Predicate Routing: Enabling Controlled Networking

Timothy Roscoe
Intel Research at Berkeley
2150 Shattuck Avenue,
Berkeley, CA 94704, USA
troscoe@intel-research.net

Steve Hand
University of Cambridge
Computer Laboratory
Cambridge CB3 OFD, UK
steven.hand@cl.cam.ac.uk

Rebecca Isaacs
Microsoft Research
7 J.J. Thomson Avenue
Cambridge CB3 OFB, UK
risaccs@microsoft.com

Richard Mortier
Microsoft Research
7 J.J. Thomson Avenue
Cambridge CB3 OFB, UK
mort@microsoft.com

Paul Jardetzky
Sprint ATL
1 Adrian Court
Burlingame, CA 94010, USA
pjardetzky@sprintlabs.com

1. INTRODUCTION AND MOTIVATION

The Internet lacks a coherent model which unifies security (in terms of where packets are allowed to go) and routing (where packets should be sent), even in constrained environments. While automated configuration tools are appearing for parts of this problem, a general solution is still unavailable. Routing and firewalling are generally treated as separate problems, in spite of their clear connection. In particular, security policies in data hosting centers, enterprise networks, and backbones are still by and large installed manually, and are prone to problems from errors and misconfigurations. In this paper, we present *Predicate Routing* (PR) as a solution to this problem. We briefly describe our centralized implementation and then outline the extension of Internet routing protocols to support Predicate Routing.

In current IP networks, the state of the system is primarily represented in an imperative fashion: routing tables and firewall rulesets local to each node strictly specify the action to be performed on each arriving packet. In contrast, Predicate Routing represents the state of the network declaratively as a set of boolean expressions associated with links which assert which kinds of packet can appear where. From these expressions, routing tables and filter rules are *derived* automatically. Conversely, the consequences of a change in network state can be *calculated* for any point in the network (link, router, or end system), and predicates derived from known configuration state of routers and links. This subsumes notions of both routing and firewalling.

We use the phrase "controlled networking" to refer to environments where every packet flow in a network has been explicitly allowed or "white listed", possibly by an automated process. Controlled networking using Predicate Routing gives precise assurances about the presence of network packets, even when network elements cannot provide the filtering and packet discrimination required by a naive, manually configured approach.

Where ideal security is infeasible with the given infrastructure and topology, Predicate Routing can be used to guide risk assessments and security tradeoffs by providing complete information about what packets are allowed to traverse each link. Furthermore, Predicate Routing aids packet traceback, by allowing those properties of a packet (such as origin machine) which cannot be directly observed at most points in the network to be logically inferred from observable properties.

Note that this differs from the various forms of coordinated firewall control or distributed firewalling in that predicates do not specify traffic rules but rather capture both current and desired network state. From this state appropriate firewall rulesets may be synthesised.

Since our declarative view of network state is very different from traditional routing concepts, we first give an abstract view of how Predicate Routing represents a network, as a precursor to discussing its application. We then detail two scenarios where Predicate Routing can be applied: a centralized environment such as a datacenter, and a distributed one such as a collection of peer networks.

2. PREDICATE ROUTING: FRAMEWORK

Predicate Routing is concerned as much with where particular packets in an IP network *can appear* as with where they should be sent. While the term "reachability" in conventional IP networks refers to the notion that some packets can reach a given point in the network, in Predicate Routing this notion is packet-specific, and subsumes both the notion of firewalling (ensuring that a particular destination is unreachable for that packet), and routing (attempting to ensure that the desired destination is reachable for the packet). Predicate Routing achieves this by employing a non-traditional abstraction of network properties. The upper half of Figure 1 shows a typical, simple IP network composed of 2 routers and 4 end nodes. Links are bidirectional, and connect ports on routers and nodes.

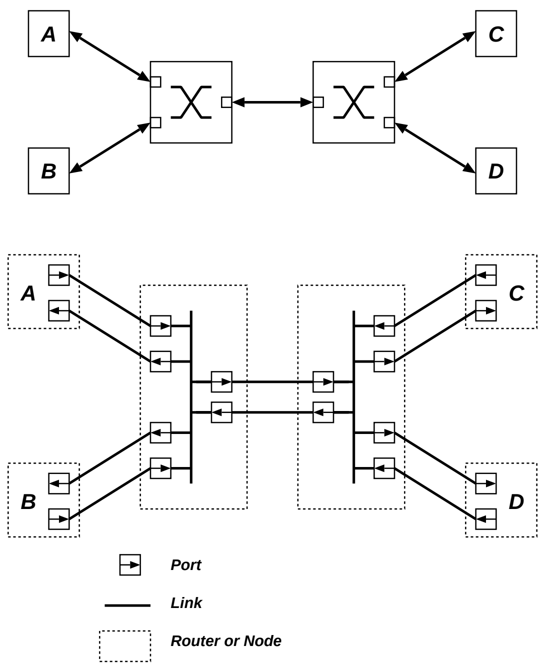


Figure 1: A traditional representation of a network, and the same network in Predicate Routing terms.

The lower half shows how the same network is represented in Predicate Routing. Some differences are immediately apparent. The most obvious is that the switch-centric representation of has been replaced by one made up of ports and links. Indeed, in Predicate Routing the notion of a switch or router *per se* is important only from an implementation standpoint (as a collection of ports with a single control interface and shared resources). Ports are now *unidirectional*, and so there are twice as many. At the same time, links are regarded as broadcast media, and so are neither unidirectional nor bidirectional. Finally, the "inside" of a router or switch is equivalent to an external network link from the point of view of Predicate Routing.

While the network has become more complex (more links, more ports), the elements making it up have become much simpler, making it easier to automate reasoning about the network. Firstly, links are now passive media elements, and so it makes sense to talk about a packet being "present" on a link without needing to specify the direction in which it is traveling. Secondly, ports have a single input and a single output and have subsumed the role of switches and routers in the traditional representation, and so they can be viewed as "gates" which allow some packets through (possibly modifying them in the process) and disallow others. These two abstractions, (unidirectional) ports and (non-directional) links, form the basis for Predicate Routing.

Predicates

Predicate Routing views the state of an IP network as a set of logical expressions—predicates—that refer to properties of packets at each point in the network. In traditional routing, network state is represented as a forwarding table at each router, which can be viewed as a function from packet properties to outgoing router ports. In contrast, in Predicate Routing a packet can potentially appear on any router output port which does not explicitly disallow it; and the forwarding table is represented as a set of output filters. These two views of a router are equivalent (Predicate Routing is just as expressive), but the more declarative approach taken by Predicate Routing simplifies automated reasoning about network state.

The primitive terms of a predicate are packet attributes like source or destination IP address, port numbers, protocols, etc. A simple (and highly restrictive) link predicate might be:

```
Proto(TCP) AND DestPort(80) AND
        DestAddr(10.10.1.2)
```

While all the attributes in this example are directly observable from the packet header, one can define other attributes which are not immediately observable, such as the origin machine of the packet (in the presence of potential source address spoofing), a particular flow or path the packet is part of, etc. Predicate Routing can allow these non-observable attributes to be inferred from the network state. Routers generally operate only on observable properties of packets.

Four kinds of predicate are involved in representing network state: *link* or *network* predicates, *switch* predicates, *port* predicates, and *filter* predicates.

Link and network predicates

A *link predicate* is an assertion about the properties of packets that can be seen on a network link. Recall that links do not have a direction, so a single boolean expression in disjunctive normal form[1] suffices to describe everything that can be observed on the link[2].

Switch predicates

A *switch predicate* is an assertion about packets that may be seen "inside" a router or switch—packets which may potentially traverse the switching fabric. We treat the inside of a router as a "sea of packets", with no notion of which port a packet entered on, or which port or ports the packet is leaving on, hence there is a symmetry between the "insides" of switches and routers, and the "outsides" of links, with a corresponding symmetry between input and output ports.

This is a relatively simple model of a router. Modern IP routers are rather more complex than this: in particular many combine the functions of switching (based on MAC address) and routing (at the IP layer) using the concept of virtual LANs (VLANS). In this paper we use the terms switch

[1] i.e. an OR of a series of AND-connected compound terms.

[2] While the idea generalizes to broadcast networks, where the predicate refers to the packets that can be present on a given segment, in this paper we restrict our discussion to switched point-to-point links.

and router interchangeably. We can capture this complexity of networking equipment in several ways. Firstly, if we treat VLANs as if they were real Ethernet broadcast domains, a Predicate Routing port now corresponds to the IP interface of the VLAN on the switch, as opposed to the physical ports. VLANs consequently have network predicates associated with them. A better approach integrates the VLAN notion into Predicate Routing's model of the switch, but that is beyond the scope of this paper.

Port predicates

A *port predicate* is an assertion about the properties of packets passing through a switch port. We view ports as unidirectional. A port predicate is identical in form to a link or network predicate.

Filter predicates

In the Predicate Routing model, input and output ports apply filters (which may be trivially simple). Thus, in addition to the port predicate (which asserts the properties of traffic flowing through the port), each port has an associated *filter predicate*, which asserts the properties of traffic which *can* flow through the port. The filter predicate for a port expresses the filter configuration which currently applies to the port. Most modern IP routers provide some facility for input port filtering, sometimes referred to as Access Control Lists, for which there is a natural mapping to input port filter predicates. Output port filter predicates are also naturally mapped onto real router configuration properties in the form of IP routing table entries. To understand this, consider the set of routing table entries in the router which cause packets to be forwarded to a given port. Each component term in the output filter predicate is the property that the packet destination address matches the address prefix of the table entry. The complete filter predicate is the OR of these terms.

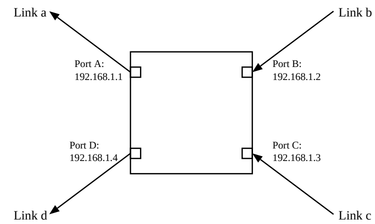
Increasingly, all but low-end routers and very high-performance core IP switches support policy routing, where a routing decision is made based not only on destination address, but also on source address, protocol, ports, etc. This additional router state information also maps naturally onto output port filters.

Relations between predicates

Figure 2 shows a very simple example of a network configuration, together with corresponding predicates which combine both the filtering and routing configuration of the network in a unified model of network state. It is clear that the four types of predicates are closely dependent on each other:

1. The port predicate for an input port on a switch is the AND of the network predicate of the attached network with the filter predicate for the port. This expresses what the port filter does: it constrains the traffic that enters the switch from the network through the port. Similarly, the port predicate for an output port on a switch is the AND of the switch predicate, and the filter predicate for the port.

2. A link predicate is the OR of the port predicates for the switch output ports which are attached to the link.




```
Routing table:
 10.1.0.0/16 -> 192.168.1.1
 10.2.0.0/16 -> 192.168.1.4

Input filters:
 Port B:
  Allow Source = 10.3.0.0/16
  Deny all
 Port C:
  Allow Source = 10.4.0.0/16
  Deny all
```

Predicates:

Filter A:
 Dest = 10.1.0.0/16
Filter B:
 Source = 10.3.0.0/16
Filter C:
 Source = 10.4.0.0/16
Filter D:
 Dest = 10.2.0.0/16

Switch:
 Source = 10.3.0.0/16 OR Source = 10.4.0.0/16

Link A:
 Dest = 10.1.0.0/16
 AND (Source = 10.3.0.0/16 OR Source = 10.4.0.0/16)
Link B:
 None (anything)
Link C:
 None (anything)
Link D:
 Dest = 10.2.0.0/16
 AND (Source = 10.3.0.0/16 OR Source = 10.4.0.0/16)

Port A:
 Dest = 10.1.0.0/16
 AND (Source = 10.3.0.0/16 OR Source = 10.4.0.0/16)
Port B:
 Source = 10.3.0.0/16
Port C:
 Source = 10.4.0.0/16
Port D:
 Dest = 10.2.0.0/16
 AND (Source = 10.3.0.0/16 OR Source = 10.4.0.0/16)

Figure 2: A very simple network showing predicates

This simply expresses the fact that any packet enters a link through one switch output port.

3. Similarly, the switch predicate for a switch is the OR of the port predicates for all the input ports on the switch.

Discussion

Given these relations, we note that in a closed network, link, port, and switch predicates can be derived from knowing only the network topology and all the filter predicates.

Strictly speaking, the notion of a port predicate is redundant in this framework: port predicates are entirely determined by link and filter predicates and so don't convey any additional information about network state. However, they are important as intermediate terms when applying the logical framework in a real implementation, as in the next section. Also, although Predicate Routing's logical framework treats networks and routers identically, in practice the difference clearly matters.

When the network is connected to other systems (like the rest of the Internet), these predicates can still be derived from a combination of the filter predicates and the link predicates at attachment points. Alternatively, predicates on links entering an administrative domain can be defined in accordance with service level agreements.

Note also that we can derive or "prove" additional properties of packets at given point in the network, not directly observable from the packet itself. For example, in Figure 2, if we see a packet on link **a** with a source address in 10.3.0.0/16, we can infer that the packet traversed link **b**. While this example is trivial, in more complex networks this can be a powerful tool for reducing human error in network configuration.

The use of declarative languages to detect network and system misconfiguration has been investigated [10]. Predicate Routing provides the logical framework in which these techniques can be applied to network management, for example automatic detection of BGP misconfiguration, a well known problem [9].

The four types of predicates, together with the relations between them, faithfully model the packet forwarding behavior of an IP network, both filtering and routing, and together form a consistent logical system. However, this representation of network state is also highly amenable to manipulation by programs controlling network elements. Because it gives complete information of possible paths traversed by packets, it is highly appropriate for implementing a controlled networking environment. In the next section, we describe one approach to this, using a logically centralized approach for managing and controlling the network in a datacenter.

3. CENTRALIZED PREDICATE ROUTING

We applied Predicate Routing to the problem of controlled networking in a "public computing platform", where (possibly distributed) third-party applications are hosted on a shared cluster of servers. In this case it is reasonable to implement a centralized network "control plane" with out-of-band control of network elements. Here we sketch the algorithm we use and our prototype design; in Section 4 we outline how Predicate Routing can be applied in the distributed case.

The control plane function we are interested in here is configuring the routers that make up the cluster interconnect in such a way that the IP packet flows required (and specified) by the hosted applications are allowed, but (ideally) no other flows can arrive at a server machine. This function must be performed online and incrementally, as the application mix is dynamic.

The cluster interconnect in our prototype at Sprint Labs is a network composed of two Enterasys SSR-8600 switch-routers and a front-end Cisco 11800 Layer-7 switch, which connect 36 servers (each with dual network interfaces) together and to the Internet. Both types of switch support policy routing and input port filters at the IP layer.

The control plane instantiates "switch driver" objects for each switch in the cluster. The driver models a switch's capabilities (how many per-port or per-line card filters are supported, for instance), and establishes a control connection to the real hardware (through SNMP or a command line emulator). In addition, the driver exports an interface to the routing algorithm corresponding to Predicate Routing's "ports and links" representation.

The routing algorithm to place a new flow first calculates a candidate path for the flow, then operates a flooding algorithm starting at the flow origin. For each port encountered, the switch driver is consulted to appropriately modify its filter predicate: either to let the flow through if the port is on the candidate path, or else to block packets from the flow (and any other unauthorized flows). The flooding stops at any port whose port predicate is unchanged as a result of the operation. The path is rejected if a link predicate at the edge of the cluster (i.e. at a server) violates an administrative security constraint, implying that placing the path would allow unacceptable packets to arrive at a host.

The switch driver abstraction allows great flexibility: a port is free to not apply a requested filter due to lack of functionality or the switch running out of resources, as long as the new flow is admitted along the candidate path. In this way the consequences are propagated "downstream", where even in a simple network other ports will often compensate and preserve the controlled environment.

Performance with our prototype is adequate, even though the control plane is implemented in the interpreted language Python. While the theoretical complexity of the algorithm is moderately high[3], in practice most loops terminate early with the reasonably powerful switch capabilities we have, resulting in much better scaling than might be expected, even with larger topologies. Communication latency with switches tends to dominate; this can in many cases be overlapped with the route computation.

4. DISTRIBUTED PREDICATE ROUTING

Controlled networking is also useful in the wider area Internet, although in this case a centralized scheme is not suitable. In this section we discuss how one might implement Predicate Routing in the Internet by modifying existing Internet routing protocols, specifically the IGP IS-IS and the EGP BGPv4.

Link-State Routing Protocols

IS-IS [3] is a link-state protocol adapted from the ISO CLNS protocol. Each router effectively broadcasts information about the other routers to which it is connected (its *link states*) in the form of *link state PDUs (LSPs)*. Routers store the LSPs they receive in a database, and then run a shortest path algorithm over this database to discover the interface on which they should transmit packets for destinations within the network. Much of this discussion also applies to OSPF, the other main link-state intra-domain routing protocol in use in the Internet today.

The link-state information is transmitted in variable length

[3] A detailed analysis is beyond the scope of this paper.

type-length-value fields appended to the LSP header information. As IS-IS was not originally intended for routing IP, it effectively distributes two forms of link-state information: the connectivity information, expressed in terms of CLNP nodes[4] and their adjacencies, and the IP information, expressed in terms of the IP prefixes available to a CLNP node.

We can extend IS-IS as follows. First, rather than simply advertise the destination IP prefixes available at a node, a set of predicates are advertised, potentially with associated resource usage information. Second, although LSP forwarding and database building takes place as normal, sets of predicates effectively form *views* of this database, defining the connectivity available to that set. The shortest path computation is run over each such view, producing a set of shortest path results, one for each collection of predicates. These can then be remerged, as allowed by the predicates in place, to create the forwarding tables to be used to actually route packets. This results in one (or more) forwarding tables that contain predicates to be applied to packets, and for each predicate, a corresponding output port on which packets can be transmitted.

Performance Implications

The performance impact of the above scheme can be separated into traffic and computation costs at both the data and control planes. The traffic impact is fairly easy to imagine: the network sees less user traffic (due to packets being filtered early), but more control traffic (since LSPs are now larger and potentially more frequent). If we expect predicates to be slowly varying (e.g. changing on the order of hours), the increased routing protocol bandwidth should not be significant.

Perhaps greater concerns are the additional computational overhead, and the risk of increased routing instability. In terms of the former, it is true that some additional overhead will occur due to the need to perform shortest path computations for every "view" of the network. However several factors mitigate this cost: firstly, we expect the number of views to be much smaller than the number of predicates, with many predicates mapping onto an empty or unconnected subgraph. Secondly, it is possible in some cases to infer shortest paths for smaller subgraphs; and thirdly, many subgraphs will be considerably smaller than the entire network (and may even be degenerate). Forwarding table performance should also not be an issue [6].

We don't expect our modifications to decrease routing stability, since the same topological information is communicated both cases. However, further investigation is a topic for future work.

External gateway protocols

Unlike OSPF and IS-IS, BGP is a path-vector routing protocol without an explicit notion of a link. Instead, each router advertises a cost to the destinations it can reach, and chooses e.g. the cheapest route to a particular destination. It then re-advertises its chosen routes to other routers, adding in a cost component to account for their own presence on the path to the destination.

BGP already has extensive support for filters, for routers to control the routes advertised to other routers and the route advertisements received from other routers. However, these filters are currently entered and managed manually. It would seem that the natural way to implement predicates in BGP is to extend BGP to allow automatic distribution and installation of filters, but the details of such an approach, in particular uhow to deal with transferring such information between administrative domains, are future work.

Discussion

Predicate Routing permits incremental deployment: as routers are upgraded to support the routing protocol extensions described above, the inferences which may be made about the state of the network become stronger. Even with a small number of enhanced routers, however, useful information is available to operators. For example, access routers could be upgraded initially which suffices to provide automated management of ingress filtering. As incremental deployment proceeds, the ability of the system to infer the origin(s) of traffic generated by attacks (for instance) increases.

A practical deployment of Predicate Routing would benefit from the ability to compare the desired and actual network state. This requires a mechanism to accurately snapshot the current network configuration. This presents a challenge in a highly dynamic environment such as the Internet and is a matter for future work.

Enhancement of Predicate Routing as presented should include the ability to refer to predicates as first class entities, in particular across administrative boundaries. This naming of predicates enables scoping and modularization thereby allowing aggregation and transformation of predicates, late biding of policy and information hiding between networks.

5. RELATED WORK

Predicate Routing builds on several ideas from the areas of firewalling, virtual private networks and signaling.

Like distributed [1] or embedded firewalling, for example, we aim to have an explicit notion of which packets may be transmitted where, and we attempt to automatically enforce this notion at multiple redundant locations. We do not rely upon a centralized security policy, but if end-users or end-user groups were to desire a shared security policy, we can envisage using e.g. the KeyNote trust management language such as as done by Ioannidis et al [8].

Another, more "overlaid" approach to the problems that Predicate Routing solves is Virtual Private Networks (VPNs). These may be constructed over IP by using tunnelling; i.e. encapsulating packets prior to routing them [5]. Using Predicate Routing, a VPN is defined simply as a set of predicates, obviating the need for tunneling. Isolation from other network users is achieved "for free", and changes in VPN topology are supported by the modification of Predicate Routing paths. Similar arguments apply to IEEE VLANs [7] in the local area.

[4]Each node in the network must be assigned a CLNP address, even if the network will only route IP traffic.

Predicate Routing also has much in common with the *hose model* [4] in that end-points are explicit (being described by predicates) while network paths are implicit.

The *network calculus* [2] provides a framework for reasoning about traffic on network links and has a natural synergy with Predicate Routing: predicate terms can be annotated with values from the network calculus.

XXX more on network calculus; policy routing/policy based routing.

6. CONCLUSION

We have presented Predicate Routing, a unified model of routing and firewalling in IP networks, and outlined both centralized and distributed implementations. Predicate Routing facilitates the *controlled networking* required to evolve the Internet toward a secure and robust infrastructure without the need for extensive protocol redesign. We are considering PlanetLab as a test bed for deployment of Predicate Routing.

Acknowledgments

Christos Tryfonas wrote the first implementation of Predicate Routing for the Sprint Labs cluster. We thank Bryan Lyles and Jon Crowcroft for their insights and discussions.

7. REFERENCES

[1] S. M. Bellovin. Distributed firewalls. *;login:*, pages 37–39, Nov. 1999.

[2] J. Y. L. Boudec and P. Thiran. *Network Calculus*. Springer Verlag LNCS 2050, June 2001.

[3] R. W. Callon. Use of OSI IS-IS for Routing in TCP-IP and Dual Environments. RFC 1195, December 1990.

[4] N. G. Duffield, P. Goyal, A. Greenberg, P. Mishra, K. K. Ramakrishnan, and J. E. V. der Merwe. A flexible model for resource management in virtual private networks. In *Proceedings of SIGCOMM*, volume 29 (4), pages 95–108, Sept. 1999.

[5] B. Gleeson, A. Lin, J. Heinanen, G. Armitage, and A. Malis. A framework for IP based virtual private networks. RFC 2764, Feb. 2000.

[6] P. Gupta and N. McKeown. Packet classification on multiple fields. In *Proceedings of SIGCOMM*, volume 29 (4), pages 147–160, Sept. 1999.

[7] IEEE. *IEEE Standards for Local and Metropolitan Area Networks: Virtual Bridged Local Area Networks (802.1Q)*, 1998.

[8] S. Ioannidis, A. D. Keromytis, S. M. Bellovin, and J. M. Smith. Implementing a Distributed Firewall. In *ACM Conference on Computer and Communications Security (CCS'00)*, pages 190–199, Nov. 2000.

[9] R. Mahajan, D. Wetherall, and T. Anderson. Understanding BGP Misconfiguration. In *Proceedings of SIGCOMM 2002*, pages 3–16, August 2002.

[10] T. Roscoe, R. Mortier, P. Jardetzky, and S. Hand. InfoSpect: Using a Logic Language for System Health Monitoring in Distributed Systems. In *Proceedings of the 2002 ACM SIGOPS European Workshop, Saint-Emilion, France.*, September 2002.