CONMan: A Step Towards Network Manageability

Hitesh Ballani and Paul Francis
Cornell University

ACM SIGCOMM 2007

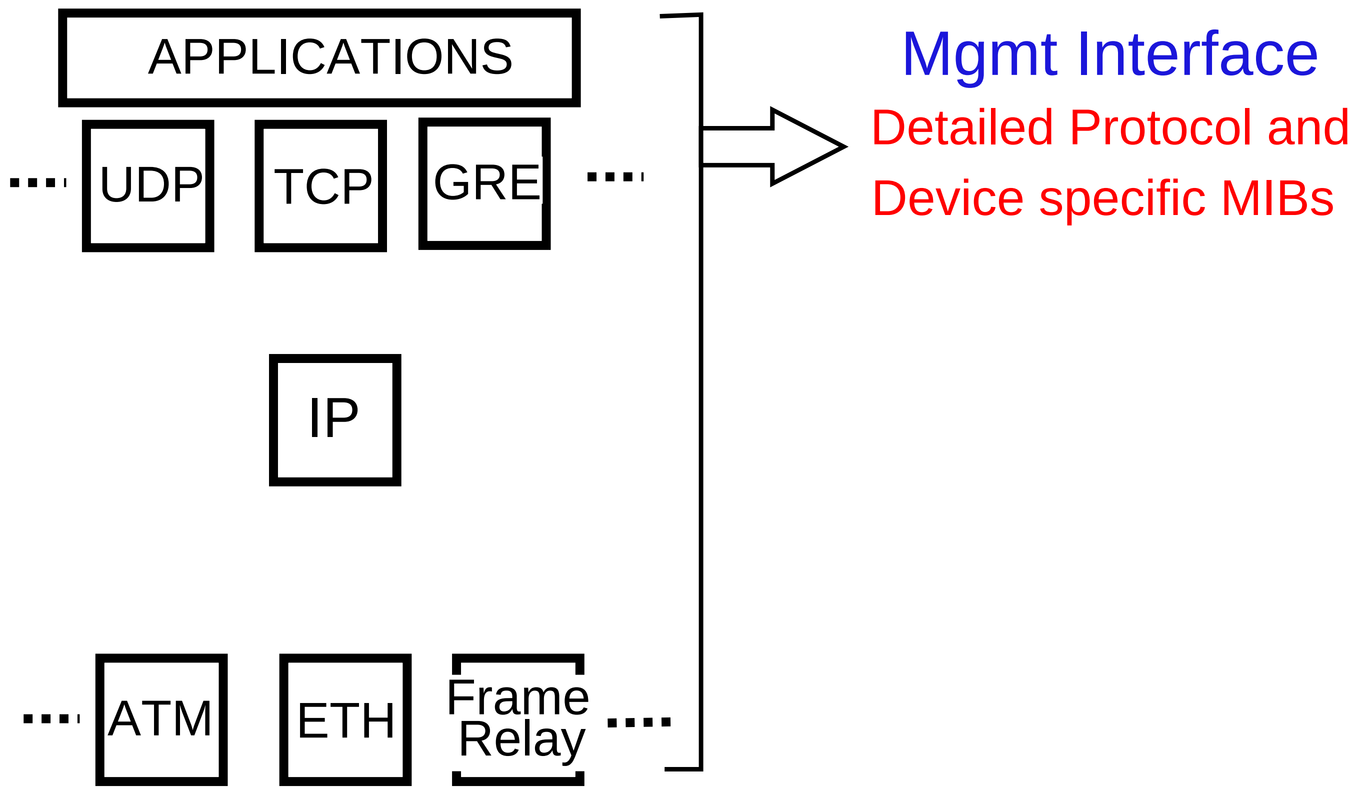
Network Management is a Mess

- Ad-Hoc
- Complex
- Error-Prone
- Expensive

Worsening situation as network complexity increases

- 80% of IT budget in enterprises used to maintain status quo [Kerravala'04]
- Configuration errors account for 62% of network downtime [Kerravala'04]

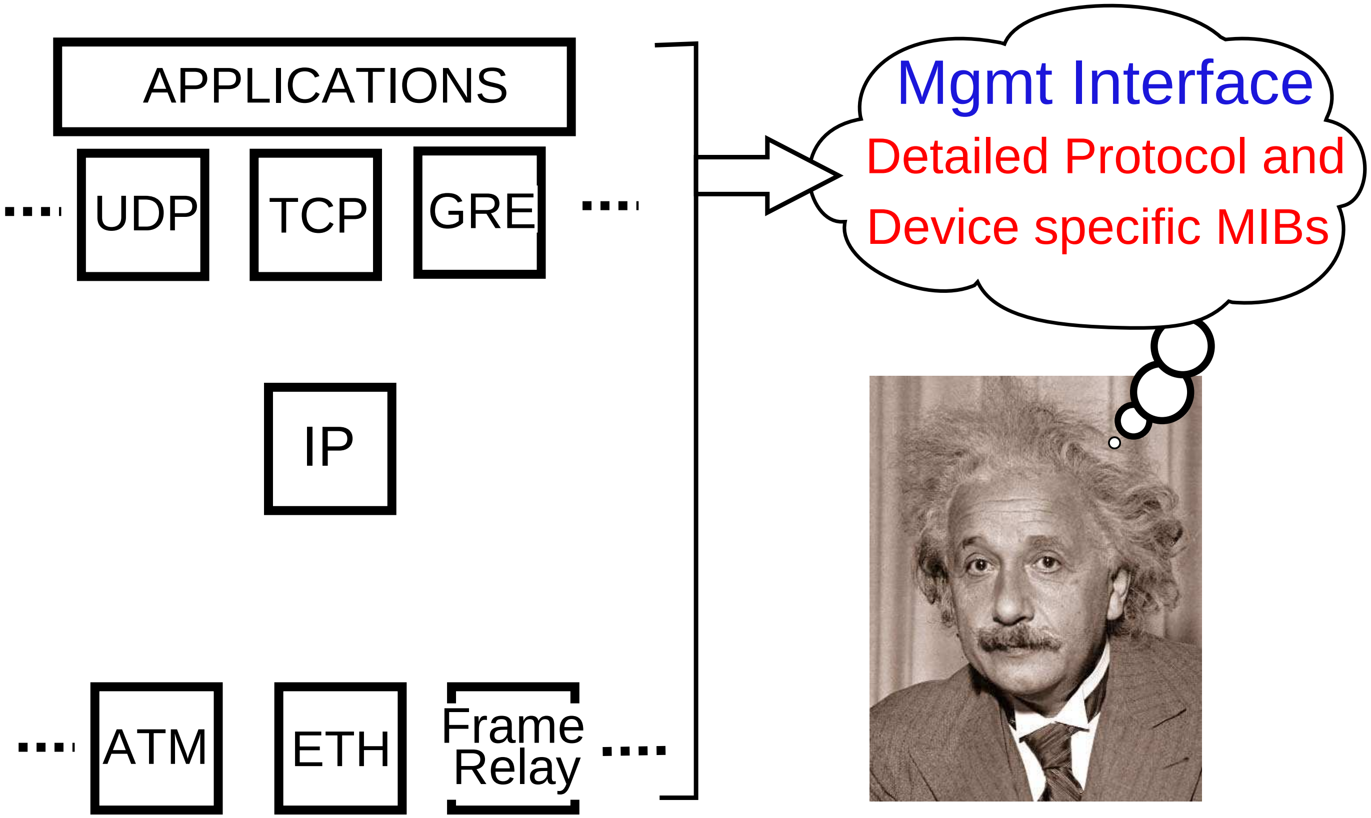
Protocols expose their gory details



MIB Depot : 6200 MIBs from 142 vendors and nearly a million MIB objects

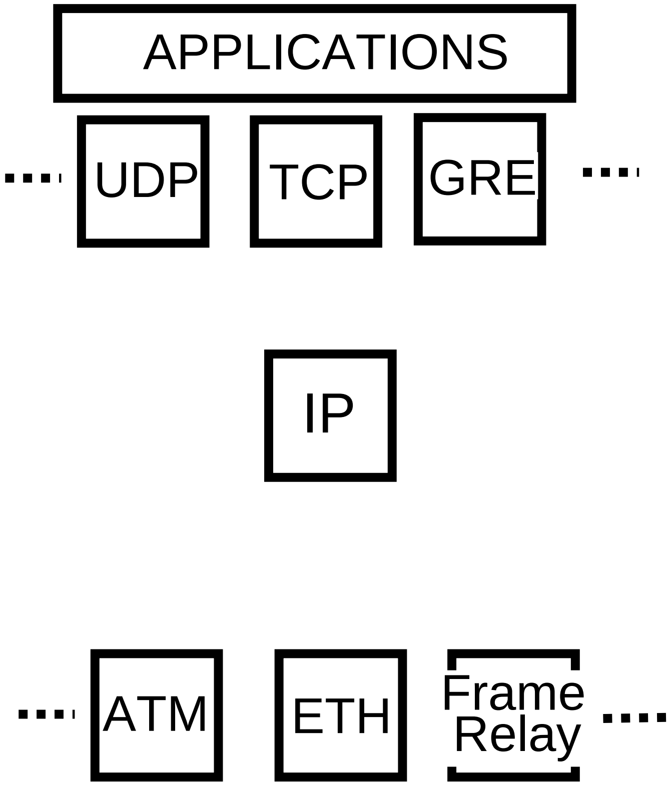
SNMPLink : More than a thousand management tools

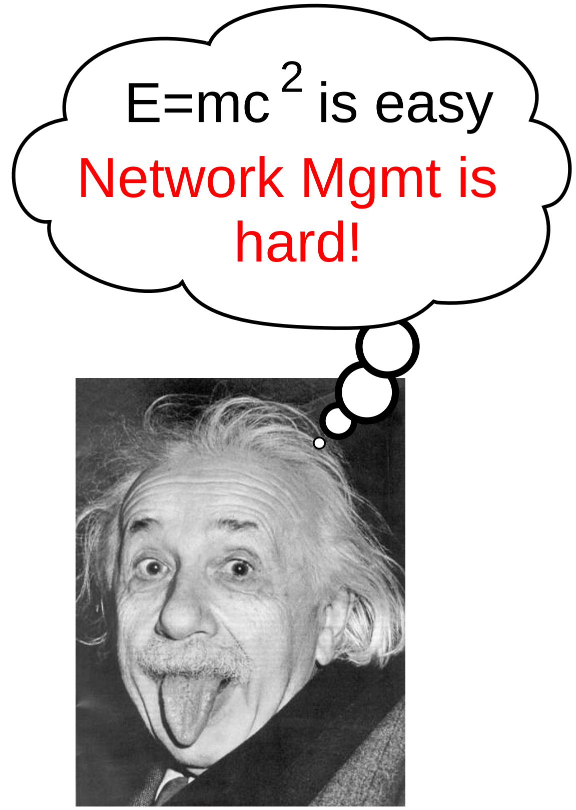
Protocols expose their gory details



Super-smart human managing the network

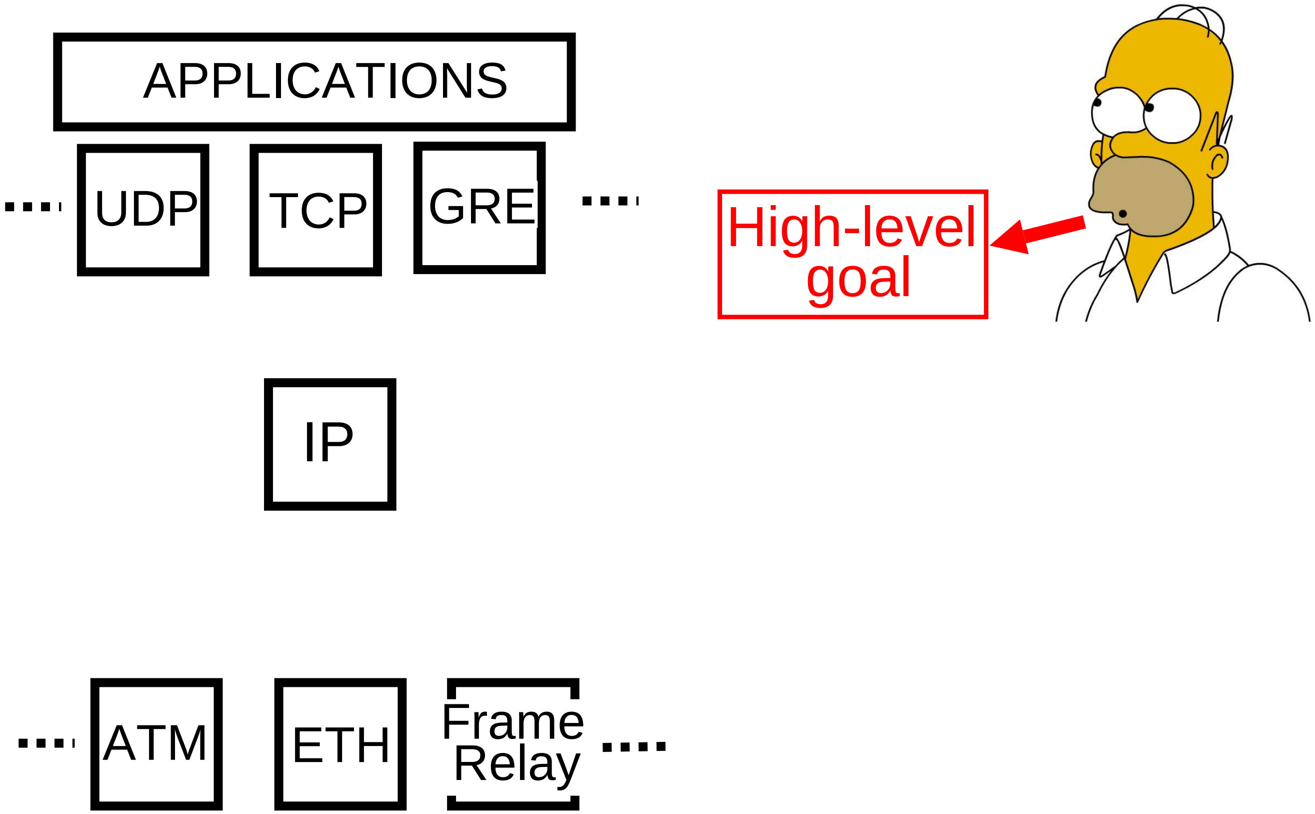
Protocols expose their gory details





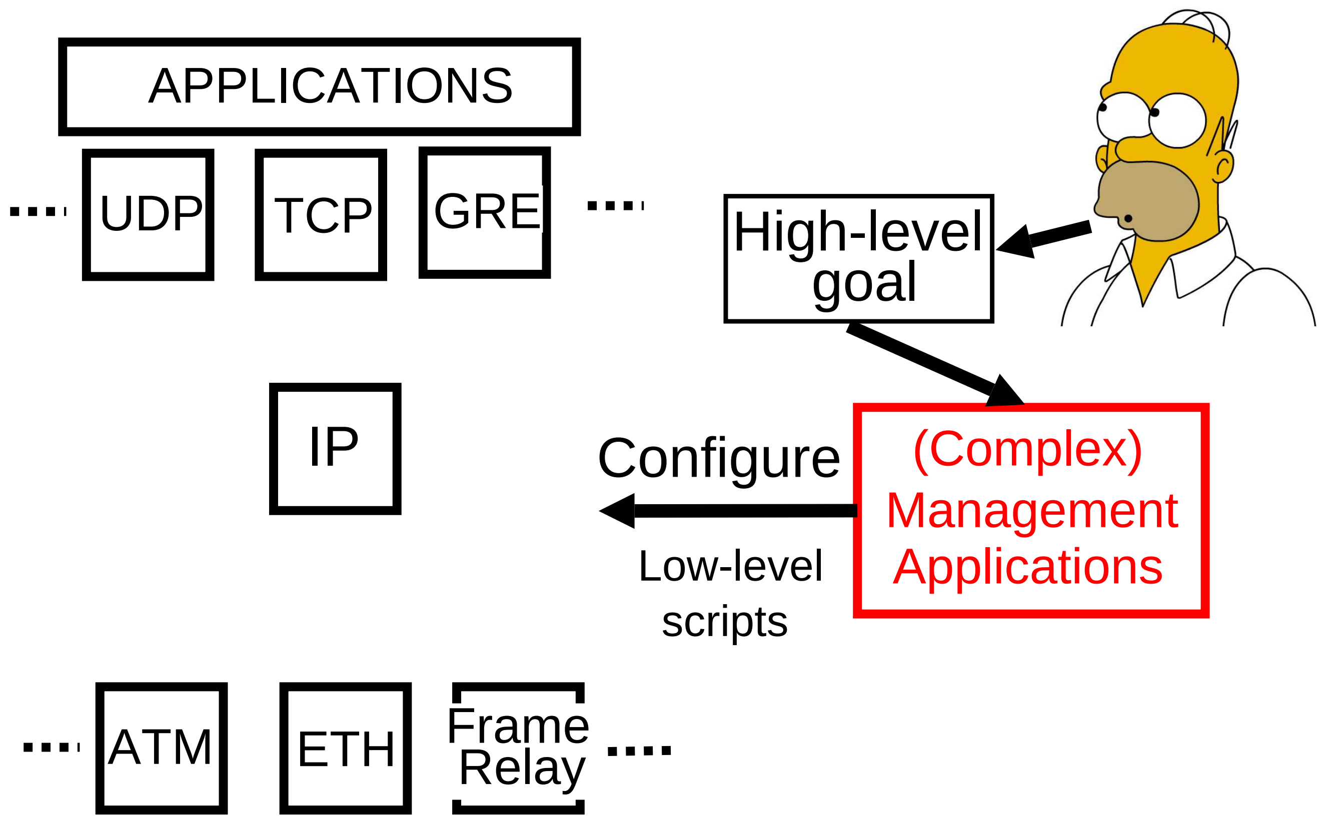
Super-smart human managing the network

Protocols expose their gory details



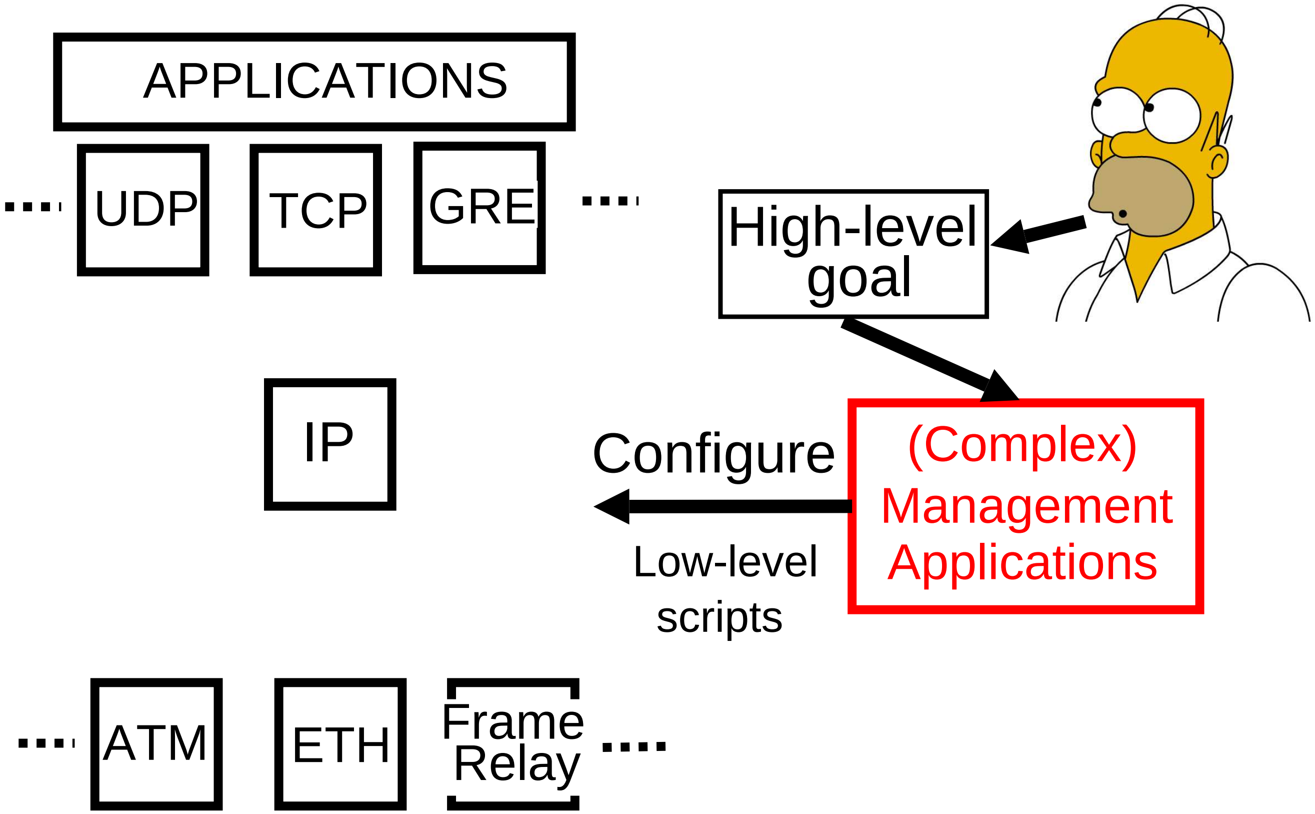
Human Manager only specifies high-level goal

Protocols expose their gory details



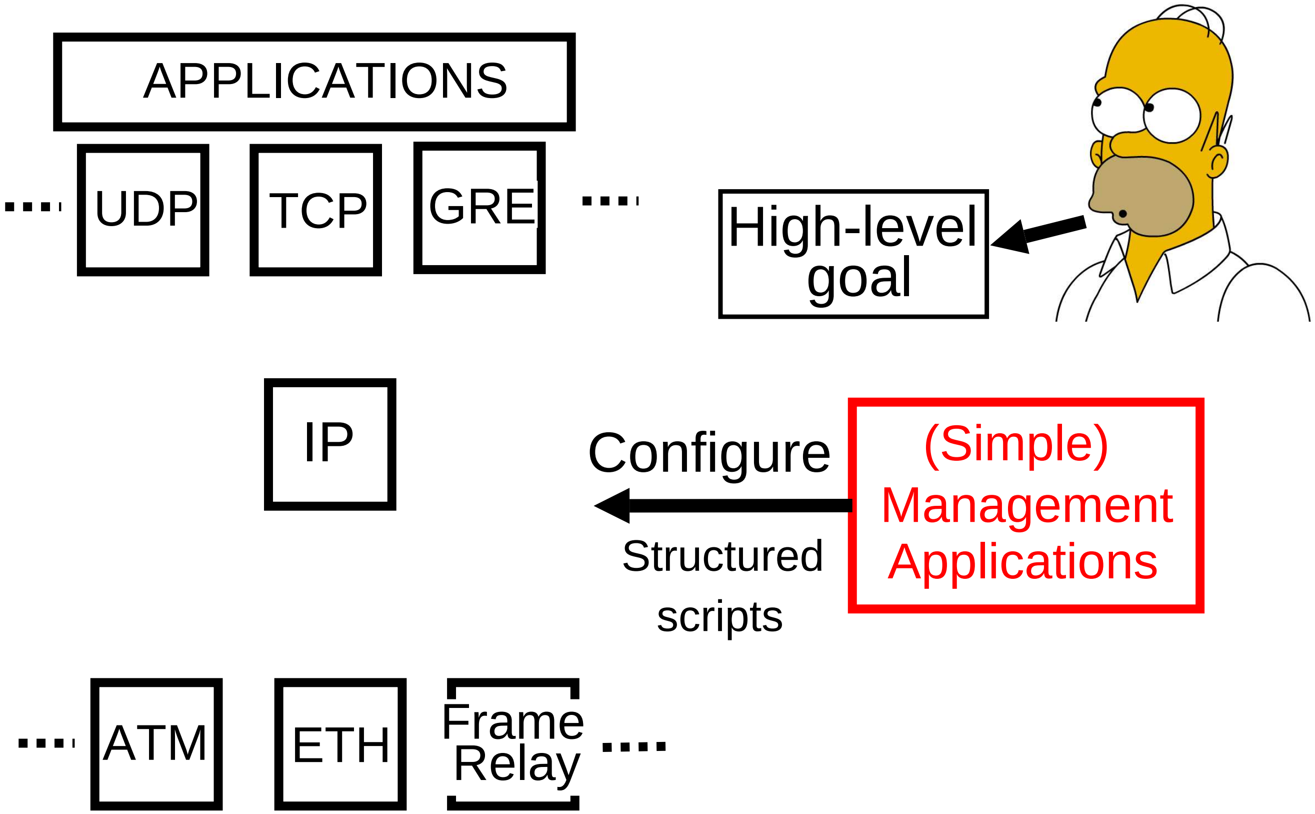
Management Application does the rest

Protocols expose their gory details



Deluge of complexity burdens the management application

Protocols expose their gory details



Refactor division of functionality between data and management plane

An Extreme Alternative

Confine the operational complexity of protocols to their implementation

An Extreme Alternative

Confine the operational complexity of protocols to their implementation

A more modest approach

The management interface of data-plane protocols should contain as little protocol-specific information as possible

Complexity Oblivious Network Management (CONMan)

A network management architecture

- (Little or) No protocol-specific information in the management interfaces of protocols
- Reduces burden on the management plane and hence, allows for simpler management

Focus on

- Network configuration tasks
- Management of data-plane protocols

Talk Outline

- Introduction
- CONMan Overview
- Module Abstraction
- CONMan primitives
- Implementation
- Conclusions and Future Work

CONMan Overview

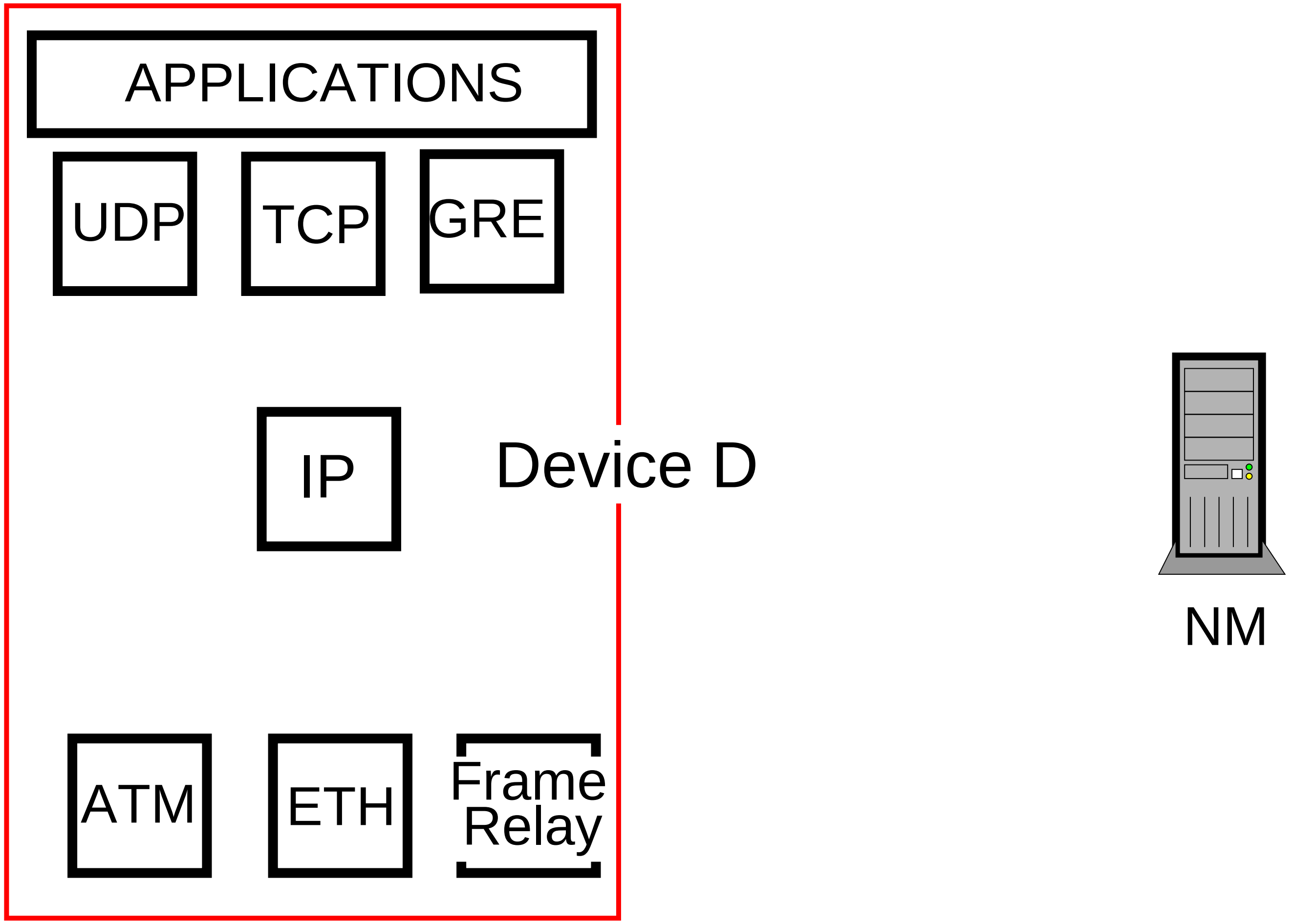
Devices with unique identifiers (*device-id*)

- Routers
- Switches
- Hosts
- . . .

Network Manager (NM)

- Software entity residing on one of the network devices
- Manages some or all of them
- One or more NMs in each network

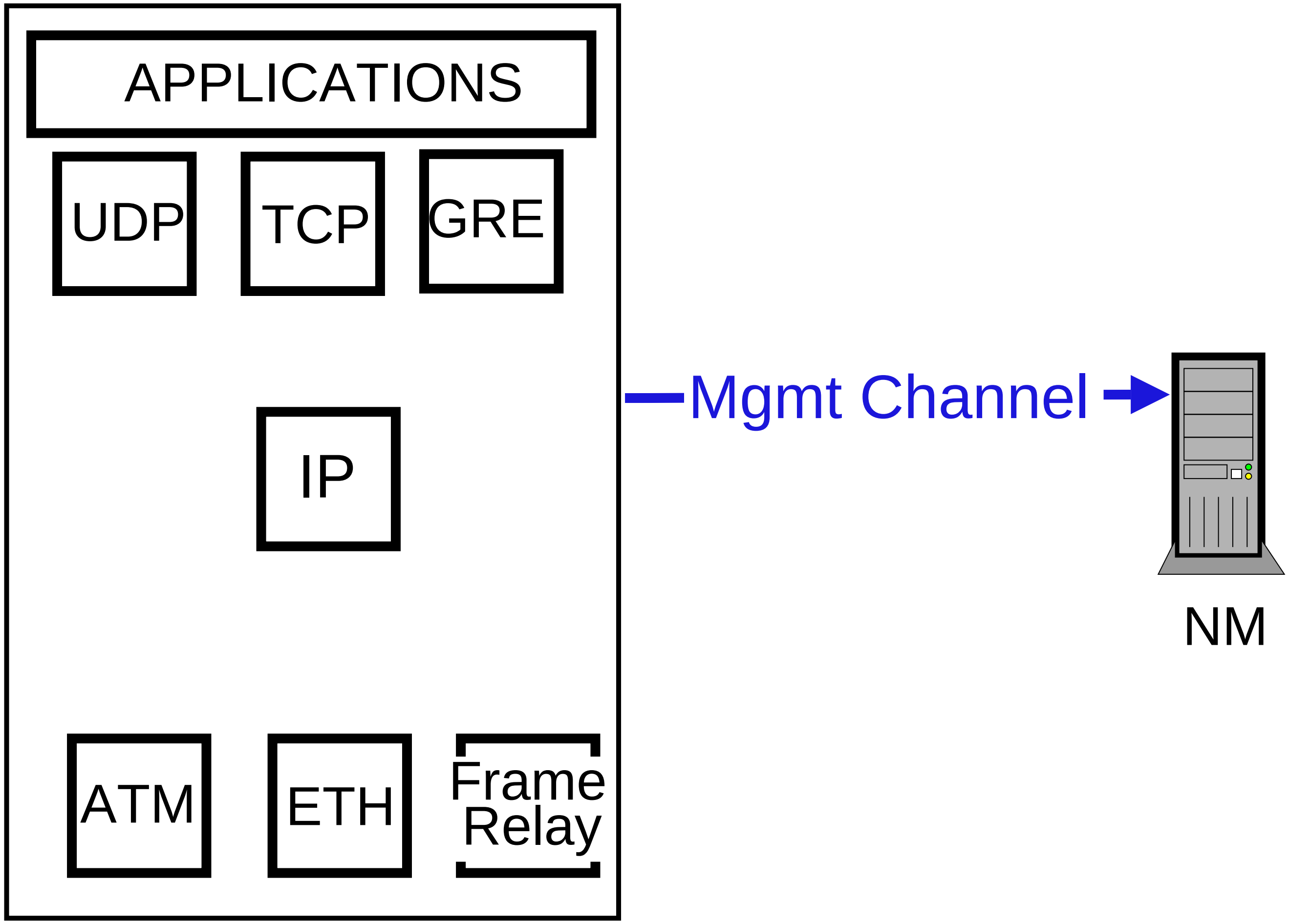
CONMan Overview



Each module has an identifier (*module-id*)

Module-id for IP module = $i \Rightarrow$ <IP,D,i>

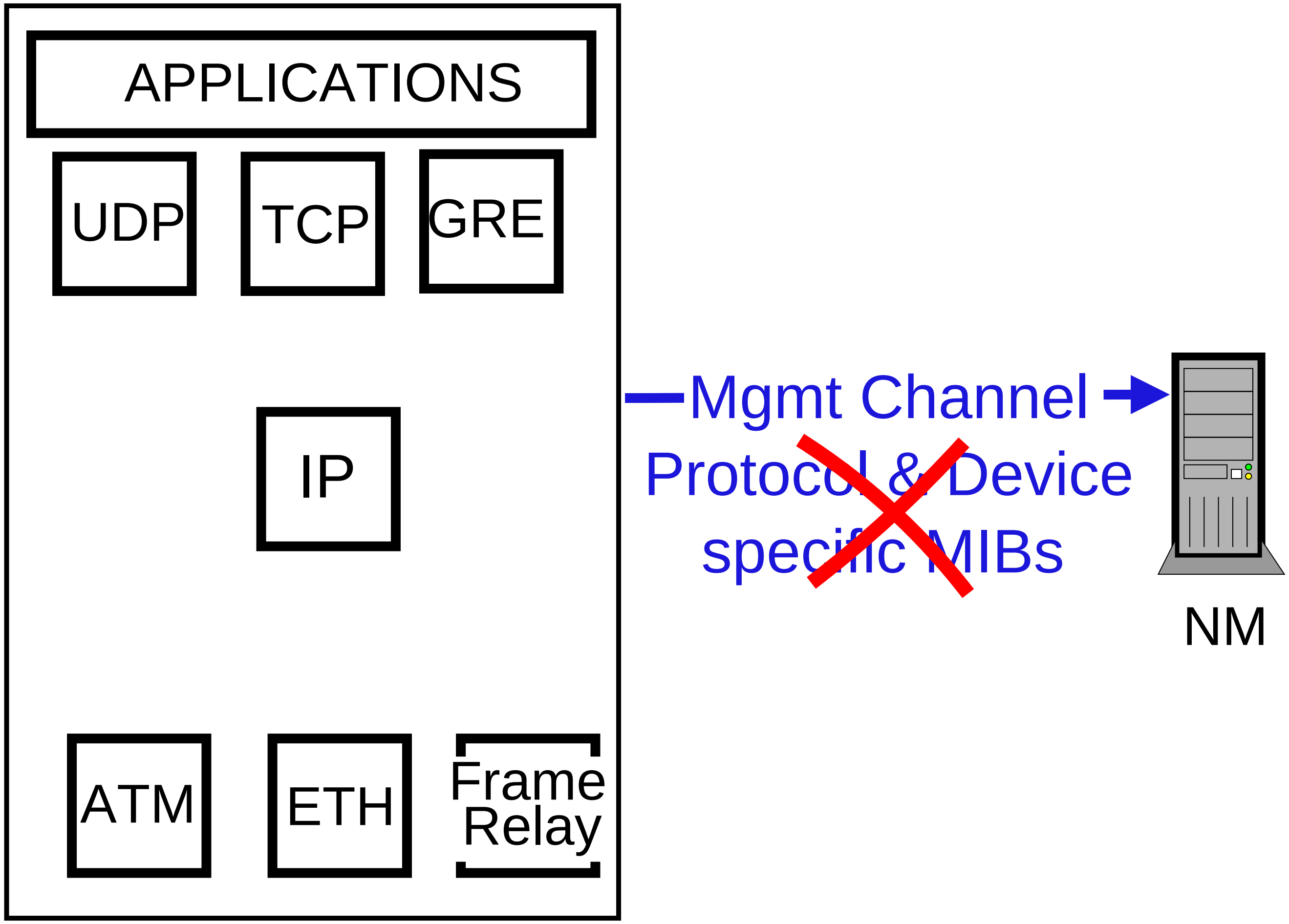
CONMan Overview



Self-bootstrapping management channel

Allows bidirectional communication between the NM and network devices [4D, Greenberg et. al. '05]

Abstract away the details



Protocols should not expose their gory details

What do the protocols expose?

Abstract away the details

Network configuration

- Provide paths between specific applications
- Ensuring that selected applications cannot use these paths

Basic characteristics of data-plane protocols

- Connect to other protocols
- Switching of packets
- Filtering of packets
- Queueing packets
- Dependence on external state

Abstract away the details

Network configuration

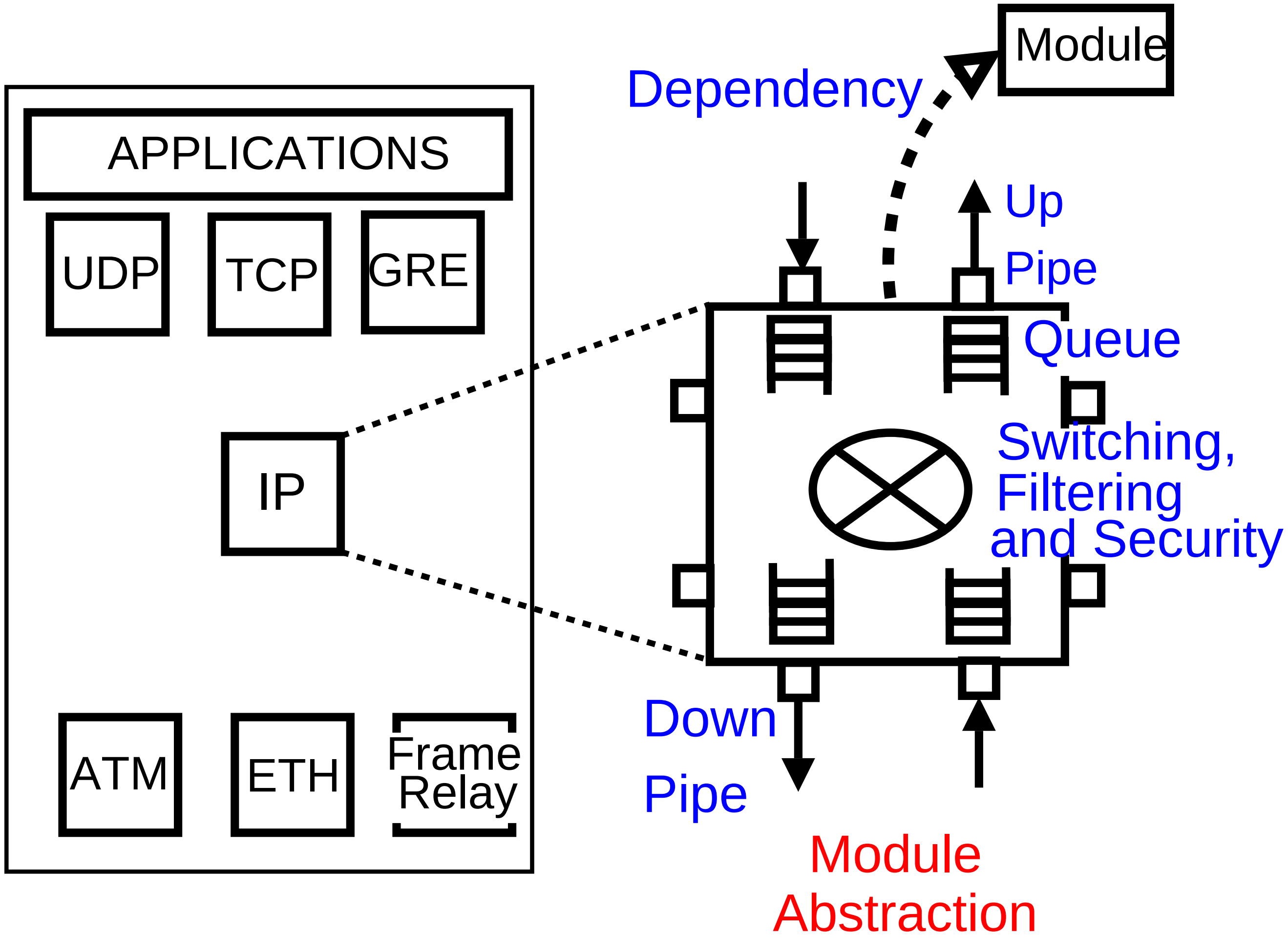
- Provide paths between specific applications
- Ensuring that selected applications cannot use these paths

Basic characteristics of data-plane protocols

- Connect to other protocols
- Switching of packets
- Filtering of packets
- Queueing packets
- Dependence on external state

These basic characteristics should serve as a narrow waist for the Internet's management plane

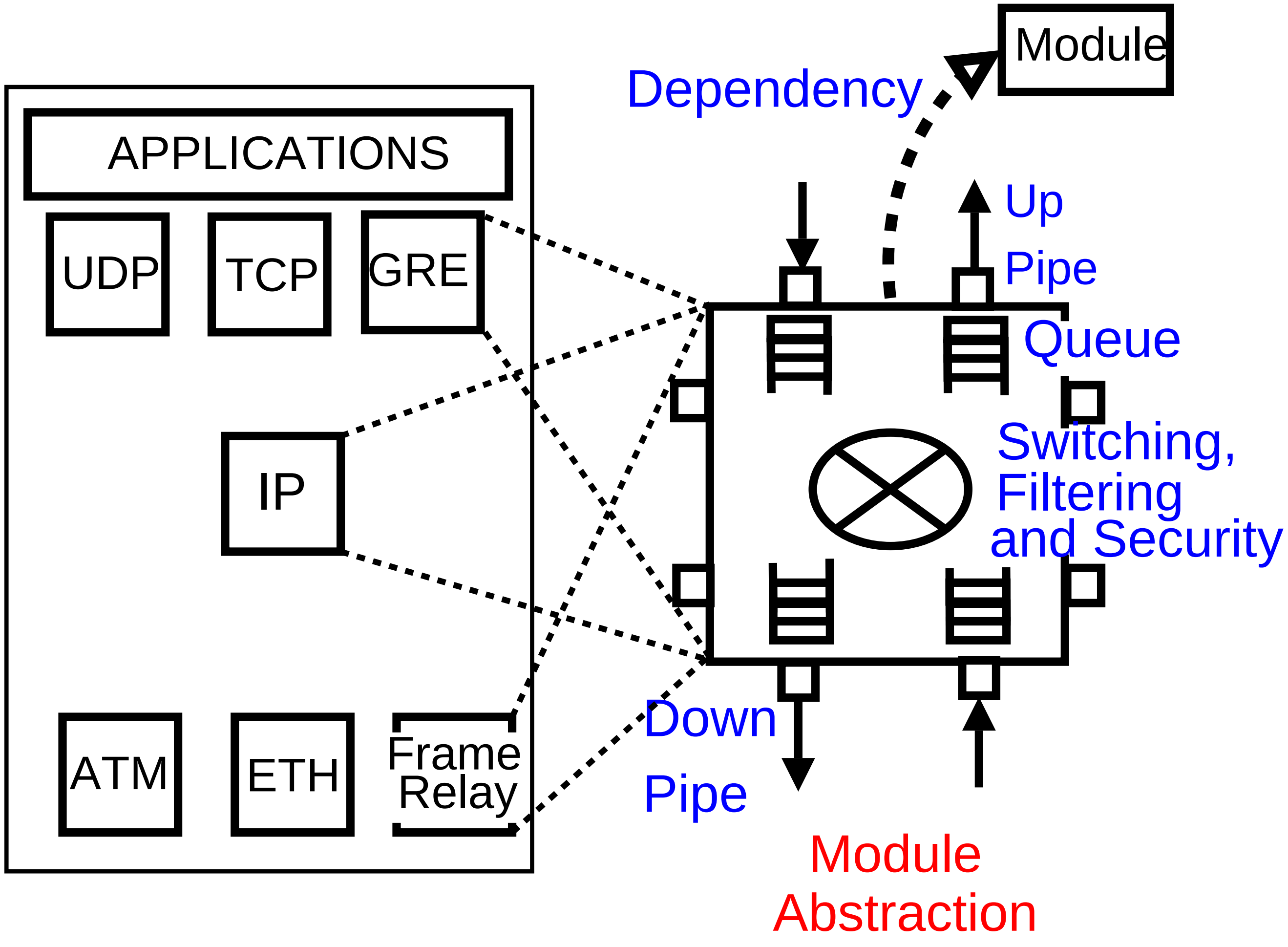
Abstract away the details



Module Abstraction: Mgmt Interface of a module

Models the protocol's potential and dependencies

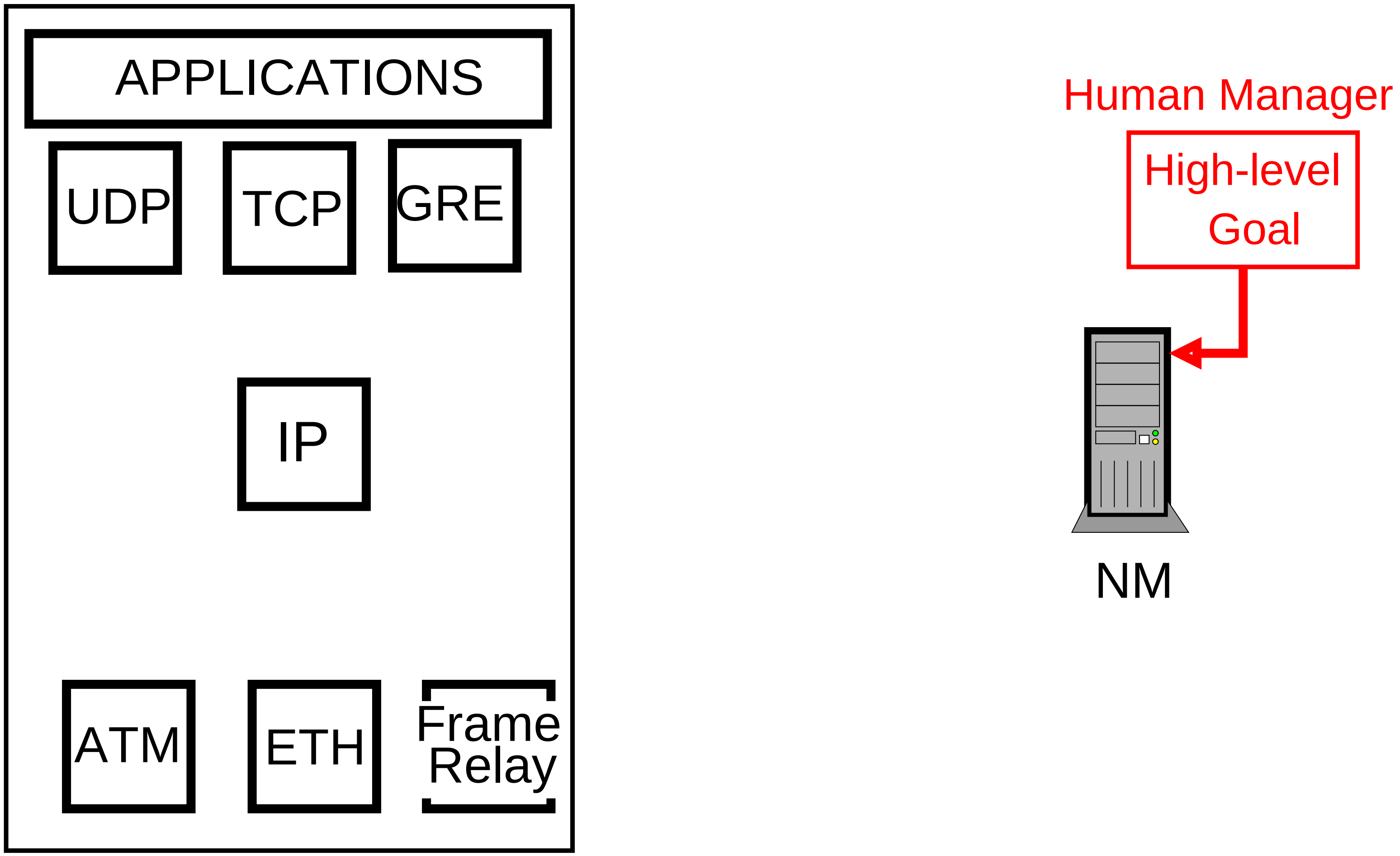
Abstract away the details



Module Abstraction: Mgmt Interface of a module

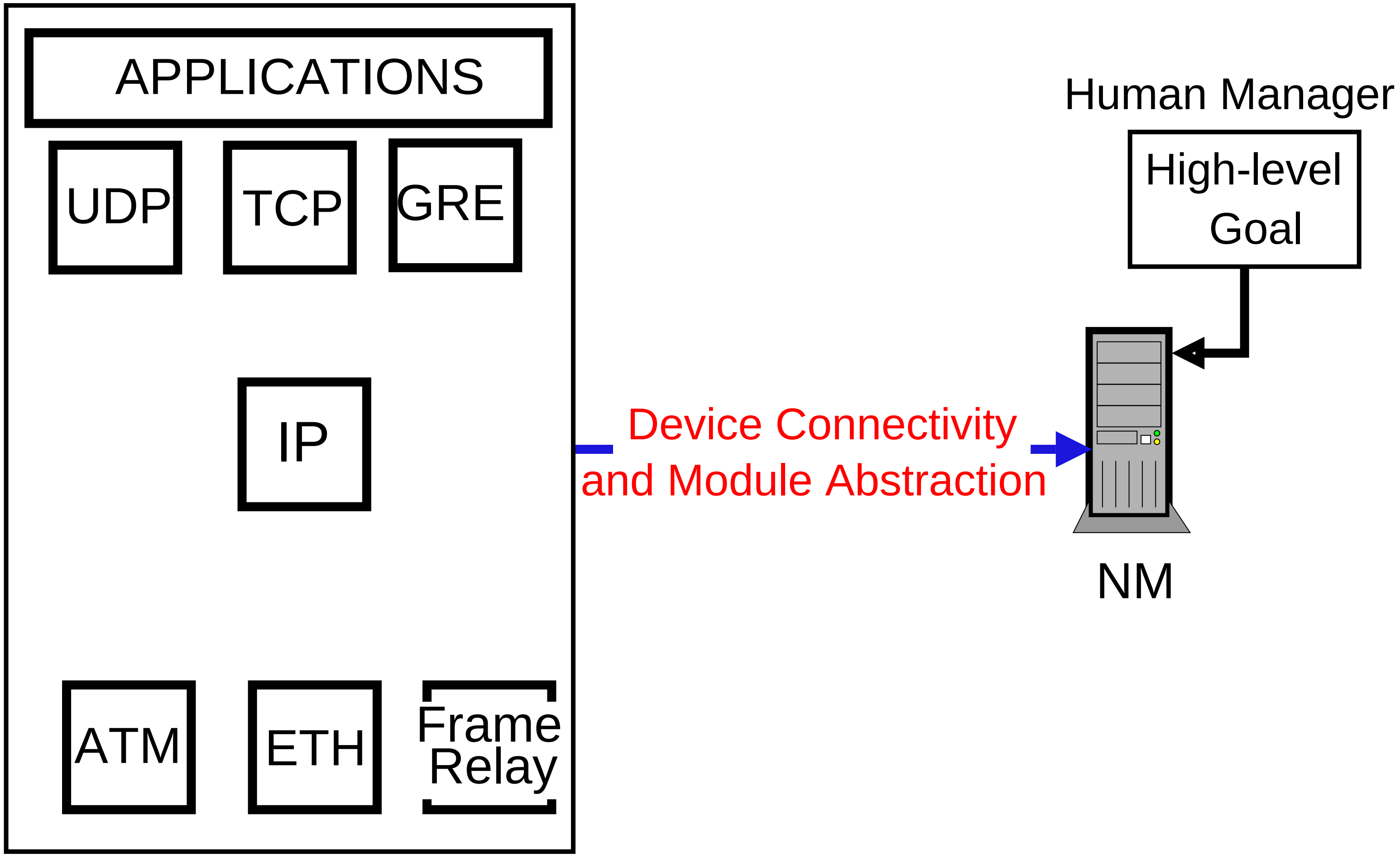
Applies to (almost) all data-plane protocols

CONMan: The big picture



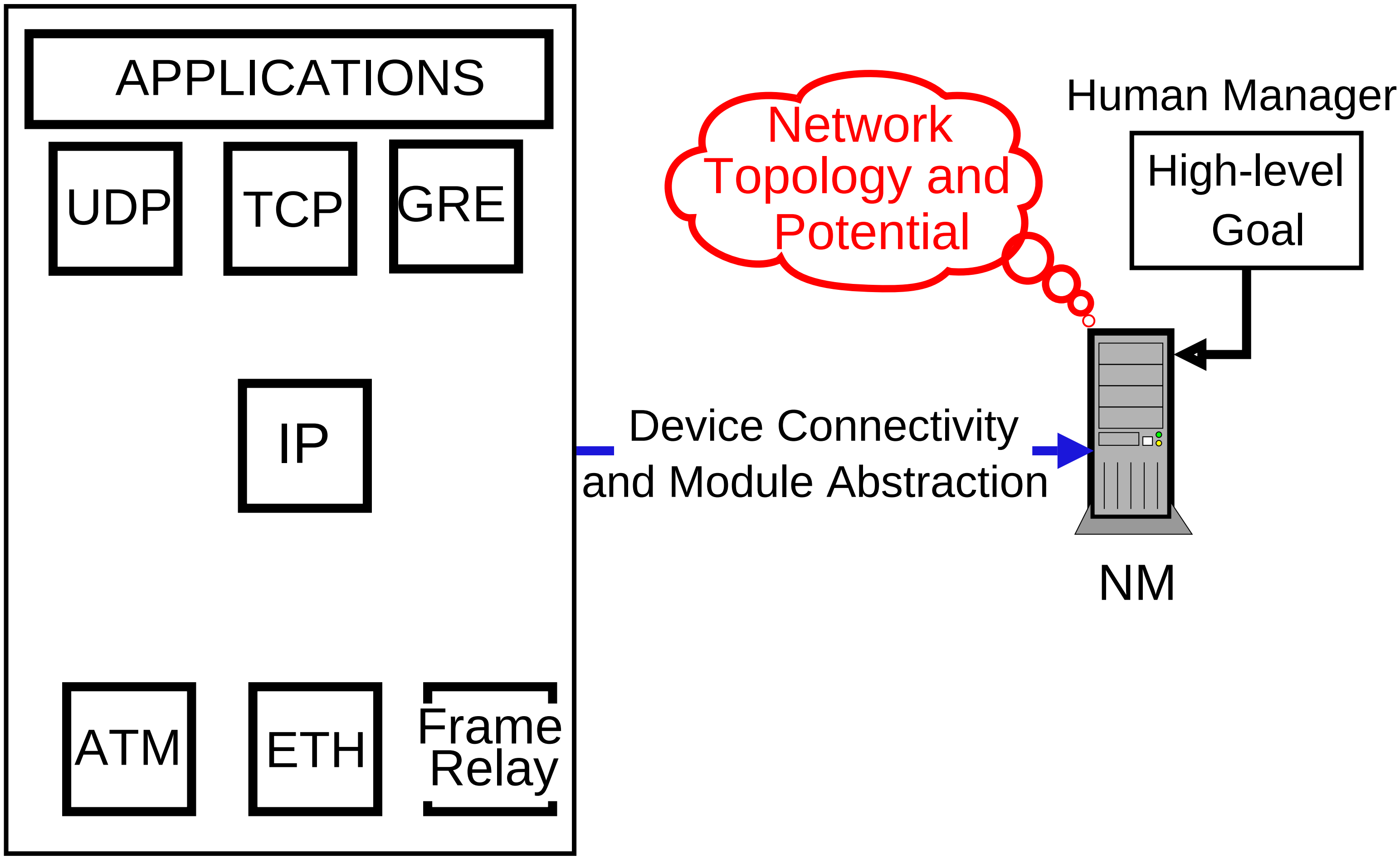
Human managers specify high-level goals

CONMan: The big picture



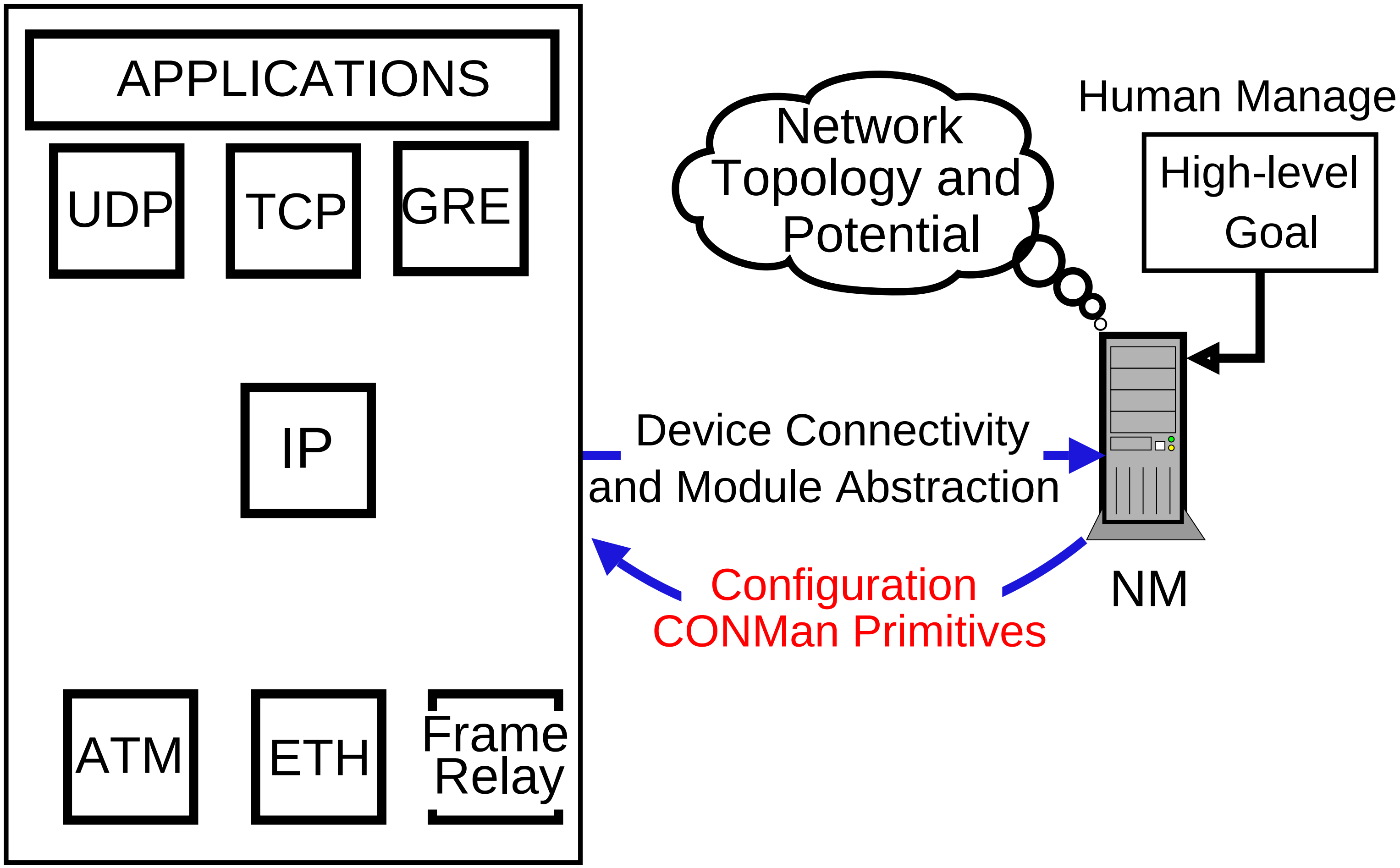
Each device's connectivity and the abstraction for its modules are sent to the NM

CONMan: The big picture



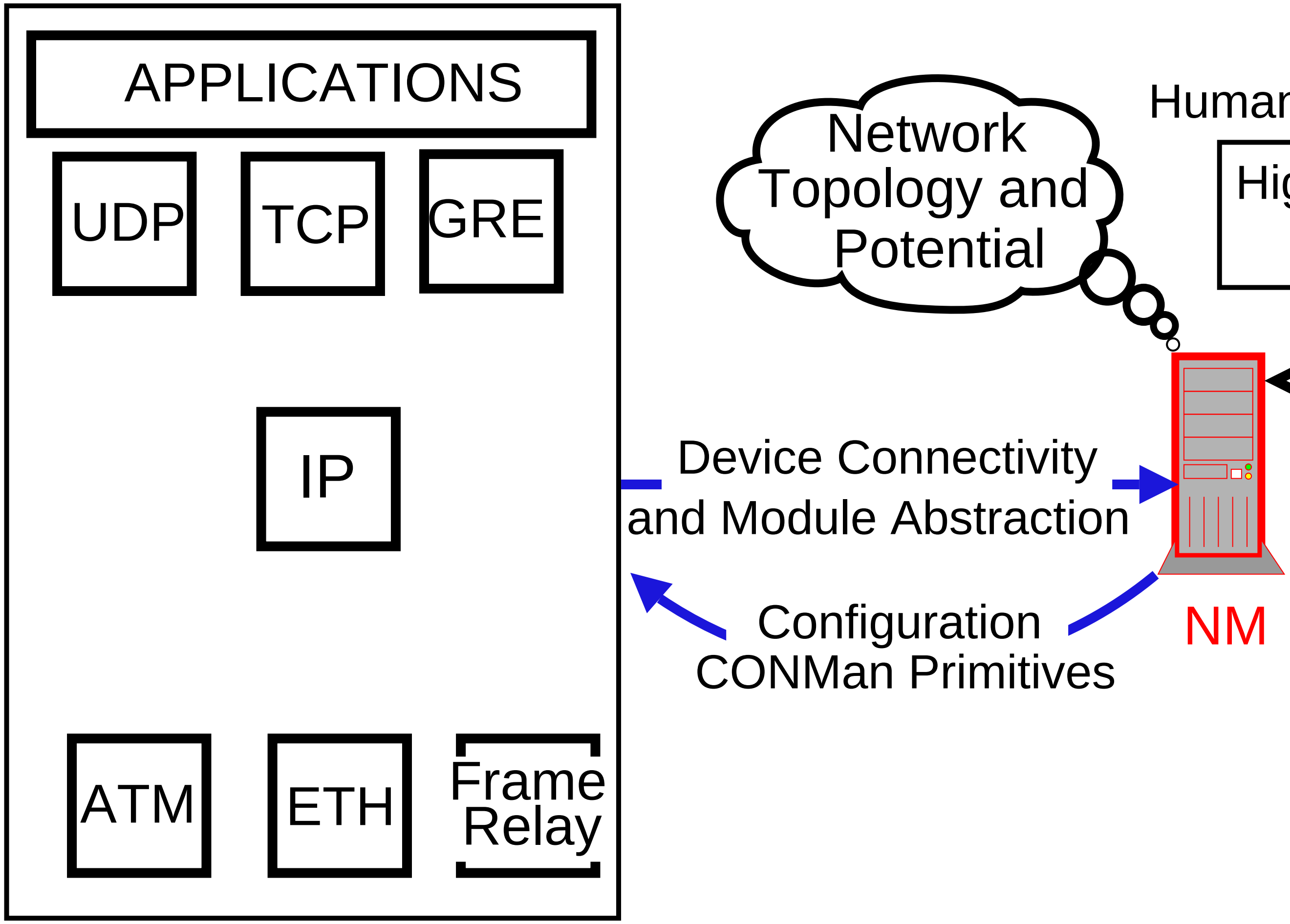
NM knows the network topology and the network potential

CONMan: The big picture



NM uses CONMan primitives to manipulate abstraction elements and configure network devices

CONMan: The big picture



The amount of complexity that the NM needs to handle is reduced!

CONMan Abstraction and Primitives

Abstraction Components

- Name
- Up-Down Pipes
- Physical Pipes
- Switch
- Filter
- Perf. Reporting
- Perf. Trade-off
- Security

CONMan primitives

- *show*
- *create*
- *delete*
- *conveyMessage*
- *listFieldsAnd-*
  -Values

CONMan Abstraction and Primitives

Abstraction Components

- Name
- Up-Down Pipes
- Physical Pipes
- Switch
- Filter
- Perf. Reporting
- Perf. Trade-off
- Security

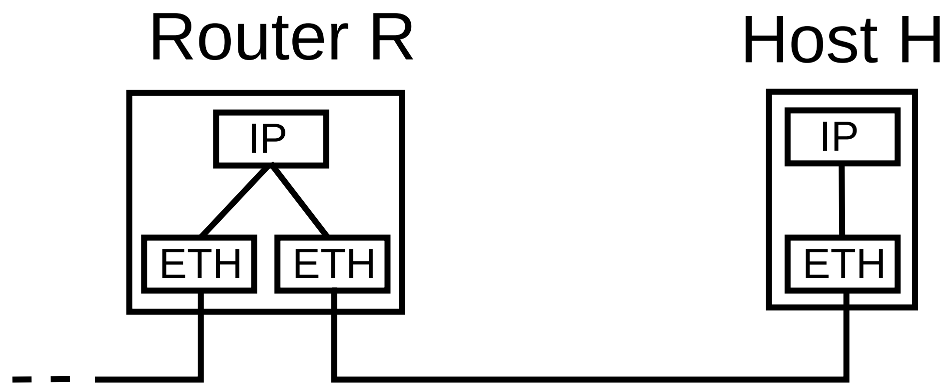
CONMan primitives

- *show*
- *create*
- *delete*
- *conveyMessage*
- *listFieldsAnd-*
  -Values

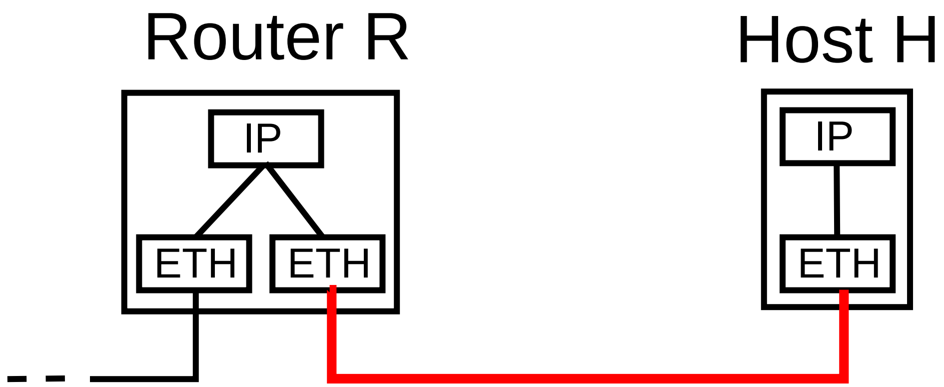
Talk Outline

- Introduction
- CONMan Overview
- Module Abstraction
- CONMan primitives
- Implementation
- Conclusions and Future Work

Pipes



Pipes

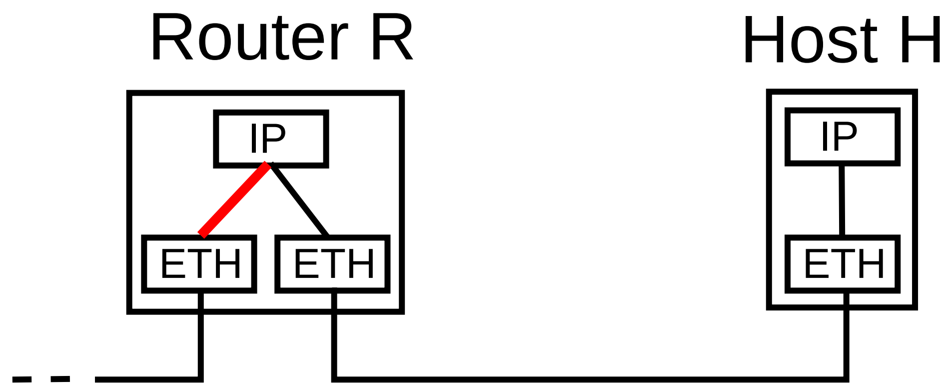


Physical Pipes

Model actual network links

Are discovered and enabled by the NM

Pipes



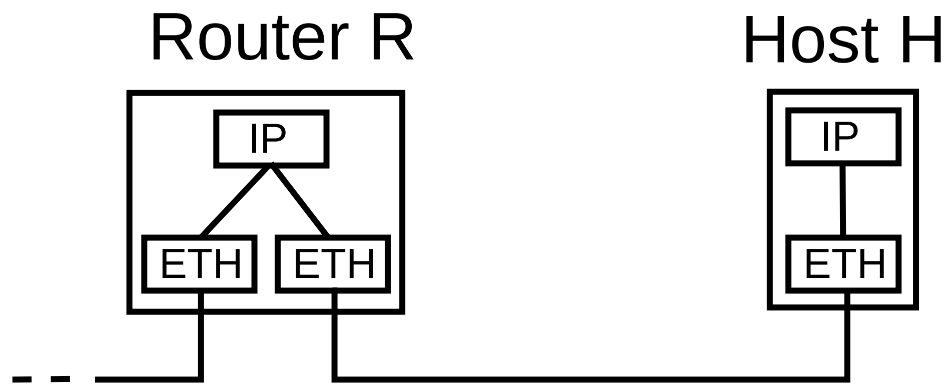
Up-Down Pipes

Between modules in the same device

Can be *created/deleted* by the NM

Pipe in figure is Down pipe for IP and Up pipe for ETH

Pipes



Connectable Modules

- Captures the possible protocol plumbing
- Eg. Connectable Modules for an up pipe of an ETH module: {IP, MPLS}

Pipes

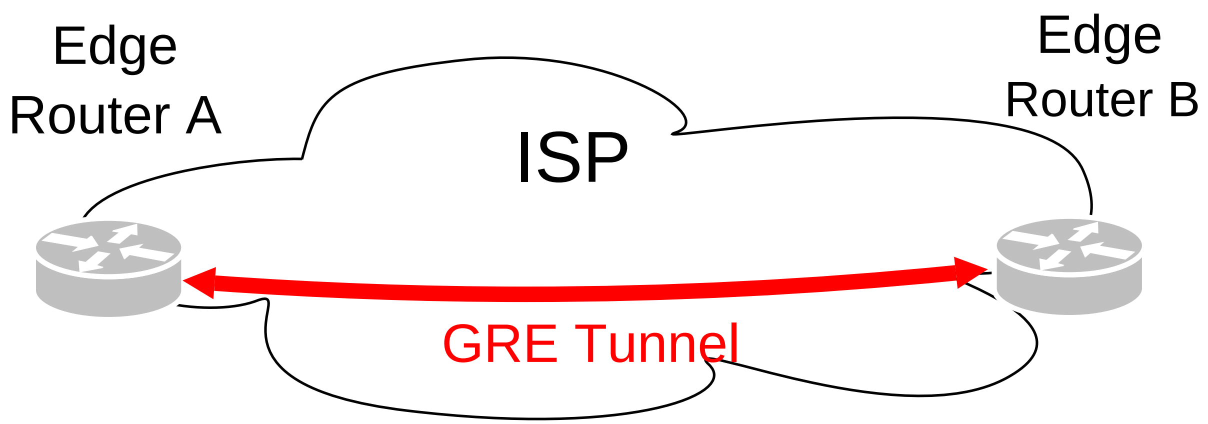
Peer modules

- Up-Down pipes associated with peer modules
- Peer modules coordinate low-level details

Pipes

Peer modules

- Up-Down pipes associated with peer modules
- Peer modules coordinate low-level details

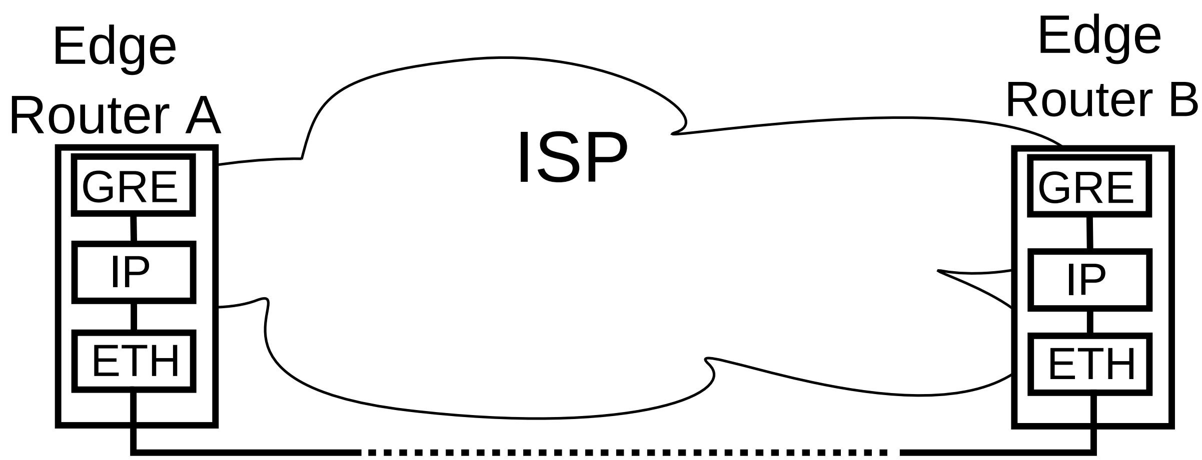


A GRE tunnel between edge routers A and B

Pipes

Peer modules

- Up-Down pipes associated with peer modules
- Peer modules coordinate low-level details



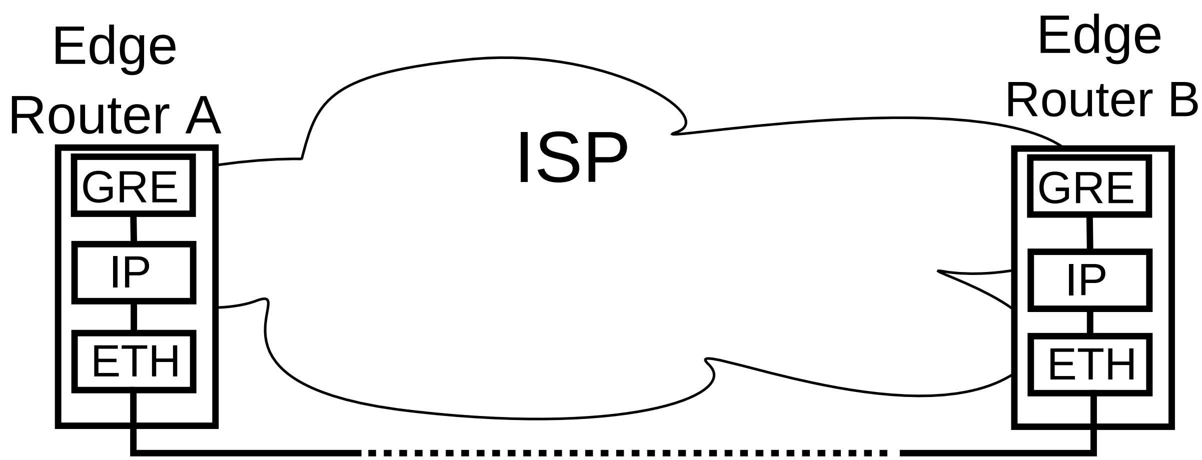
NM builds the path by creating the requisite pipes

NM can invoke *create* and *delete* primitives at the devices

Pipes

Peer modules

- Up-Down pipes associated with peer modules
- Peer modules coordinate low-level details



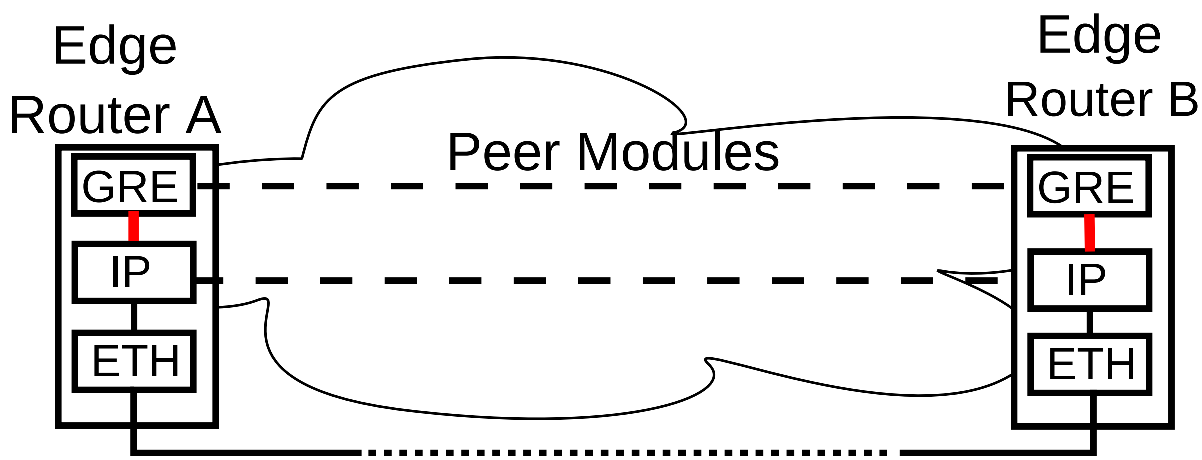
What about the low-level details?

```
ip tunnel add name gre-A-B mode gre remote 204.9.169.1
local 204.9.168.1 ikey 1001 okey 2001 icsum ocsum iseq
oseq
```

Pipes

Peer modules

- Up-Down pipes associated with peer modules
- Peer modules coordinate low-level details



Peer modules can coordinate low-level values

Eg. Peer GRE modules can exchange key values (1001, 2001)

Hiding Complexity

NM operates in terms of abstract components

- Eg. Filter rules specify abstraction components

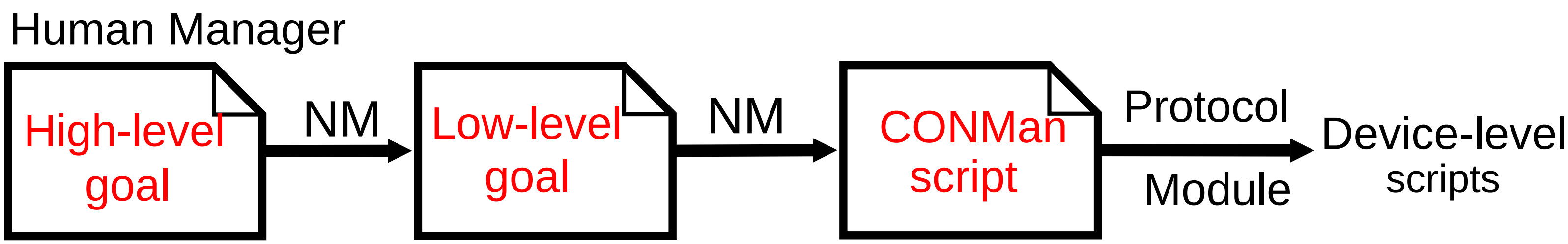
Exceptions

- IP address assignment
- Filtering based on regular expressions in HTML
- Broadcast suppression on switch ports
- ...

Talk Outline

- Introduction
- CONMan Overview
- Module Abstraction
- CONMan primitives
- Implementation
- Conclusions and Future Work

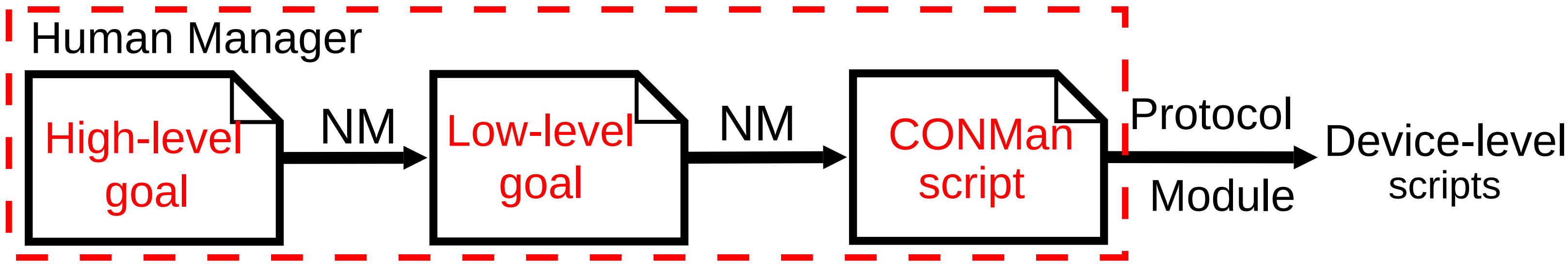
CONMan Workflow



Implementation

- A **Network Manager (NM)** that understands the CONMan abstraction and implements the CONMan primitives
- **Protocol Modules**: GRE, MPLS, IP, ETH

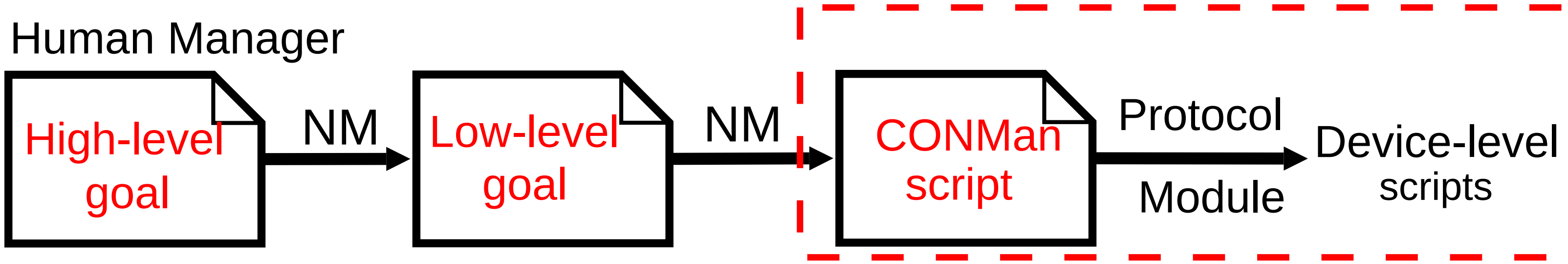
CONMan Workflow



Implementation

- A **Network Manager (NM)** that understands the CONMan abstraction and implements the CONMan primitives
- **Protocol Modules**: GRE, MPLS, IP, ETH

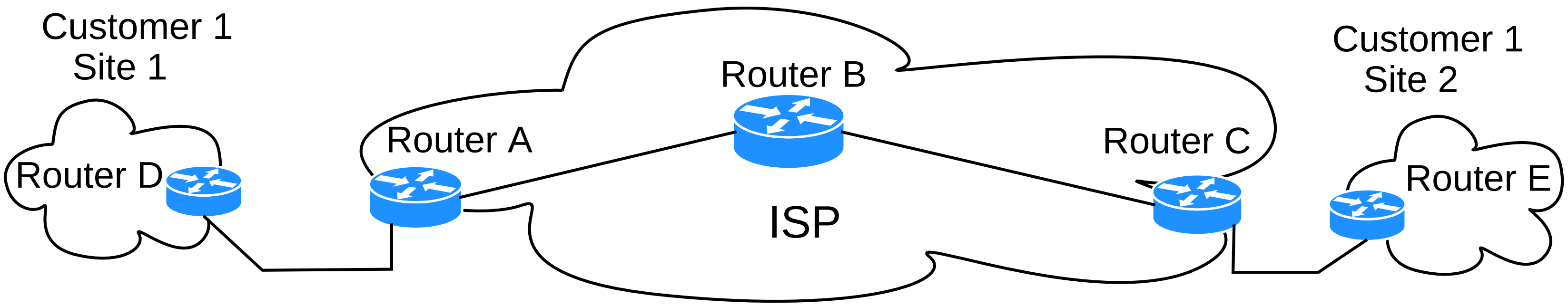
CONMan Workflow



Implementation

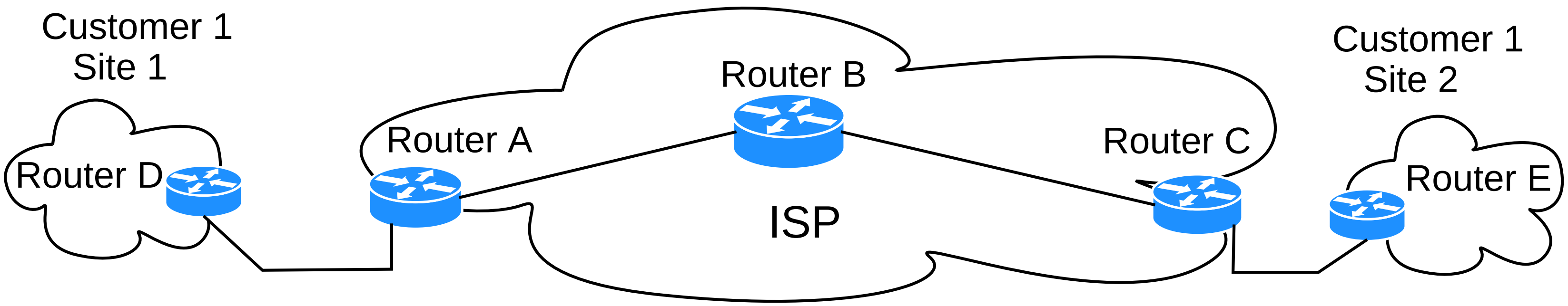
- A **Network Manager (NM)** that understands the CONMan abstraction and implements the CONMan primitives
- **Protocol Modules**: GRE, MPLS, IP, ETH

Virtual Private Networks

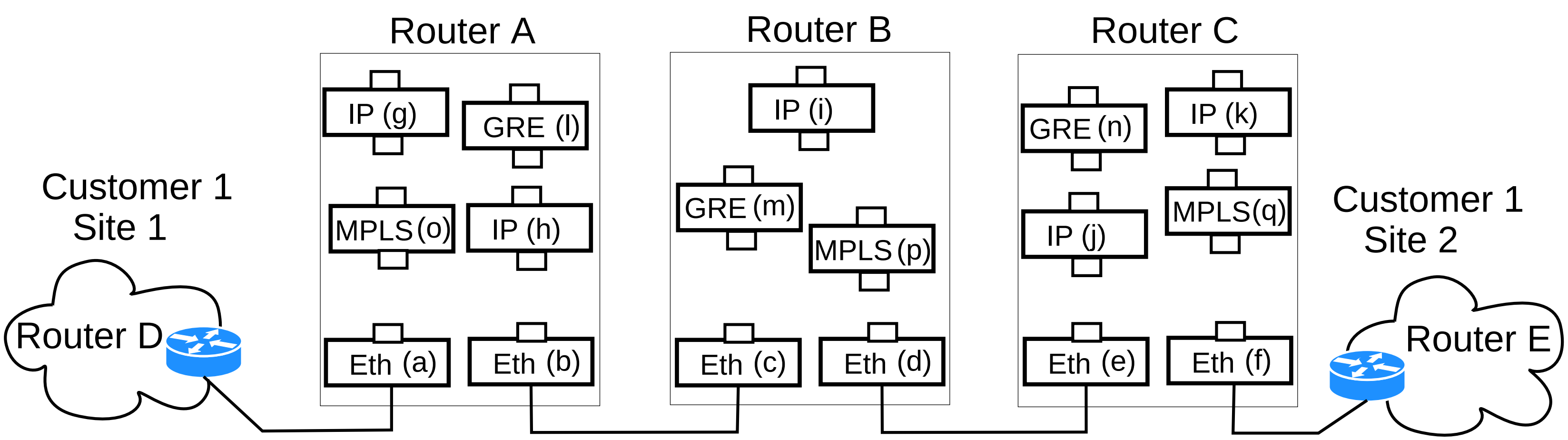


Configure connectivity between sites S1 and S2 of customer C1

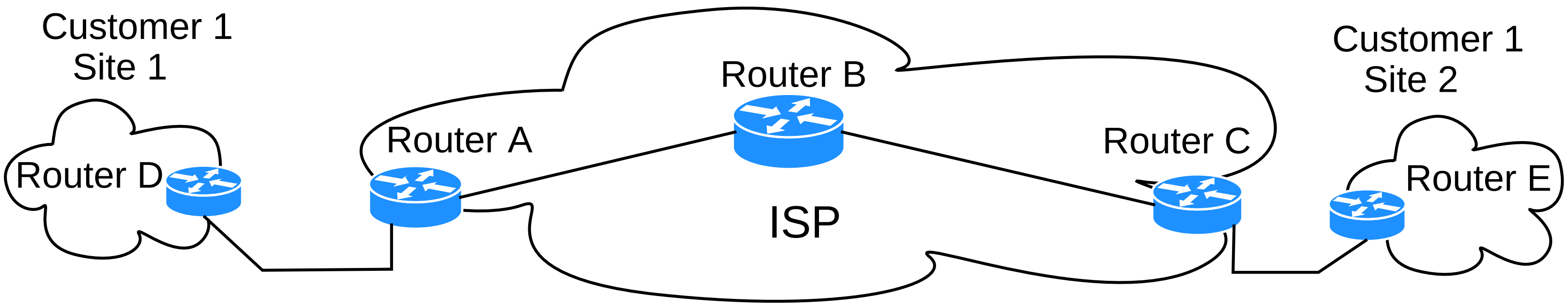
Virtual Private Networks



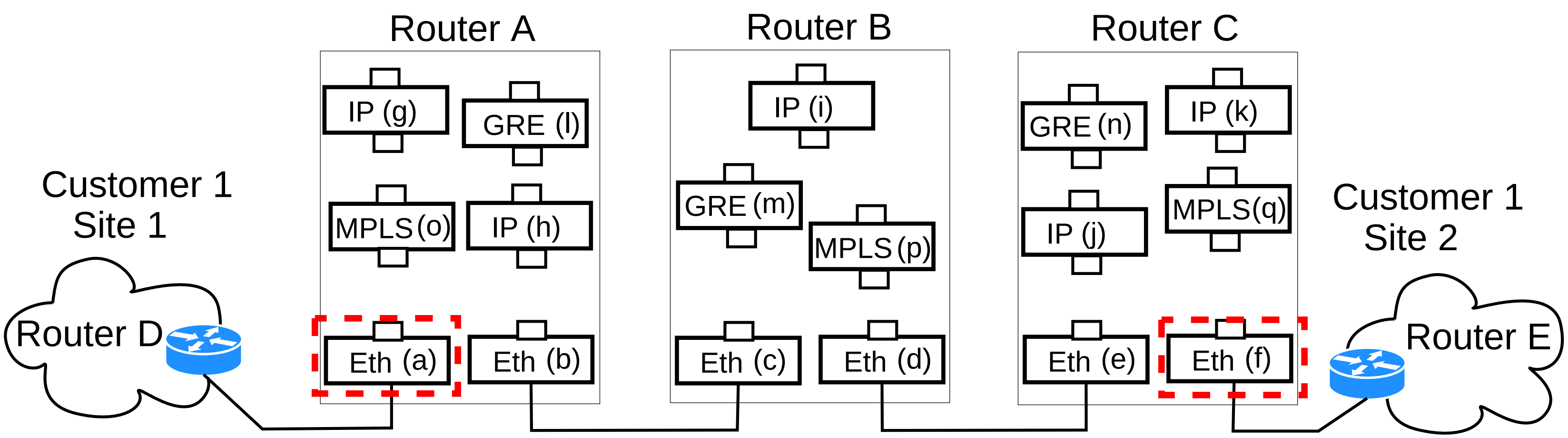
Configure connectivity between sites S1 and S2 of customer C1



Virtual Private Networks

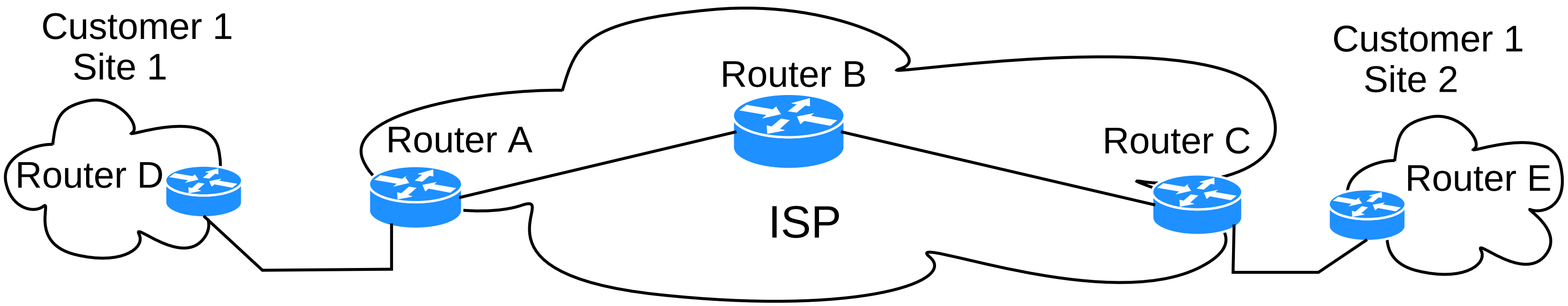


Configure connectivity between sites S1 and S2 of customer C1

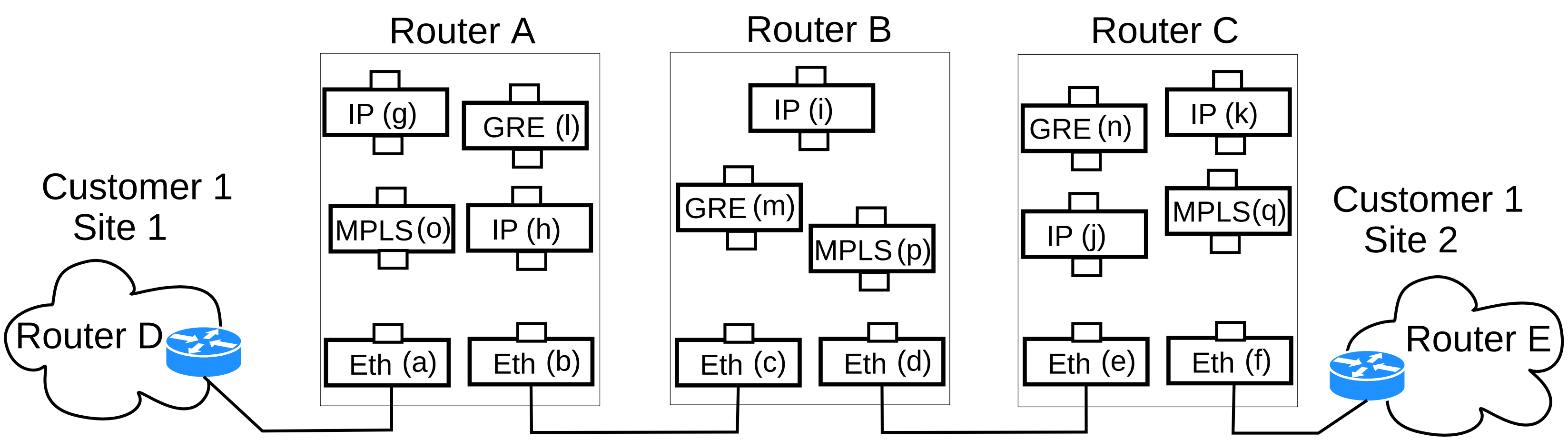


High-level goal: Configure connectivity between the customer-facing interfaces <ETH,A,a> and <ETH,C,f> for traffic between C1-S1 and C1-S2

Virtual Private Networks



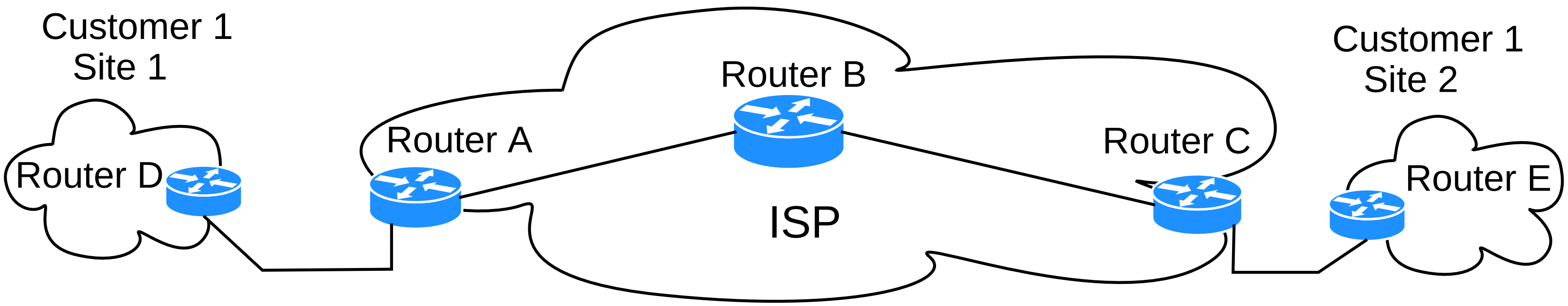
Configure connectivity between sites S1 and S2 of customer C1



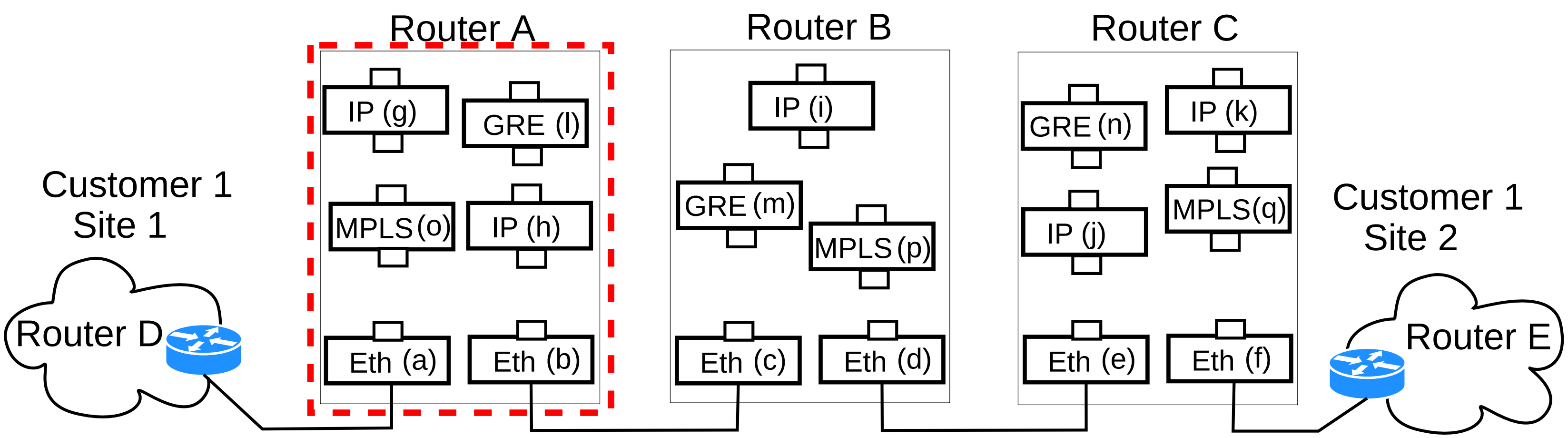
Routers inform the NM of their connectivity and their modules

The figure represents the network map as seen by the NM

Virtual Private Networks



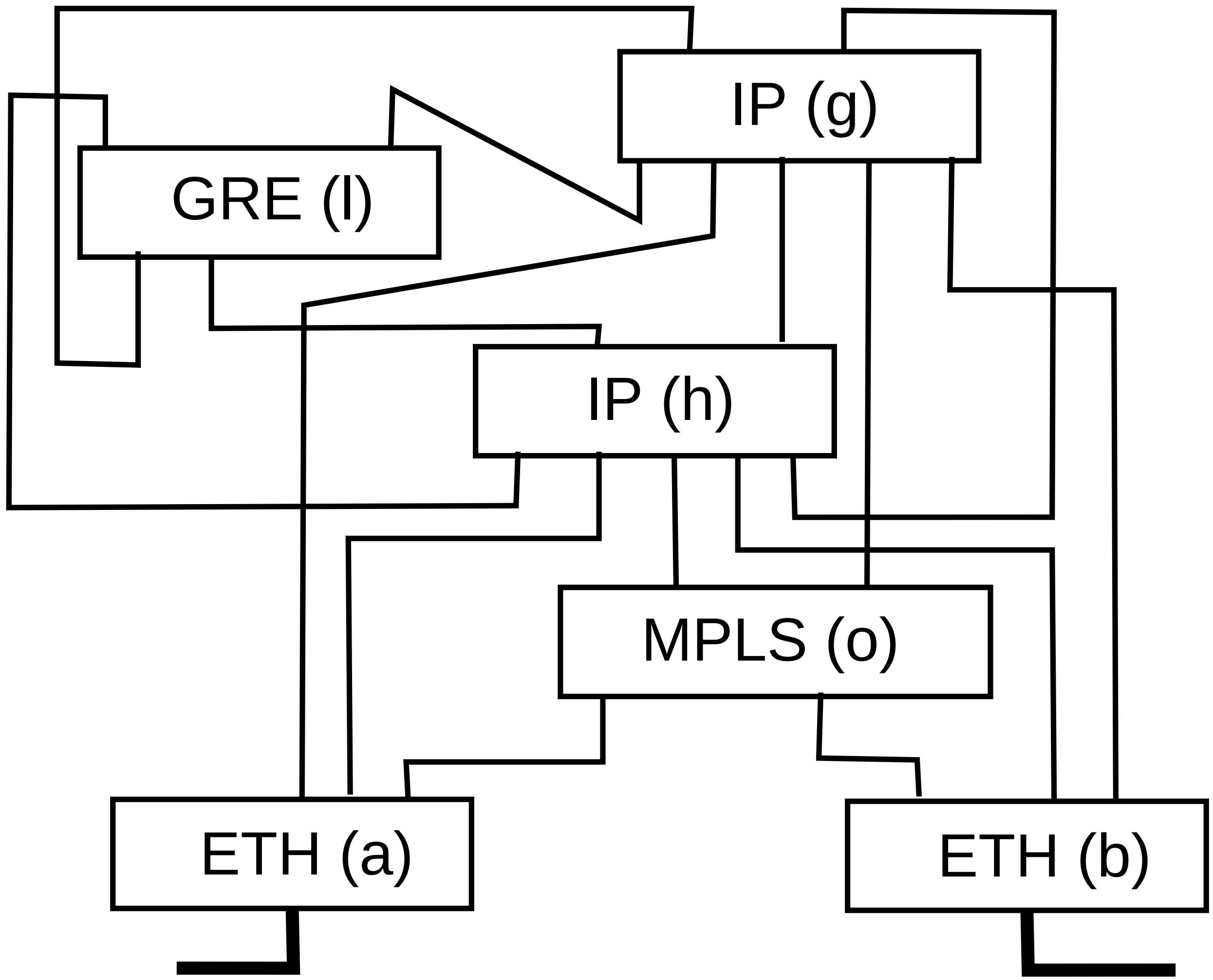
Configure connectivity between sites S1 and S2 of customer C1



NM is also presented with the abstraction for various modules

This includes pipe connectivity and switch capabilities

NM Implementation



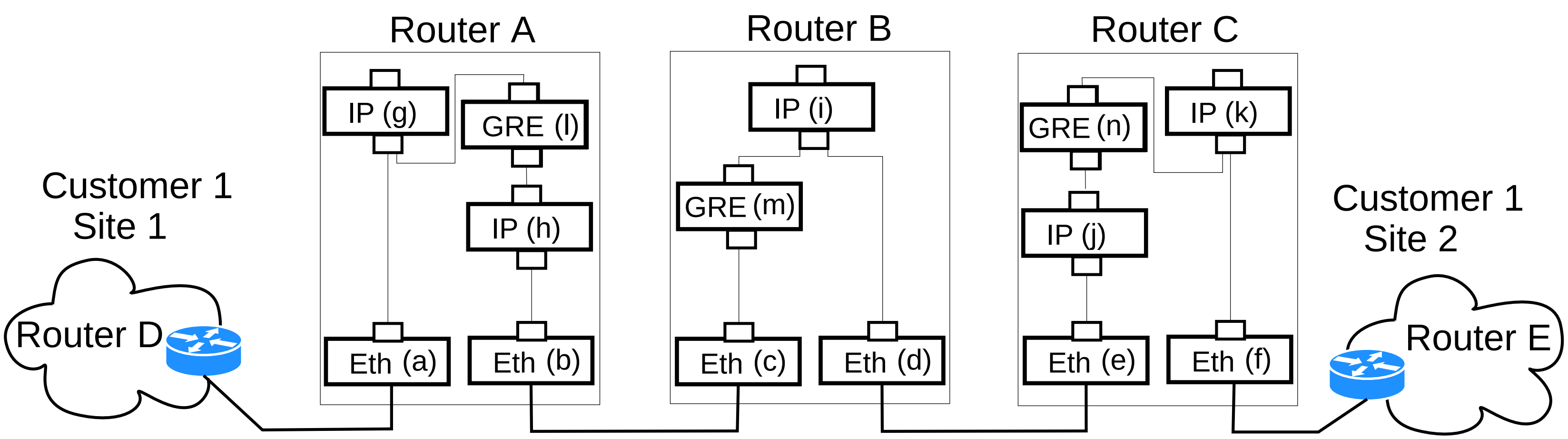
Potential Connectivity sub-graph for router A

NM Implementation

Path Finder

- Find *all* paths between any two modules
- Depth First Search across the graph

For example, *find_path* (<ETH,A,a>, <ETH,C,f>)



One possible path (using GRE-IP Tunnel)

a, g, l, h, b, c, i, d, e, j, n, k, f

NM Implementation

For example, *find_path* (<ETH,A,a>, <ETH,C,f>)

- Using IP-IP Tunnel: *a, g, h, b, c, i, d, e, j, k, f*
- Using GRE-IP Tunnel: *a, g, l, h, b, c, i, d, e, j, n, k, f*
- Using MPLS: *a, g, o, b, c, p, d, e, q, k, f*
- Using IP-IP over MPLS
- Using GRE-IP over MPLS
- Using IP-IP over MPLS only between A and B
- Using IP-IP over MPLS only between B and C
- Using GRE-IP over MPLS only between A and B
- Using GRE-IP over MPLS only between B and C

NM Implementation

For example, *find_path* (<ETH,A,a>, <ETH,C,f>)

- Using IP-IP Tunnel: *a, g, h, b, c, i, d, e, j, k, f*
- Using GRE-IP Tunnel: *a, g, l, h, b, c, i, d, e, j, n, k, f*
- Using MPLS: *a, g, o, b, c, p, d, e, q, k, f*
- Using IP-IP over MPLS
- Using GRE-IP over MPLS
- Using IP-IP over MPLS only between A and B
- Using IP-IP over MPLS only between B and C
- Using GRE-IP over MPLS only between A and B
- Using GRE-IP over MPLS only between B and C

NM Implementation

For example, *find_path* (<ETH,A,a>, <ETH,C,f>)

- Using IP-IP Tunnel: *a, g, h, b, c, i, d, e, j, k, f*
- Using GRE-IP Tunnel: *a, g, l, h, b, c, i, d, e, j, n, k, f*
- Using MPLS: *a, g, o, b, c, p, d, e, q, k, f*
- Using IP-IP over MPLS
- Using GRE-IP over MPLS
- Using IP-IP over MPLS only between A and B
- Using IP-IP over MPLS only between B and C
- Using GRE-IP over MPLS only between A and B
- Using GRE-IP over MPLS only between B and C

NM needs to be able to choose amongst the paths based on high-level directives/metrics

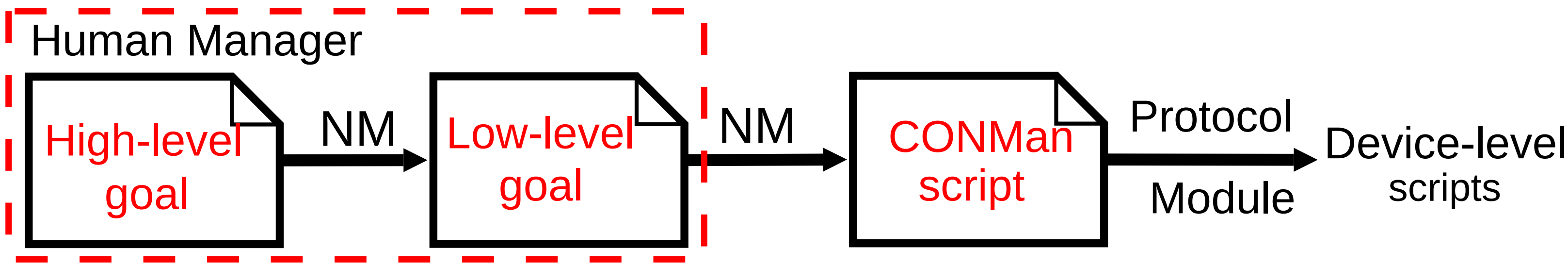
NM Implementation

For example, *find_path* (<ETH,A,a>, <ETH,C,f>)

- Using IP-IP Tunnel: *a, g, h, b, c, i, d, e, j, k, f*
- Using GRE-IP Tunnel: *a, g, l, h, b, c, i, d, e, j, n, k, f*
- Using MPLS: *a, g, o, b, c, p, d, e, q, k, f*
- Using IP-IP over MPLS
- Using GRE-IP over MPLS
- Using IP-IP over MPLS only between A and B
- Using IP-IP over MPLS only between B and C
- Using GRE-IP over MPLS only between A and B
- Using GRE-IP over MPLS only between B and C

NM needs to be able to choose amongst the paths based on high-level directives/metrics

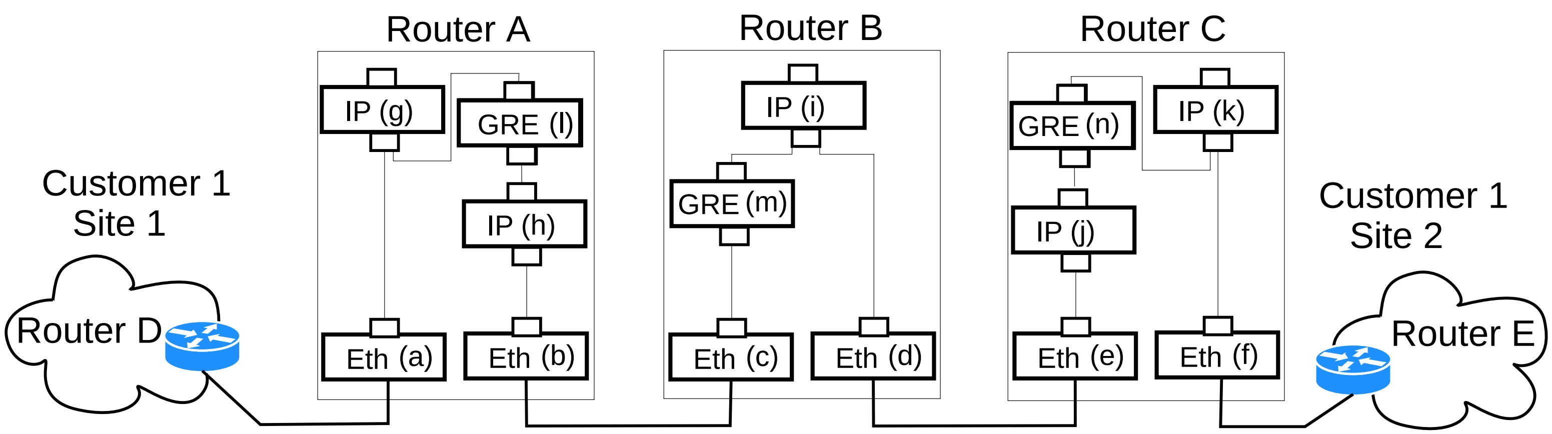
NM Implementation



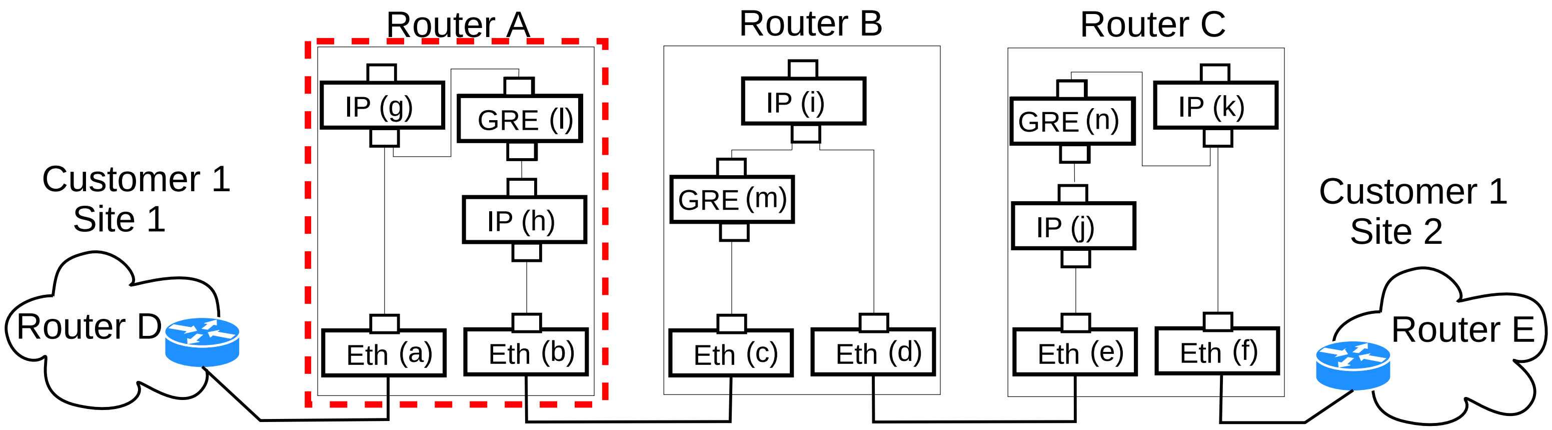
High-level goal: Configure connectivity between the customer-facing interfaces <ETH,A,a> and <ETH,C,f> for traffic between C1-S1 and C1-S2

Low-level goal: Configure the path comprising of modules *a*, *g*, *l*, *h*, *b*, *c*, *i*, *d*, *e*, *j*, *n*, *k*, *f*

NM Implementation



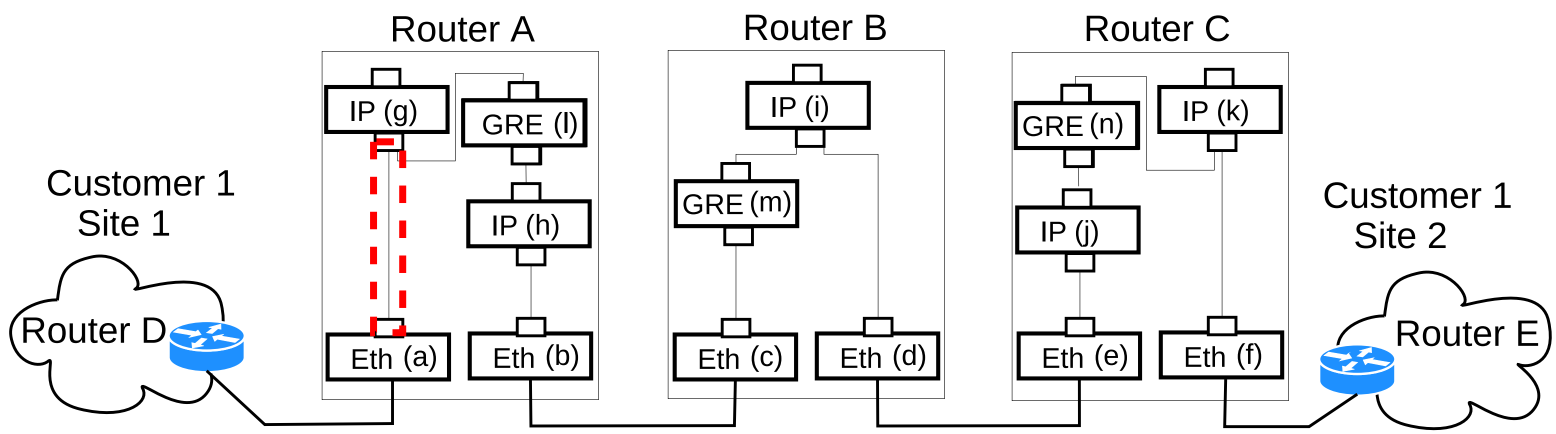
NM Implementation




```
P0 = create (pipe, <IP,A,g>, <ETH,A,a>, None, None, None)
P1 = create (pipe, <IP,A,g>, <GRE,A,l>, <IP,C,k>, <GRE,C,n>,
trade-off: in-order delivery, trade-off: error-rate)
create (switch, <IP,A,g>, [P0, dst:C1-S2 ⇒ P1])
create (switch, <IP,A,g>, [P1 ⇒ P0, S2-gateway])
P2 = create (pipe, <GRE,A,l>, <IP,A,h>, <GRE,C,n>, <IP,C,j>,
None)
create (switch, <GRE,A,l>, P1, P2)
P3 = create (pipe, <IP,A,h>, <ETH,A,b>, <IP,B,i>, <ETH,B,c>,
None)
create (switch, <IP,A,h>, P2, P3)

create (switch, <ETH,A,b>, P3,P4)
```

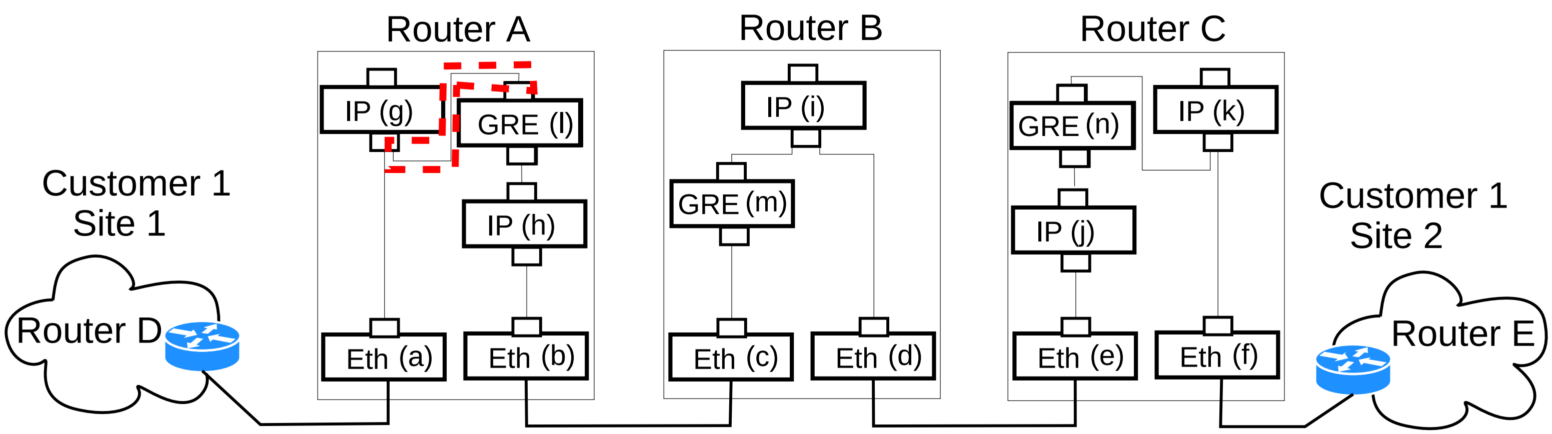
NM Implementation




```
P0 = create (pipe, <IP,A,g>, <ETH,A,a>, None, None, None)
P1 = create (pipe, <IP,A,g>, <GRE,A,l>, <IP,C,k>, <GRE,C,n>,
trade-off: in-order delivery, trade-off: error-rate)
create (switch, <IP,A,g>, [P0, dst:C1-S2 ⇒ P1])
create (switch, <IP,A,g>, [P1 ⇒ P0, S2-gateway])
P2 = create (pipe, <GRE,A,l>, <IP,A,h>, <GRE,C,n>, <IP,C,j>,
None)
create (switch, <GRE,A,l>, P1, P2)
P3 = create (pipe, <IP,A,h>, <ETH,A,b>, <IP,B,i>, <ETH,B,c>,
None)
create (switch, <IP,A,h>, P2, P3)

create (switch, <ETH,A,b>, P3,P4)
```

NM Implementation

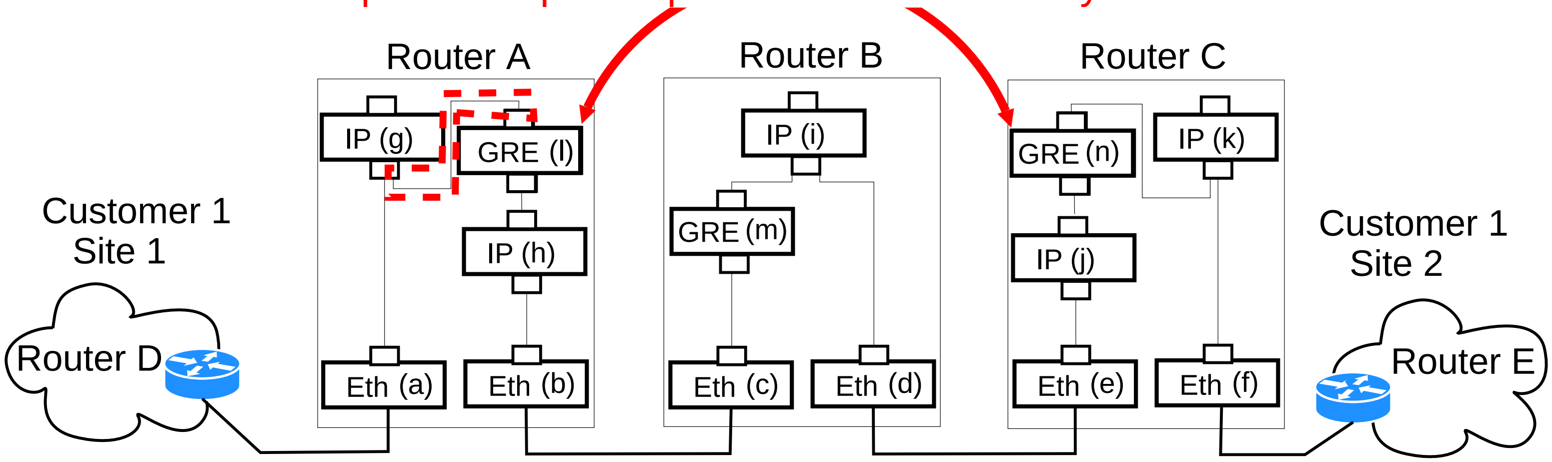



```
P0 = create (pipe, <IP,A,g>, <ETH,A,a>, None, None, None)
P1 = create (pipe, <IP,A,g>, <GRE,A,l>, <IP,C,k>, <GRE,C,n>,
trade-off: in-order delivery, trade-off: error-rate)
create (switch, <IP,A,g>, [P1 ⇒ P0, S2-gateway])
P2 = create (pipe, <GRE,A,l>, <IP,A,h>, <GRE,C,n>, <IP,C,j>,
None)
create (switch, <GRE,A,l>, P1, P2)
P3 = create (pipe, <IP,A,h>, <ETH,A,b>, <IP,B,i>, <ETH,B,c>,
None)
create (switch, <IP,A,h>, P2, P3)

create (switch, <ETH,A,b>, P3,P4)
```

NM Implementation

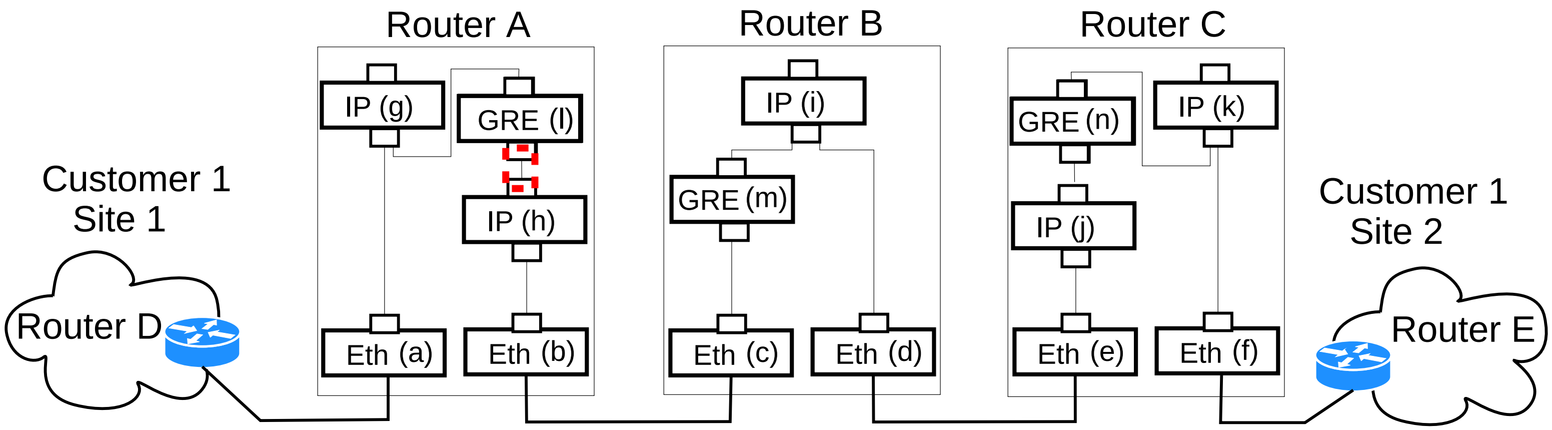
GRE modules use conveyMessage() to exchange protocol-specific parameters such as key values




```
P0 = create (pipe, <IP,A,g>, <ETH,A,a>, None, None, None)
P1 = create (pipe, <IP,A,g>, <GRE,A,l>, <IP,C,k>, <GRE,C,n>,
trade-off: in-order delivery, trade-off: error-rate)
create (switch, <IP,A,g>, [P1 ⇒ P0, S2-gateway])
P2 = create (pipe, <GRE,A,l>, <IP,A,h>, <GRE,C,n>, <IP,C,j>,
None)
create (switch, <GRE,A,l>, P1, P2)
P3 = create (pipe, <IP,A,h>, <ETH,A,b>, <IP,B,i>, <ETH,B,c>,
None)
create (switch, <IP,A,h>, P2, P3)

create (switch, <ETH,A,b>, P3,P4)
```

NM Implementation

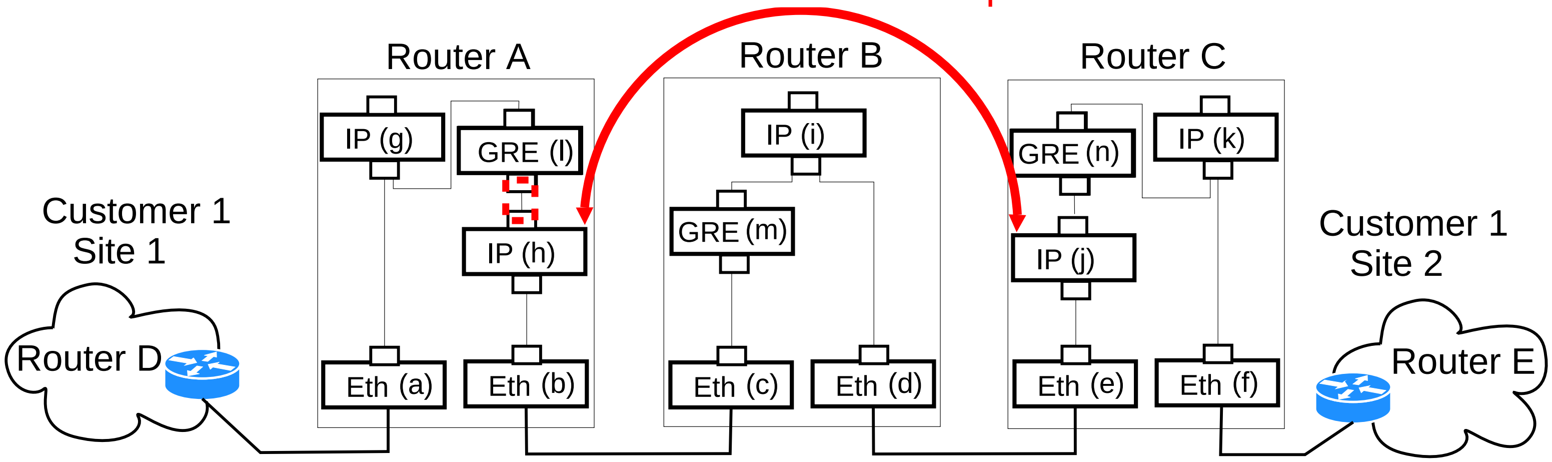



```
P0 = create (pipe, <IP,A,g>, <ETH,A,a>, None, None, None)
P1 = create (pipe, <IP,A,g>, <GRE,A,l>, <IP,C,k>, <GRE,C,n>,
trade-off: in-order delivery, trade-off: error-rate)
create (switch, <IP,A,g>, [P0, dst:C1-S2 ⇒ P1])
create (switch, <IP,A,g>, [P1 ⇒ P0, S2-gateway])
P2 = create (pipe, <GRE,A,l>, <IP,A,h>, <GRE,C,n>, <IP,C,j>,
None)
create (switch, <GRE,A,l>, P1, P2)
P3 = create (pipe, <IP,A,h>, <ETH,A,b>, <IP,B,i>, <ETH,B,c>,
None)
create (switch, <IP,A,h>, P2, P3)

create (switch, <ETH,A,b>, P3,P4)
```

NM Implementation

IP modules use conveyMessage() to exchange
IP addresses of tunnel end-points




```
P0 = create (pipe, <IP,A,g>, <ETH,A,a>, None, None, None)
P1 = create (pipe, <IP,A,g>, <GRE,A,l>, <IP,C,k>, <GRE,C,n>,
trade-off: in-order delivery, trade-off: error-rate)
create (switch, <IP,A,g>, [P0, dst:C1-S2 ⇒ P1])
create (switch, <IP,A,g>, [P1 ⇒ P0, S2-gateway])
P2 = create (pipe, <GRE,A,l>, <IP,A,h>, <GRE,C,n>, <IP,C,j>,
None)
create (switch, <GRE,A,l>, P1, P2)
P3 = create (pipe, <IP,A,h>, <ETH,A,b>, <IP,B,i>, <ETH,B,c>,
None)
create (switch, <IP,A,h>, P2, P3)

create (switch, <ETH,A,b>, P3,P4)
```

NM Implementation

```
#!/bin/bash
# Insert the GRE-IP kernel module
insmod /lib/modules/2.6.14-2/ip_gre.ko
# Create the GRE tunnel with the appropriate key
ip tunnel add name greA mode gre remote 204.9.169.1 local
204.9.168.1 ikey 1001 okey 2001 icsum ocsum iseq oseq
ifconfig greA 192.168.3.1
# Enable Routing
echo 1 > /proc/sys/net/ipv4/ip_forward
# Create IP routing from customer to tunnel
echo 202 tun-1-2 >> /etc/iproute2/rt_tables
ip rule add to 10.0.2.0/24 table tun-1-2
ip route add default dev greA table tun-1-2
# Create IP routing from tunnel to customer
echo 203 tun-2-1 >> /etc/iproute2/rt_tables
ip rule add iff greA table tun-2-1
ip route add default dev eth1 table tun-2-1
ip route add to 204.9.169.1 via 204.9.168.2 dev eth2
```

Linux script generated by the protocol modules

NM Implementation

NM-generated CONMan script snippet

```
P0 = create (pipe, <IP,A,g>, <ETH,A,a>, None,
None, None)

P2 = create (pipe, <GRE,A,l>, <IP,A,h>,
<GRE,C,n>, <IP,C,j>, None)

create (switch, <GRE,A,l>, P1, P2)
```

Module-generated Linux script snippet

```
# Insert the GRE-IP kernel module
insmod /lib/modules/2.6.14-2/ip_gre.ko

# Create the GRE tunnel with the appropriate key
ip tunnel add name greA mode gre remote 204.9.169.1
local 204.9.168.1 ikey 1001 okey 2001 icsum ocsum iseq
oseq
```

Linux script generated by the protocol modules

Talk Outline

- Introduction
- CONMan Overview
- Module Abstraction
- CONMan primitives
- Implementation
- Conclusions and Future Work

Conclusions

CONMan: Complexity Oblivious Network Mgmt.

- Strives to reduce protocol-specific information in the management interface of protocols

Balances division of functionality

- Management applications don't deal with protocol-specific details
- Protocols still need low-level details to operate
- Protocol implementor needs to understand protocol operation

Future Work

- Scalability
  - Load on the NM
  - Dynamic network configuration
- Multiple NMs
- Management channel
- NM design
  - User-side
  - Network-side
- Deployment model


ありがとう