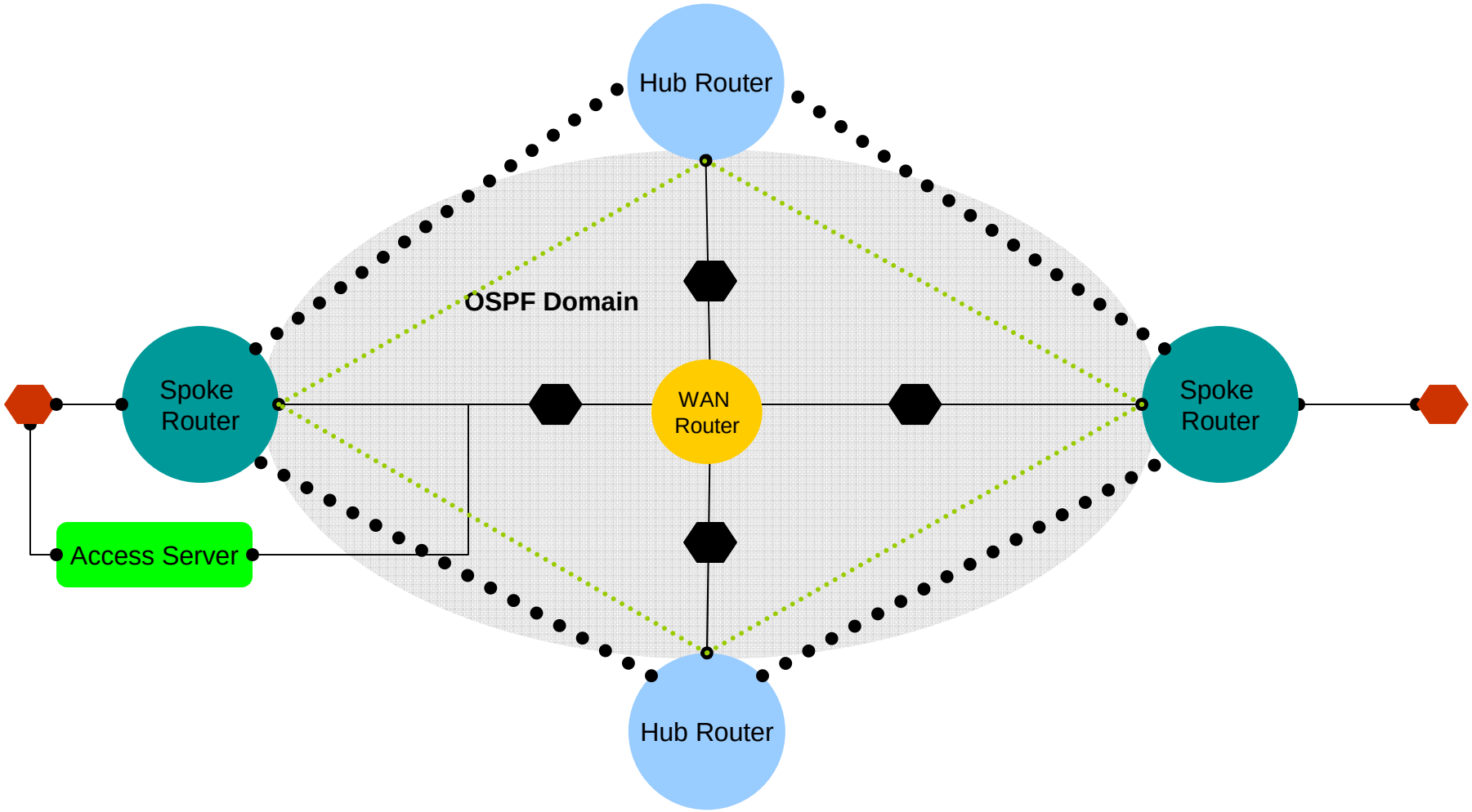


(PhysicalSpec ∧ GRERequirements ∧ SecureGRERequirements ∧ AccessServerRequirements)

New spoke router is physically connected just to the WAN router

GRE and IPSec tunnels are automatically set up between the new spoke router and hub router and physical interfaces and GRE tunnels are placed in the correct routing domains.

Verification: Discovering Cause of IP Packet Drop



Symptom: Cannot ping from one internal interface to another
Define BadReq = ip packet is blocked
Check if R1-R16 & BadReq is satisfiable
Answer: WAN router firewalls block ike/ipsec traffic

Writing Efficient Requirements: Scope Splitting

```
sig router {}
sig interface {chassis: router}
pred EmptyCond () {}
```

- When Alloy tries to find a model for EmptyCond in a scope of 50 routers and 50 interfaces it crashes(ed)!
- This is because the cross product of the set of all routers and chassis' has 50*50=2500 pairs.
- Each subset of this product is a value of the chassis relation. There are 2^2500 subsets!
- We can now try splitting the scope and redefining the specification:

```
sig hubRouter {}
sig spokeRouter {}
sig hubRouterInterface {chassis:hubRouter}
sig spokeRouterInterface {chassis:spokeRouter}
```

- Now, Alloy returns a model of EmptyCond for the scope consisting of 25 hubRouters, 25 spokeRouters, 25 hubRouterInterfaces and 25 spokeRouterInterfaces in seconds!
- Note that the scope still contains 50 routers and 50 interfaces. But there are now "only" 2^625 * 2^625 = 2^1250 possible values of chassis relation, or a factor of 2^1250 less.

Writing Efficient Requirements: Using Fewer Quantifiers

```
pred FirewallPolicyRequirements ()
{(all t:ipsecTunnel | some p1:firewallPolicy |
      p1.protectedInterface = t.local &&
      p1.prot = ike &&
      p1.action = permit) &&
(all t:ipsecTunnel | some p1:firewallPolicy |
      p1.protectedInterface =t.remote &&
      p1.prot = ike &&
      p1.action = permit) &&
(all t:ipsecTunnel | some p1:firewallPolicy |
      p1.protectedInterface = t.local &&
      p1.prot = esp &&
      p1.action = permit) &&
(all t:ipsecTunnel | some p1:firewallPolicy |
      p1.protectedInterface = t.remote &&
      p1.prot = esp &&
      p1.action = permit)
(no disj p1,p2:firewallPolicy |
      p1.protectedInterface=p2.protectedInterface
      &&
      p1.prot=p2.prot && !p1.action=p2.action)}
```

- Boolean formula contains 216,026 clauses and 73,4262 literals, and the entire process (compilation to solution) took 2 minutes and 59 seconds.

```
pred FirewallPolicyRequirements ()
{(all t:ipsecTunnel | some p1,p2,p3,p4:firewallPolicy
      |
      p1.protectedInterface = t.local &&
      p1.prot = ike &&
      p1.action = permit &&
      p2.protectedInterface = t.remote &&
      p2.prot = ike &&
      p2.action = permit &&
      p3.protectedInterface = t.local &&
      p3.prot = esp &&
      p3.action = permit &&
      p4.protectedInterface = t.remote &&
      p4.prot = esp &&
      p4.action = permit)&&
(no disj p1,p2:firewallPolicy |
      p1.protectedInterface=p2.protectedInterface
      &&
      p1.prot=p2.prot && !p1.action=p2.action)}
```

- Boolean formula contains 601,721 clauses and 2,035,140 literals and the entire process took 8 minutes and 19 seconds.

Limitations

- Alloy works well for complex logic and small scopes, not large scopes

- Need for tighter control over FOL→ Boolean compilation led to ConfigAssure:
  - Arithmetic QFF, a good intermediary between FOL and Boolean
  - Partial evaluation at application layer
  - Can handle much larger scopes

References

- Daniel Jackson. Software Abstractions. MIT Press, 2006

- Alloy: http://alloy.mit.edu/community/

- Kodkod. http://alloy.mit.edu/kodkod/

- Sanjai Narain. Network Configuration Management Via Model Finding. Proceedings of USENIX Large Installation System Administration (LISA) Conference, San Diego, CA, 2005. Full report.

Next Two Lectures

- Professor Boon Thau Loo on the use of Datalog for implementing routing protocols