Technical Report

UCAM-CL-TR-898
ISSN 1476-2986

Number 898

Access control for network management

Dongting Yu

January 2017

This technical report is based on a dissertation submitted May 2016 by the author for the degree of Doctor of Philosophy to the University of Cambridge, Robinson College.

ISSN 1476-2986

Access control for network management

Dongting Yu

Summary

Network management inherently involves human input. From expressing business logic and network policy to the low level commands to networking devices, at least some tasks are done manually by operators. These tasks are a source of error whose consequences can be severe, since operators have high levels of access, and can bring down a whole network if they configure it improperly.

Software-Defined Networking (SDN) is a new network technology that brings even more flexibility (and risk) to network operations. Operators can now easily get third-party apps to run in their networks, or even host tenants and give them some control over their portion of the network. However security has not caught up, and it is easy for these third parties to access network resources without permission.

Access control is a mature concept; it has been studied for decades. In this dissertation I examine how access control can be used to solve the above network management problems. I look at the Border Gateway Protocol (BGP) from an operations perspective and propose a mandatory access control model using role-based policies to separate long-term invariants from day-to-day configurations. As a result the former would not be accidentally violated when configuring the latter, as this is a significant source of BGP misconfigurations today. Then, for SDN, I propose to add access control to controllers so that virtual controllers and applications that run within them cannot have unlimited access to the network infrastructure, as they do currently. Adding attribute-based access control makes the system much less fragile while it still retains the essential flexibility provided by SDN. Lastly, I propose a hierarchical architecture which, with SDN, can isolate security compromises even when some devices are physically compromised. This is achieved by using access control to both enable network access and deny unexpected connections.

Acknowledgements

I am very grateful to my supervisor Ross Anderson for his mentorship and support throughout my PhD programme. His insightful feedback has been greatly beneficial starting from my pre-application emails to dissertation writing. I have also been lucky to observe his 'sales techniques' and learn it to use as a general skill even outside of academia. I cannot ask for a better supervisor.

I thank Andrew W. Moore for being my second advisor and collaborator. His expertise in networking shows me the real life side of things and keeps my research relevant and useful. To my other collaborators, Chris Hall, L. Jean Camp and the rest of the group in the DARPA project, I am thankful for their help and input during the project.

I have been fortunate to be part of the Security Group, and thank the former and current members for creating this intellectual and fun environment. In particular, I thank Richard Clayton for lending his practical experience of operating a network and Robert N. M. Watson for being resourceful in getting me started with OpenFlow and SDN research; I also thank Joseph Bonneau, Omar Choudary, Richard Clayton, Markus Kuhn, Steven Murdoch, Frank Stajano, for thoughtful discussions about my research and about security in general, and Jonathan Anderson, Michael Bond, Wei Ming Khoo, David Llewellyn-Jones, Michael Roe, Laurent Simon, Rubin Xu, for further reading and commenting on my dissertation. Outside of the group, I thank Peter Neumann for hosting and mentoring me at SRI International, and Phillip Porras and Vinod Yegneswaran for valuable discussions while I was there; at the ISP I interned, I thank everyone who has made it possible.

I am also lucky to have made great friends during my stay here. Three became my house-mates: Fei Jin, Hao Wang, Rubin Xu (with whom I have also shared an office for four years), and to them and Deyi Hou I thank for the fun and for keeping me well. Hours away, I thank Yunfeng Zhang from Eugene, OR, for the social and emotional support, and Erin Sherry from Vancouver, BC, for keeping it real.

Finally, this PhD (and the two degrees before it) would not have been possible without the generous support from my parents, who have provided me with everything they can. I am very thankful to them for continuously encouraging me to lead my own path and to realise my dreams.

Contents

Chapter 1

Introduction

Networks connect machines together by carrying data from one node to another. To achieve this, each forwarding device in a network needs to have a sense of direction, or a map of the virtual world, in order to know where a piece of data should go. Generically, this process is called *routing*.

Today's networks use packet switching, a technology that supports large, dynamic and resilient networks. Each individual packet is a unit of data transmission; the routing and switching devices between the sender and the receiver make decisions on where to pass each packet. If a route is disrupted, packets can still arrive at their destination so long as it is reachable by a different path.

The part of a routing system that deals with packet forwarding is termed the *data plane*. It usually involves optimised hardware that can move packets at high speed from one interface to another. It looks up where a packet should go, either with an internal table or through other methods, and performs the low-level operations to send the packet out on the appropriate wire or fibre. But it is not involved in the process of building the internal table.

The logical plane above the data plane is the *control plane*. The control plane's role is to build, update, and maintain an internal lookup table which the data plane can consult, or to respond to the data plane if a packet does not match existing rules. In other words, the control plane determines data plane behaviour.

Networks need to be managed. New devices that join a network need to be set up, configured, and attached. The rest of the network may need to be updated to reflect changes. Devices or parts of a network are taken offline regularly for maintenance, or for updates to their software or hardware. For both technical and business reasons, network policies are updated regularly to direct traffic through a network on different routes. These management tasks can be automated with management software, or manual with direct human input. Monitoring tools also run within the network at this layer to provide real-time feedback. This part of the network is sometimes called the *management plane*.

Network protocols reside in the control plane, and are the focus of much security research. An update to a protocol, for example to patch a vulnerability, is a change in the control plane. The data plane is sometimes used, for example to observe the actual path that packets take, and whether this is consistent with the version presented by the control plane.

However, security researchers have not studied network management to the same depth. Protocols and networking devices can be made secure, but this is of limited use if operators cannot use them correctly. As operators define the network both directly (through manual input) and indirectly (through running scripts), we need to support secure behaviour on their part, too, by providing an environment in which serious errors are harder to make. If not, the management of a network can be the weakest link, subject to exploits and attacks.

As humans are involved in network operations, mistakes happen. Non-malicious accidents only differ from attacks in intent; both can easily put a network in the same undesirable state. The operators can be affected by a lack of experience, habit, working under a false impression, not receiving updates due to team miscommunication, and so on. As operators have full access to the network, mistakes can, and occasionally do, lead to catastrophic failures.

In the history of security research, many ideas and tools have been devised to help computer systems be resilient in the face or error and well as malice. One such is *access control*, where actions performed by subjects are checked for permission before execution. Access control policies are abstractions to make management easier and have be generalised from a variety of specific protection requirements. A typical problem is analysing a specific use case, working out which access control abstractions might help achieve the protection goal, and using it correctly (e.g., making sure it is not just relevant but also cannot be circumvented).

In this dissertation we show how access control can be used in novel ways for network management. We take two technologies, the Border Gateway Protocol (BGP) and Software-Defined Networking (SDN), and show how access control can help solve real problems when used in new ways.

In the BGP scenario, the biggest threat comes from operators themselves, as it is currently too easy to make mistakes and existing tools fail to limit their effect. Access control can be deployed as a helper, making sure operators cannot configure BGP into a state that violates a higher-priority invariant. This can help ensure that critical operational procedures are not affected by minor configuration changes, thus making the system less fragile.

With SDN, operators can allow tenants to be allocated some resource within the network, creating a virtual network over which they have some management authority. This brings added security risks as incentives may be misaligned among tenants and between tenants and the network owner. Access control can be deployed to enforce security policies and make sure each tenant stays within its allotted boundaries. We develop the SDN scenario further by proposing a new network topology for a cyber-physical use case and show how access control can be used within it.

1.1 Network routing

1.1.1 Inter-domain routing

The Internet evolved as a way to interconnect existing networks. Companies, universities, and other organisations built networks, took them online, and generally became domains. Hence the goal of inter-domain routing is to forward a packet to the larger domain to which it is destined, without worrying about the final few hops within the organisation. If we were to use the postal mail system as an analogy, inter-domain routing concerns how mail is forwarded to the correct country, passing through other countries in between.

The most straightforward, but naive, way to establish relationships among domains for inter-domain routing is to have pairwise connections between every two domains in the network. Of course this method does not scale; it does poorly at accommodating new domains and other changes. Better ways are taught in courses on distributed systems. Where we have decentralised domains with no unified command, desire not just connectivity but resilience (which means redundancy), a standard approach is to model the connections among domains as a graph, and develop algorithms for the routing logic to forward packets from the source to the destination.

Although in the past other inter-domain routing protocols have been discussed, such as Policy Routing by Clark [36], today's Internet uses BGP as its *de facto* inter-domain routing protocol. In BGP, domain entities are called Autonomous Systems (ASes), each having an Autonomous System Number (ASN) as its identifier. BGP specifies the logic to be used, along with sufficient freedom in defining a routing policy between two connecting ASes. The rest of this section describes ASes and BGP in their current form.

Autonomous system (AS)

An AS is generally a single administrative organisation with a consistent routing policy existing as a member in the BGP ecosystem. Each AS has an alphabetical name and, more importantly, an assigned AS Number which is its unique identifier within BGP. It is a logical unit independent of organisation size or location. In fact, it is possible for an AS to span a vast geographical area. It is also not necessary for each organisation to have only one ASN. It is not uncommon to see multiple ASNs existing in parallel for the same company, say after a corporate acquisition or a merger. Many different types of organisations have their own ASNs, including large corporations (IBM, Google), universities and groups of them (JANET, University of Oregon), Internet Service Providers (BT, AT&T), and traffic-related companies (Akamai, Level 3 Communications). It is also possible to get an Internet connection without one's own ASN; many small and medium companies do not run their own BGP networks but buy connectivity from one or more ISPs.

ASes connect to whomever they need to. Usually, they connect with their local counterparts (peers), umbrella organisations to which they belong, specialised network carrier companies, and others that they feel make performance or business sense. Larger ASes that are ISPs also connect with their clients, including those that participate in the BGP system with their own ASN [58].

The interconnection graph is clustered around a few core ASes, and spreads to the edge to stub ASes. The most connected ASes are *tier-1* networks; followed by tier-2 and tier-3 networks, and finally the edge networks. However, the graph is not a tree. For cost, performance, and resilience reasons, many middle-tier and edge networks also connect with peers of similar size or pay to be clients of multiple upstream providers.

The BGP graph topology is constantly changing. Established links disconnect and recover all the time, new ASes join and leave on a regular basis, and new connections are made among ASes as business relationships are formed. It is impossible to present a topology graph of the Internet, but only an estimate at some particular time. The difficulty is exacerbated by secret business agreements that result in hidden links that are only activated when needed or when certain, undisclosed, conditions are met.

Border Gateway Protocol (BGP)

BGP is a protocol among ASes to help each establish a local view of the Internet. Briefly, BGP allows each AS to keep its own version of a IP-prefix to next-hop-AS table, along with the full AS-path for each IP prefix. If an AS receives a message from a neighbour, indicating a path through this neighbour, and if this new path is 'better' than the previously-selected path for this particular IP prefix, then the local AS selects this path as the new best, and announces this decision to all of its neighbours, who can then go through the same process, each comparing with its previous version to decide on its own optimal path.

The notion of 'better' in path comparison is up to each individual AS. BGP provides a guide and BGP software provides the commonly needed rules, but the ultimate decision lies in the AS. Different attributes that can be compared include AS hop length, local preference, multiple exit discriminator, AS-path, and next hop. The order in which these attributes are compared is the AS's decision, and usually the local preference is the most influential.

BGP constructs and works within the control plane; it does not care about the data plane. The separation of the control and data planes keeps network maintenance simpler and allows the control plane to take its time to calculate new routes without affecting data forwarding. Similarly, because BGP works in the control plane, it is unaware of data plane attributes, such as latency and bandwidth capacity, unless manually converted into a weight and reflected in the local preference values of the BGP route decision process. However these two planes sometimes share the same wire, as BGP packets are communicated in-band.

1.1.2 Intra-domain routing

The complement of inter-domain routing is intra-domain routing, which deals with the delivery of packets within an organisation. In the postal example above, the equivalent of intra-domain routing would be the movement of mail from customs to its final address.

One advantage in intra-domain routing is the availability of link information. Within a domain, there can be a lot of sensitive information, and firms want to reduce the information available to others. Between domains this can lead to suboptimal routing decisions, but intra-domain, routing everyone falls under one organisation, so this disadvantage is minimised. Another advantage is that intra-domain routing does not need to make scalability a main requirement.

Popular intra-domain routing algorithms include Open Shortest Path First (OSPF), Intermediate System to Intermediate System (IS-IS), and Interior Border Gateway Protocol (iBGP). These fall into two categories: link state routing algorithms and vector distance routing algorithms. The details of these algorithms and their differences are not discussed here, as they are irrelevant to the rest of this dissertation.

1.1.3 Software-Defined Networking

Software-Defined Networking (SDN) is a recent innovation in networking and grew out of earlier research into active networks [50, 133]. It is more a concept and approach rather than a single set of protocols. SDN is not just about routing, but is also about network management. Eventually it may blur the lines dividing network layers, enable different protocols to work together, and even shift or erase the boundary between inter-domain and intra-domain networking.

The essence of SDN is that forwarding is done by simple but fast hardware, while the control logic is offloaded to commodity PCs [115]. Traditionally, router vendors such as Cisco and Juniper provide both the hardware and software implementations of needed protocols to the end user, without much room for customisation other than by scripting languages. SDN-enabled hardware is much simpler: it does not come with a set of implemented protocols but rather a set of primitive interfaces that allow an external computing unit, the SDN controller, to control the routing behaviour and gather hardware statistics. As a result, SDN can be made much cheaper with simple hardware and a commodity PC as the controller. It can also be much more flexible. An AS can implement a custom intra-domain protocol all by itself, or use a custom protocol for inter-domain communication with networks with-which it peers [23, 123]. It is currently gaining traction in data centres and Internet exchange points due to the freedom it offers and the capital cost it saves [89].

1.1.4 Network management

Human operators currently manage networks. Often they are generalists: the same operator works on all aspects of a network, including inter- and intra-domain routing. It is also not uncommon to use the same set of tools and management interface to manage both. Sometimes the same devices are used for traffic of both types, and running multiple protocols at the same time, especially for devices at the edge of a network.

As networking moves from vendor-supplied software embedded in routers to SDN, it will facilitate innovation, hopefully leading to more and better tools for operators. Ideally, humans would do what we do best by laying out high-level intentions, whose descriptions are transformed into lower-level commands and fed into network devices. The legacy router architectures have prevented the last step being fully automated; and sometimes it is still a pure manual process, leaving lots of room for error.

1.2 Principles of routing security

Routing security is under increasing scrutiny. There are many potentially severe consequences of errors and attacks, including packet misdirections, interception, or even making oneself disappear from the Internet altogether. The stakes are both financial and strategic, so many players from black-hat hackers to national intelligence agencies have an active interest in this area.

In this section we do not attempt to discuss security solutions or their effectiveness, but rather lay out an analysis of what is possible and how much has been studied. The importance of such an analysis is that many different protocols and systems share very similar routing security threats, but defences may need to be adapted to each particular case. It is also important because with a better understanding of the problems, many threats can be eliminated at the initial design stage rather than patched afterwards. Because routing is such a vast topic and not everything can be covered, we try to present the commonalities and stay protocol-agnostic where possible.

We divide the analysis into three parts, each trying to answer one question, in order:

1. Where are the vulnerabilities? (in Section 1.2.1)

2. What is being attacked? (in Section 1.2.2)

3. What is at stake? (in Section 1.2.3)

1.2.1 Threat model

We first take a look at the various threats to routing. Through this analysis we can systematically identify *where the vulnerabilities are*. We identify five categories: hardware, software, protocol, operator, and tenant.

Hardware threats

Hardware threats are those due to the behaviour of networking equipment and vendor-supplied code that is not modifiable by the user. There are several attack vectors. Most commonly, the hardware vendor includes buggy code in its products. When discovered, it is expensive to upgrade a piece of hardware as many operators need to ensure correct working. Often, hardware or firmware upgrades are delayed or skipped altogether.

The hardware is thus open to external attacks, either because of unpatched vulnerabilities or because it is out of date. The variety of brands and models is limited so any single vulnerability can often be used in a large number of devices. Shared (and pirated) code means that an attack discovered on one router may work against its competitors too.

The recent disclosures by Edward Snowden reveal that the NSA conducts supply-chain attacks, opening router hardware in transit from the factory to customers and adding trojan devices. No doubt other agencies do the same.

Software threats

Next there are software threats, which arise from code running on general purpose computers. The individual consequences of each vulnerability may not be as serious as in the case of infrastructure hardware, but the user base and the attack surface may be greater.

One major recent example is that of the Heartbleed Bug[1], where, despite urgent security warnings and a strong push for everyone to update, servers were still attacked in the period after public disclosure. In addition, Privacy International has documented the widespread sale to repressive regimes of systems that can install malware on thousands of citizens' PCs for the purpose of building a covert infrastructure for monitoring network traffic.

[1]https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0160

Protocol threats

We see various threats from the protocol: bad protocol design, faulty implementation, inappropriate usage, and key-related issues.

Protocol design has always been hard. Even very small design decisions can lead to unexpected results under rare but plausible conditions. Protocols can also be used for a different purpose than originally intended. DNS, for example, started as a lightweight protocol mapping domain names to IP addresses. It has been upgraded to DNSSEC while much of the original intent remains. However there are now proposals to tie DNSSEC with TLS certificates in web browsing[2], which may require a new analysis of DNSSEC, simply because it was not designed with this 'leverage' in mind.

Even with a good initial protocol design, specifications get longer as the complexity of the protocol increases with time; and since the specification is often done in plain English (for example in IETF, IEEE, and ISO standards), inconsistencies creep in due to human errors, lack of formal rigour, and indeed the ambiguities in human language. Specifications are incomplete, and implementation details result in incompatibilities or insecure behaviour. Getting memory management and timing issues correct while threading the underlying protocol together is no easy task.

Next we have the user of the protocol. As we have seen from the classic experiment [178] on using PGP, security usability is hard. So it should surprise no-one that another source of vulnerabilities is key management. It is not straightforward to get key material to routers in remote locations, particularly if they are supplied by, maintained by, and hosted by foreign companies that are not entirely trusted, and that may be subject to coercion by local law enforcement.

Operator threats

Operators of a network also pose interesting threats — to their own and neighbouring networks. Being a conservative crowd, many network operators are reluctant to change a part of the network if it is believed to work correctly. We have discussed delaying or foregoing firmware updates as a possible security threat, and this is partly due to operator practices. Others include improper use of network devices and careless configurations.

A good practice, followed by larger networks, is to automate the device and network configuration process as much as possible, including plans for rollback should something go wrong. Unfortunately, in many networks operators use the command line interface (CLI) to issue commands directly into the live network. This is slow and prone to inconsistencies and errors such as 'fat fingering' typos. Using the CLI also leaves a poor record of work

[2]https://datatracker.ietf.org/wg/dane/charter/

carried out, if any record is kept at all. Too much network state knowledge is in the hands (and heads) of too few operators.

Another way operators can pose a threat to the network is through improper knowledge of the hardware and its included software. For example, competing hardware may have different CLI commands and arguments. When more than one brand exists in the same network, operators may get confused and issue the wrong command. There are also default services that run on many network devices, such as FTP servers, of which the operator may not be aware. Leaving them on by default can create major vulnerabilities.

Tenant and user threats

Every network serves one or more tenants, or users. At one extreme, a data centre network may have only one tenant, and the tenant and the owner may be the same firm. At the other, an Internet service provider (ISP) sells connectivity to millions of clients, all of which are users to the its network. There are also different levels of participation by tenants. If a downstream client simply wants Internet connectivity, as with domestic customers, then the tenant does not require much more from the ISP than reachability.

In a data centre, the tenant may want to be more involved. Network optimisation can be done both within the data centre and with links connecting to the outside. The tenant may also want to do load balancing or monitoring within the network. These actions all place some level of control in the hands of the tenant.

Threats are particularly likely to develop where there are misaligned incentives. If tenants and the owner are separate entities, or even different departments within a larger company, their incentives may be different. Tenants' attitudes towards other tenants may also be hostile, and competition for resources can occur if the owner operator is not careful. For example, a tenant may want multiple redundancy within the network at the cost of other tenants, or it may want to cheat on traffic volume to cut its bills. If a tenant discovers that another tenant within the same network is a competitor, it may even try to discreetly launch an attack.

So while the introduction of SDN makes networking more flexible, it also makes it more hazardous. Tenants will be offered more freedom to do both good and ill. It will therefore be necessary to improve protection. What might this consist of?

1.2.2 Attack surfaces

A second part of routing security is the logical surface that an attack affects — technically, *what can be attacked.* In network routing, there are two main attack surfaces: the control plane and the data plane. The control plane calculates where a particular packet should be forwarded. The data plane does the actual packet handling by extracting the necessary information and then performing the forwarding. These two planes are sometimes tightly coupled and sometimes reside on physically separate machines, but a distinction remains.

Control plane

The control plane, being the brain of a network, is the target for many attacks. In BGP, the protocol provides the necessary framework for information hiding as well as information sharing. ASes share just enough information to achieve connectivity and fulfil business contracts, but no more. As a result, it is possible to lie in BGP. Sometimes such lies are needed for operational purposes, but it is also possible for false information to be believed and passed on either intentionally or by mistake.

In April 1997, AS7007 claimed to be the source of all Internet prefixes, then proceeded to deaggregate the table into /24 prefixes — more than 73 000 of them — and announced them to the core Internet. The hours that ensued were filled with non-reachability, disconnections, and confusion in America and the rest of the world. The problem even persisted past a total shutdown of AS7007's routers, as caches in other ASes were not cleared soon enough. Even though this was an accident, a great part of the Internet was disrupted by one single AS. The carrier and target of this disruption was the BGP control plane.

More generally, an AS can attack the BGP control plane by announcing that it has a more preferred path than those known by its neighbours. The preference can come in the form of a shorter AS-path length, more specific prefix, or better business relationships (revenue generating neighbours are preferred). The specifics of who can launch such an attack and how are discussed in Section 1.2.1.

Many similar incidents have taken place since 1997, most accidental, some suspicious, and a few even without the true identity of the originator being known. Until we have a secure BGP, or at least a wide adoption of better operational practices, such attacks on BGP will continue to be easy.

Intra-domain routing suffers from control plane attacks, too. A single compromised router is often sufficient to fool the rest of the network. The lack of malicious cases in real life is probably due to staff discipline and well aligned incentives generally. Of course, any accidental cases will be seen as simple errors and will probably not be publicised.

Data plane

The data plane deals with the actual forwarding of the packets. With more hardware and less code, it is probably harder to attack.

Yet a successful data plane attack could be hard to detect. Control plane logic leaves a trail that other devices can record for later auditing; the data plane, on the other hand, does instantaneous action and can be changed at a moment's notice. An analogy is highway traffic: it is easy to install speed limit signs and check that they are installed correctly, but much harder to make sure that every car respects the limit. The usual method is to check for data plane misbehaviour through inserting check packets into the live network. Even this method cannot cover all cases, such as if only specific types of packets are targeted.

Apart from persistent attacks, a data plane device can be triggered into malicious behaviour by a sequence of pre-determined packets. One example is 'port knocking', where the device sees (likely failed) connections made into a sequence of specific ports. It is difficult for the network operator to detect such signatures without prior knowledge, since the traffic can come from anywhere, and the attack signature can take any form.

Once in malicious mode, a switch can do lots of things: mirror traffic to a different interface, silently insert or drop particular packets, or misreport statistics. Because statistics are reported from the device itself, malicious behaviours can be stealthy. A better way would be for devices to report each other's statistics, and it is even more effective if they come from different vendors; but this can be more computationally expensive and harder to coordinate.

With the development of SDN we are seeing the data plane device moving in two different directions: in some setups the data plane becomes even more separated from the control plane, adhering to the SDN mentality and offloading computations to separate controllers; another direction is to have a lightweight controller next to the forwarding device for performance and resilience reasons, which in effect makes the data plane even smarter. In the latter case, the data plane can be better protected against malicious behaviour by a custom detection module, but also opened up to easier attacks via vulnerabilities in the controller.

1.2.3 Security principles

The third part of routing security is security principles. In this part we can analyse each component that can fail as a result of an attack, and answer *what the consequences are*. Four components are at stake: confidentiality, integrity, availability, and accountability.

Confidentiality

Confidentiality in routing mostly refers to control messages. Protecting the data plane is possible, but it is generally done in thigher layers. In the intra-domain case, operators may feel the need to protect the specifics of routins, especiallywhere the control plane extends into uncontrolled areas such as shared sites. Then control plane data could potentially reveal sensitive business relationships. For example, a backup path may exist which does not normally carry traffic and is thus not normally observable to competitors.

In inter-domain routing it is much harder to achieve confidentiality. Once a path is announced to a neighbour, it becomes the neighbour's decision to announce it further. Special filters can prevent private peerings from being propagated, but this is only realistically effective within a one-hop distance. As for BGP packets between two neighbouring ASes, there is less incentive to protect confidentiality because the packets only travel on the wire between them. The adversary for physical wiretapping is usually government agencies, who are not normally intersted in private commercial agreements.

Integrity

Compared to confidentiality, integrity receives much more attention, especially in inter-domain routing. Ity refers to the correctness of control plane packets upon arrival and further distribution. In intra-domain routing, the assumed (and most of time achievable) physical security and ownership of network devices ensures integrity of the content, origin and routing of the packets.

The main concern, then, is integrity of control plane information in inter-domain routing. In BGP, the main piece of information in an announcement is essentially the AS-path. BGP as it is currently used affords some optional checks to verify the integrity of BGP TCP packets over the wire (which we will discuss later), but no integrity at all on the content origin: the neighbouring AS can make up a completely fake AS-path and send it as if from a valid source. This type of attack is the main reason that BGP needs to be secured. It is also arguably one of the weakest links in the Internet infrastructure. It has been known for decades but it is only recently that we have seen work on realistic solutions.

Availability

At the heart of routing lies availability, as the exchange of routing information is precisely to assure availability. Service disruptions can occur in both inter-domain and intra-domain routing, but, as is the case with other security principles, it is less of a problem in the intra-domain case due to the protected environment and stronger local trust. However, there is still little room for error. For example, a single well-placed misbehaving switch can still bring down an MPLS network.

In inter-domain routing, the issue becomes more complex. Every AS wants to be reachable by others and wants to reach as many as possible. Multi-homing is common to reduce the possible points of failure. Yet it is still common to hear of routing incidents where one AS causes the temporary disappearance of another. The offending AS can accidentally (through a software bug or operator error) leak a list of deaggregated, more specific prefixes that it does not own. The decoupled control and data planes and the lack of active cross-referencing between the two means that such errors are not easy to spot locally; operators may remain unaware until others report an anomaly.

Accountability

Acountability is a further property of interest to self-regulating inter-domain routing. Suppose it is possible to find out the originator of a routing incident; then it is possible to assign a lower priority to that AS in future route selections. This idea is the foundation for reputation-based routing proposals [30, 185] which can incentivise ASes to be more careful in what they announce and thus make the control plane less error-prone. A negative reputation can mean an AS having fewer ASes that want to peer, forcing it to pay more for transit.

However, it is not so easy. We have seen bad updates from a third-party while passing a route that is not its own. Without proper bookkeeping of all intermediate BGP updates (and the few looking glasses and route view servers are not sufficient), it is hard to know which AS introduced an anomaly. Post-mortem analysis is not always possible and neither is a reproduction of the error. Often ASes do not cooperate even if they are not to blame. The design of BGP makes accountability hard, and the social dynamic of the AS community prevents collaborative debugging.

1.3 Access control policies

Access control became necessary in the 1960s with the advent of time-sharing computer systems where multiple users shared the same system concurrently, rather than sequentially. Even in cooperative environments this led to a need for separation and confinement so that users (and the bugs in the software they wrote) would not inadvertently interfere with each other; classic papers include those of Lampson [106] and Saltzer and Schroeder [149]. Early stateless access control systems typically had an access control matrix of which user's processes could access which resource; this could be stored in columns as access control lists (ACLs), or in rows as capabilities. To begin with, users could generally choose which files they wished to share, but this *discretionary access control* (DAC) philosophy was supplanted in government systems with *mandatory access control* (MAC), based initially on the work of Bell and LaPadula, to characterise systems that prevent information leaking down from higher to lower levels of classification — and do so independently of user actions [12]. The history and early evolution of mandatory access control systems are described in [4].

The Bell–LaPadula model is not the only approach to MAC, although it led to the Orange Book and the Trusted Network Interpretation [85] — the first attempt to impose MAC on networks. This seeks to ensure that data can flow from a network to other networks at a similar or higher level of classification. Where the control objective is integrity rather than confidentiality, the relevant policy may be Biba [19], which is designed so that subjects at a given integrity level may not corrupt objects at a higher level. Such models are used in industrial control systems where the highest 'safety integrity level' is the safety system whose job is to shut down the plant if its operating parameters are exceeded. This can report to, but must not be corrupted by, the primary control system, which in turn feeds information to management systems but must resist any tamper attempts from these systems. Industrial control systems are another potential application area of SDN and the protection goal is often robust reperimeterisation.

More general models appeared starting in 1990s, with the *role-based access control* (RBAC) model adding a layer of abstraction by assigning users to roles. This reflected real-world practice in organisations where the great majority of individuals occupied one of a small number of clearly defined roles such as (to take banking as an example) 'teller', 'assistant accountant' or 'branch manager'. Each of these roles has clearly defined powers and responsibilities so it is natural to separate user-role management from role-object management. This eventually became a NIST standard [150]. While FreeBSD ships with support for the two standard multilevel MAC systems, Bell–LaPadula and Biba, SELinux supports both of these and RBAC too.

Recently, RBAC was extended to include arbitrary attributes that are then evaluated to sets of permissions, unlike RBAC roles which directly represent such sets. The new model is *attribute-based access control* (ABAC) [81]. ABAC lets us incorporate environmental information such as time; thus a bank teller may be given access to the building only from 7am to 6pm. We noted that time-dependent network access may be a requirement in multi-tenanted networks.

1.4 BGP security

Two important works identify BGP vulnerabilities and possible attacks [128, 22]. From these, the consensus is that BGP offers little protection in guaranteeing several important security properties such as the confidentiality, integrity, and authenticity of announcements and paths, and availability of BGP routers. This lack of protection can lead to various exploits including replay, man-in-the-middle, cut-off (exclusion), and denial of service [132]. It can also result in the most serious form of attack, prefix hijack [59]. A prefix hijack, interchangeably referred to as a route leak in this dissertation, occurs when an AS announces to its peers either that it owns a certain prefix or that it knows a very short path to that prefix (the latter case is sometimes also referred to as prefix interception). The neighbours of this AS, without the ability to check the validity of these claims, can only believe the announcing AS, and spread the information even further, by passing it on to their peers. As a result, those ASes who believe these path announcements, and use these paths to route packets, will route packets to an unintended destination, with the actual owner not receiving any traffic. It is even possible for an AS to hijack all traffic destined to a particular AS, leaving the hijacked AS unreachable from the rest of the Internet, including emails that try to warn this AS of the hijack.

On the more practical side, there has been work measuring and analysing BGP updates during normal activities [51, 113, 148, 175, 181]. Many more works focus on BGP behaviour during special periods, usually abnormal events, such as prefix hijacks [9, 76, 188, 189], multiple origin AS conflicts [190], unallocated prefix blocks [48], natural disasters [180], and observing from just one vantage point [134]. Due to the distributed nature of the Internet and the limited visibility which any BGP study can get, it is important not to infer too much into the unknown. In this aspect, Roughan et al. offer some practical suggestions on measuring BGP properly [147], and Bush et al. on how to design better experiments [21]. Others have also offered observations and explanations of specific BGP phenomena, for example IP block deaggregation [35] and BGP table transfers [31]. A few works try to understand AS policies and their goals, as they play a direct role in the topology of the Internet, and is a prime source for error [24, 61, 104, 174]. Mahajan et al. try to understand policies with a focus on misconfiguration [120]. With BGP incidents being a real threat, detecting such incidents has been a major focus in the history of BGP research. Several proposals detect general routing instability [52, 57, 68, 82, 179], and others are designed specifically for detecting prefix hijack events [99, 105, 118, 158]. Nagaraja et al. propose a way for ISPs to safely share their data for anomaly detection while preserving privacy [129]. At a higher level, other empirical works in this area include explanations for multi-homing and consequences [37], examining the routing market [170], and IPv6 adoption in the Internet [40], which can potentially generate many more prefix blocks for BGP to handle.

1.4.1 Replacing or securing BGP

Proposed replacements for BGP have seen a long but unfruitful history [131, 62, 83]. Proposals to secure BGP in the past decade include Secure BGP (S-BGP) [95, 96], Secure Origin BGP (soBGP) [177], and Interdomain Route Validation (IRV) [63]. Other approaches [2, 87, 88, 91, 92] have also been proposed to improve or modify these major proposals based on criteria such as complexity or overhead, highlighting the tension between security versus deployability. Partial deployment scenarios have also been analysed, since that is a practical requirement [29, 60, 119].

Growing recent concern about the resilience and security of BGP [45, 69] led to the IETF's SIDR working group, which has been working on making BGPSEC a standard. BGPSEC secures the AS-path attribute by including signatures at each AS with each hop layering everything before it. RPKI, a prerequisite public key infrastructure that provides AS-number and prefix block ownership mappings along with cryptography attestations, is already a standard [108] as a result of the SIDR effort. BGPSEC is still in its draft stage [109, 110], but will be finalised shortly. While operational issues are considered to some extent in its threat modelling [97] and security requirements [14], it is out of the drafts' scope to consider existing operational issues, particularly usability. Already, BGPSEC and its related components are under scrutiny [73, 137]. RPKI has been examined in the presence of malicious parties [20, 41], as well as in an Internet exchange point (IXP) scenario [7]. Osterweil et al. have provided a scaling estimate for RPKI [135], while Wählisch et al. share their experience on using RPKI [173].

1.5 SDN security

SDN moves us from the problem of coordinating multiple policy holders to a single policy holder driving multiple policy implementations. It emerged and evolved from earlier work on active networks [168], in response to demand for complexity management, and to meet the need for cost reduction [50]. A significant enabler was OpenFlow [123], an openly-available standard control-plane data-plane protocol that facilitated wide adoption by being independent of any particular vendor. SDN's flexibility has led to wide adoption in data centres [89] and IXPs [65]. Applying ideas from earlier control-data plane separation [171], FlowVisor [157] supports the slicing of a physical network and presentation of a subset of the network to each controller at switch-port and switch granularity. A further virtualisation approach is to consider the networking logic of such a subset as a single big switch; subordinate (tenant) users may then describe policy based on the overall input and output of the network, rather than at the level of individual physical switches.

However, many challenges remain in SDN, such as scalability [27, 154] and security [103, 153]. We group previous research into three components (data plane, control plane, and management), depending on where security takes place.

Securing SDN from the data plane has revolved around monitoring and policy enforcement at the packet level. We see an effort to dynamically analyse OpenFlow application code and to identify bugs using symbolic execution in [25], with an earlier work that applies model-checking on the state space of the SDN system [26]. VeriFlow sits between controllers and forwarding engines to check for violations in network-wide invariants, as the rules are inserted [100]. FlowSense takes a different approach, and lets switches push network performance numbers upwards to the controller, in order to monitor and measure the network [183]. Scaling horizontally, Yu et al. propose collaborative monitoring across the network [186]. Middleboxes are also considered, allowing them to join the policy decision process [47, 142]. Some networking-focused research tackles particular issues, such as scalability [166], reachability failures [94, 6], and performing a traceroute through an SDN [1]. From the outside, Shin has proposed a scanner that can identify if a network is using SDN, allowing further SDN-specific attacks to be launched [159].

Control-plane security research has mostly focused on the controller, since that is where novel ideas can flourish. A few controller implementations have been engineered [18, 101], including specialised controllers such as for inter-cloud connections [124], for interacting with BGP networks [102, 145], for detecting and resolving security policies [140], and for IXPs [167]. In fact, the concept of a software-defined exchange [66, 65] and has even seen a real-world trial deployment at the Wellington Internet Exchange in New Zealand [49]. Other works discuss specific issues of controllers, such as their placement in a network [74, 112, 169], addressing them as resources in the network [8, 10], enforcing policies [53, 172], abstracting the routing platform as one logical unit [90], and security permissions for applications [176]. Outside of the controller, FRESCO makes it simple to develop a security application to detect and mitigate threats within the network [160].

Some researchers are working on applying language-based abstractions to achieve stronger guarantees in security and policy enforcement. There is FSL [75], a policy language that lets operators write policies in a enforceable way. Another work uses a programming abstraction to define and isolate network slices [67]. Frenetic is also an ongoing project that aims to bring an abstraction to network programming, especially in querying for state, defining policies, and updating network configurations [55]. In updating network configurations, we have also seen a work that abstracts away potential inconsistencies during a network-wide configuration update [143].

Managing an SDN, however, has not received the attention it deserves. Panopticon discusses potential issues in a partially deployed SDN, a scenario that will be met for any traditional networks wishing to upgrade [111]. Zhu et al. propose a service-oriented architecture to enable network applications scale horizontally [191]. Galis et al. discuss the evolution of network applications within an SDN and how to perform them smoothly [56]. One particularly interesting proposed controller, Fleet, specifically sets out to limit the damage that can be done by malicious network administrators [121]. My own research has also been about security needs that arise from managing the network: we propose a hierarchical network architecture to minimise the attack surface in a cyber-physical environment such as Heathrow airport [184], and bridging SDN into inter-domain networking by modifying Quagga so it can collaborate with the BGP side [70].

Network virtualisation in SDNs are cross-plane technologies and have also been a topic of research. Perhaps the most important software that results is Open vSwitch [138]. A different direction is taken by Network Function Virtualisation (NFV) paradigm, which is less powerful in flexibility but more compatible with existing hardware [11]. Programming the network in the presence of virtualisation has also been discussed [3, 39]. Tools using virtualisation to support networking research have also been developed [71], with Mininet being the most mature and widely used [5, 107].

1.5.1 OpenFlow

OpenFlow (OF) is a protocol used in SDN and, being the first, is the most mature. The original paper [123] describes the concept and now the design and improvement of the OF specification was absorbed by the Open Networking Foundation as its mandate[3]. Since then, much of the research in OF and SDN is general enough to be categorised as SDN research, and is reviewed below, but some is specific to OF and will be discussed here. The first attempt at using OF for a novel technology is FlowVisor [156, 157], which partitions the network into slices with each application only able to see their own virtual slice. Others have also focused on specific aspects of using OF, as seen in FleXam [162], which enables better sampling, and research by Liu et al. [116], which improves on prefix caching. Lu et al. [117] and Shelly et al. [155] have proposed ways to improve cache flows in an OF network. In terms of security, we have seen Flover, which checks for security invariants in a system [165]. Closer to the topic of this dissertation, OFLOPS provides a framework to evaluate how well a switch handles OF messages [146], and Fernandez compares reactive and proactive OF rules and particularly how well they can scale [54].

1.6 Structure of this dissertation

In this dissertation we explore a number of networking scenarios and how access control can help. These scenarios range from operators that can easily make errors during regular network management, to competing tenants with incentives to do more than it is permitted, and how access control can be used to provide security in new deployment scenarios.

In summary, the scenarios and security focus increase in complexity. At the same time, the protection offered by our contribution also increases. We start by addressing accidental errors as the main security problem, and exclude intentional malicious attacks. This first scenario is an operational one that can affect a network's resilience. We explore how access control can be used to help, rather than restrict, its users. In the second scenario we introduce a more general software environment where participants can cause or allow malicious attacks to occur. Here the security focus is on the architecture and implementation of SDN controllers, and access control is used to enforce information flow to and from participants and isolate unauthorised access. The focus and scope is within one instance of the physical controller. We take the complexity further in the third scenario, by constructing a 'system of systems' built upon and scaled from the second scenario. Here we discuss how a set of controllers can interact among each other and stay secure in a real-life cyber-physical environment. Further, we include an additional security assumption of possible physical compromise. As protection, access control plays a more involved role in this system: it is no longer a feature in the software, but becomes

[3]https://www.opennetworking.org

a fundamental part of the communication channel, by acting as an ignition key that is always required.

However, there are some similarities among these scenarios as well. Access control as a security policy is the major common theme in each scenario. While the first uses role-based access control and the second uses an attribute-based one, ABAC actually evolved from RBAC and is a superset of RBAC. The same ABAC core mechanism is again used in the third scenario, with capability tokens to help performance.

Traditionally the control and management planes are administered from within one single logical space. Separating them into smaller units is another commonality throughout this dissertation. Instead of providing full network access each time networks are configured, we allow only a subset of the privileges according to contextual information. By imposing this limitation we can reduce careless errors and block unauthorised use.

More specifically, we first discuss BGP's operational security as observed in the real world in Chapter 2. We discuss many ad hoc security measures that operators use actively, but highlight the core problem, which is that long-term invariants are being managed within the same space as day-to-day tasks, and errors within the latter can affect the former. Operators are under-supported in their tools and so have to rely on 1970s technologies while working with networks some of whose downtimes can be visible nationwide. We propose to add mandatory access control into BGP management software and show through our implementation that it can complement operators to prevent accidents. This chapter was based in large-part on discussions with various network engineers enabled during an internship at an ISP. Ross Anderson also provided input in the research of this work.

Then, we shift our focus to SDN in Chapter 3, where we present AXE, an attribute-based access control system for SDN controllers and virtual controllers to manage tenants and apps. We realise that having stateless access control as a reference monitor in SDN controllers and virtual controllers has the same effect as stateless firewalls. This observation allows us to combine the vast existing research literature on access control and firewalls so we can draw parallels. The work in this chapter is done in collaboration with Andrew W. Moore and Ross Anderson.

We scale out in Chapter 4 and look at the bigger picture of networking including flexibility and management. As new networking environments take advantage of SDN's added flexibility, we examine how to secure them. We take cyber-physical systems as our use case and propose a hierarchical architecture that can cope with physical compromise. We improve on AXE to secure this topology so that a networking device has less chance of being affected (and therefore propagating bad states) even when its neighbours are compromised. Here, access control mechanisms give a clear added benefit. The work in this chapter is done as part of a DARPA seedling project, in collaboration with Andrew W. Moore, Chris Hall, and Ross Anderson, and based on our publication [184].

Finally, we conclude with Chapter 5 by summarising contributions of this dissertation and discussing future work.

Chapter 2

BGP operations security

The Border Gateway Protocol contains little in the way of security mechanisms, relying on ASes to trust each other. This was sufficient in the early days of the Internet, when there were merely a handful of networks. Today, with over 53 000 ASes worldwide, and with ASes owned by governments and companies in countries that are unfriendly towards each other, the trust assumptions of the initial design no longer hold.

BGP attacks were first studied in 1989 [15]. There has been intermittent effort invested in adding security mechanisms to BGP since then. The IETF now has a secure interdomain routing group working on BGPSEC, a protocol enhancement that will enable route announcements to be signed, and which we will discuss in more detail later. It enables relying parties to check that claims made about specific IP addresses are made by principals who control signing keys registered to the organisations to whom those IP addresses have been allocated. While improved BGP proposals are certainly welcome, there is room for doubt about the incentives that ASes will have to deploy and operate BGPSEC, and about the proportion of the currently experienced problems that it will solve. For these reasons, the current version of BGP deserves to be studied for the incidence of anomalies, malicious or accidental.

We decided to look at what goes wrong in a real BGP network, and how incidents are initiated, propagated, and resolved. We are also interested in what can be done to fix these problems in a realistic way considering current practices, which can complement long-term security improvements such as BGPSEC.

In Section 2.1 we first describe an example BGP network to set the context for the rest of the chapter. We then describe how operational anomalies occur in Section 2.2, and note other features of BGP that are found in real life rather than protocol in Section 2.3. In Section 2.4 we present some observations from operating a medium-sized AS and discussions on operators mailing lists, including current ways operators deal with potential anomalies. To tackle shortcomings of BGP control software, we implemented and evaluated a mandatory access control in Quagga in Section 2.5. Finally, we discuss the likely upgrade of BGP, BGPSEC, and how it impacts operations in Section 2.6, and conclude in Section 2.7.

2.1 An example BGP network

We first briefly illustrate how BGP works using the simple example network shown in Figure 2.1. There are four ASes in this network, labelled A, B, C, D, each announcing one prefix to the world, as shown. The links are labelled a, b, c, d, e. We assume a vanilla BGP with no local preference or special policies, which leaves us with the number AS hops as the first metric to determine precedence.

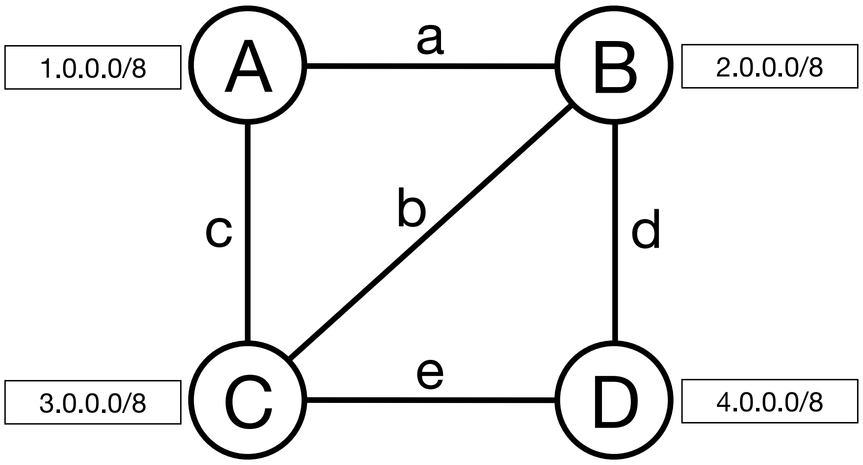

Figure 2.1: A simple Internet with ASes A, B, C, D and links a, b, c, d, e. The prefixes that each AS announces to the world are labelled beside each AS.

At equilibrium, each AS should see all prefixes with each seeing a different path. There are three logical tables that each AS keeps, the Adjacent Routing Information Base, Incoming and Outgoing (*Adj-RIB-In* and *Adj-RIB-Out*), and a Local RIB (*Loc-RIB*). Each AS keeps all routes that have been received from its neighbours in the Adj-RIB-In, chooses one best route for each prefix and keeps it in the Loc-RIB, sends out prefix reachability information to its neighbours if necessary and keeps the outfacing view in the Adj-RIB-out. For example, AS C would have its Adj-RIB-In, Loc-RIB, and Adj-RIB-Out tables as shown in Tables 2.1 in their respective orders.

Suppose now that the link c is broken, AS C would need to find a new best route from its Adj-RIB-In, which is $B\ A$. Its new tables are shown in Table 2.2 in the same order, note the changes relating to `1.0.0.0/8`.

In reality, many other factors influence the route selection process. Instead of AS-hop distance as the metric to determine preference, ASes more often use the financial cost

Prefix	AS path
`1.0.0.0/8`	A
`1.0.0.0/8`	$B\ A$
`1.0.0.0/8`	$D\ B\ A$
`2.0.0.0/8`	$A\ B$
`2.0.0.0/8`	B
`2.0.0.0/8`	$D\ B$
`4.0.0.0/8`	$A\ B\ D$
`4.0.0.0/8`	$B\ D$
`4.0.0.0/8`	D

Prefix	AS path
`1.0.0.0/8`	A
`2.0.0.0/8`	B
`4.0.0.0/8`	D

Prefix	AS path
`1.0.0.0/8`	$C\ A$
`2.0.0.0/8`	$C\ B$
`4.0.0.0/8`	$C\ D$

Table 2.1: Adj-RIB-In, Loc-RIB, and Adj-RIB-Out tables for AS C at equilibrium.

Prefix	AS path
`1.0.0.0/8`	$B\ A$
`1.0.0.0/8`	$D\ B\ A$
`2.0.0.0/8`	$A\ B$
`2.0.0.0/8`	B
`2.0.0.0/8`	$D\ B$
`4.0.0.0/8`	$A\ B\ D$
`4.0.0.0/8`	$B\ D$
`4.0.0.0/8`	D

Prefix	AS path
`1.0.0.0/8`	$B\ A$
`2.0.0.0/8`	B
`4.0.0.0/8`	D

Prefix	AS path
`1.0.0.0/8`	$C\ B\ A$
`2.0.0.0/8`	$C\ B$
`4.0.0.0/8`	$C\ D$

Table 2.2: Adj-RIB-In, Loc-RIB, and Adj-RIB-Out tables for AS C at equilibrium after link c is cut.

of each outgoing link and thus the business relationship with each neighbour as route selection criteria, and use AS-hop only in the event of a tie. Sometimes the Adj-RIB-Out is also filtered so private links used in Loc-RIB are not exposed to others.

2.2 Prefix hijacks and route leaks

Prefix hijacks, or route leaks as they are also known, are one of the most commonly seen operational incidents in ASes. Many anomalies and incidents are a form of prefix hijack and can take from minutes to hours to fix; occasionally days if they are noticed late.

A *prefix hijack* occurs when an offending AS inserts itself in the AS path of a certain prefix against the wish of the victim AS (the real originator). The offending AS can pose as the originator of the prefix or an AS close to the origin. It is not hard for the offender to deceive its neighbours to unknowingly propagate the false routes. If any traffic flows in as a result of this attack, the offending AS may drop packets destined to the hijacked prefix, creating a *black hole*, pose as the real destination with an *impersonation attack*, or forward to the real originator of the prefix, performing an *interception attack* [22].

As far as we can see, a *route leak* has not been formally defined in the academic literature. It is a conventional term from the network operator community describing a prefix hijack but usually implying non-malicious intent by the offending AS (i.e., operator error). Recently, in a draft RFC, Dickson defines it as a route announcement to a BGP link that violates its link classification, where a classification is one of 'customer', 'transit', 'peer', or 'special' [43]. In other words, a route leak is defined as a violation of link role. From the viewpoint of either an attack or a compromised resource, a route leak and a prefix hijack are the same; they differ in intention. Many operators also imply large scale when referring to a route leak, since smaller ones usually do not require attention from operators. Operators care about reachability; a large-scale route leak can change traffic patterns enough to cause routers to malfunction and require manual intervention. In many large-scale route leaks, reachability is also affected for prefixes that are not being hijacked until the router is restarted. In the rest of this dissertation we treat prefix hijacks and route leaks as the same and use the terms interchangeably.

Operationally, a route leak usually occurs when an offending AS announces a large number of prefixes that it has received from one or more neighbours. Other neighbours select this new announcement as their best route because of contractual relationships (which make an AS prefer customers) or modified AS-path values (making them shorter or ending in the offending AS). Then, depending on the error configuration and capacity, the offending AS may forward all new traffic to the proper next-hop (which is inefficient), or blackhole it (which is bad), or fail under the volume of new traffic (which can make the offending AS go offline or be temporarily blacklisted by others).

Let us illustrate a prefix hijack using Figure 2.1 as an example topology. Suppose that all links are functional, and that AS C and AS D have a business agreement to prefer link e over others whenever possible. One way that AS D can cause a prefix hijack is to make a mistake on the link e side, erroneously advertise its whole table to AS C, and further erroneously indicate that it owns all of these prefixes (so the AS path is just D). From AS C's perspective, its Adj-RIB-In, Loc-RIB, and Adj-RIB-Out tables are listed in Table 2.3. Because of a very short AS path, as well as the preferred link e, AS C now forwards all of its outgoing traffic to AS D (who may or may not forward to the correct destinations depending on the error configuration). AS C would also advertise these new best routes onwards to its neighbours, possibly influencing other ASes as well.

There are some famous route leaks that are often mentioned in academic literature. We offer a brief description of each below.

- **AS7007** (Apr 25, 1997)[1]: the first major route leak incident, in which AS7007 received a full view from its downstream and deaggregated these 23 000 routes into

[1]http://www.merit.edu/mail.archives/nanog/1997-04/msg00444.html

Prefix	AS path
`1.0.0.0/8`	A
`1.0.0.0/8`	$B\ A$
`1.0.0.0/8`	D
`2.0.0.0/8`	$A\ B$
`2.0.0.0/8`	B
`2.0.0.0/8`	D
`4.0.0.0/8`	D
`4.0.0.0/8`	$B\ D$

Prefix	AS path
`1.0.0.0/8`	D
`2.0.0.0/8`	D
`4.0.0.0/8`	D

Prefix	AS path
`1.0.0.0/8`	$C\ D$
`2.0.0.0/8`	$C\ D$
`4.0.0.0/8`	$C\ D$

Table 2.3: Adj-RIB-In, Loc-RIB, and Adj-RIB-Out tables for AS C at equilibrium after AS D falsely announce its whole table to C and claiming the prefixes as its own. Note in this case we assume link e to be preferred.

around 73 000 routes as well as rewriting the AS path advertising AS7007 as the best. Likely due to a router software bug.

- **TTnet** (Dec 24, 2004)[2]: in this incident, Turkish ISP TTnet (AS9121) advertised over 100 000 routes with itself as the destination. Its upstream took the advertisements and propagated on, eventually affecting the whole Internet.

- **Con Edison** (Jan 22, 2006)[3]: Con Edison (AS27506) advertised many prefixes as its own (thereby hijacking them), some of which belong to its customers, while others are unaffiliated.

- **Pakistan Telecom and YouTube** (Feb 24, 2008)[4]: while implementing a governmental censorship policy, Pakistan Telecom (AS17557) accidentally advertised as its own a more-specific prefix in a range owned by YouTube (AS36561), which was propagated to the world by its upstream. YouTube responded by first advertising the same prefix, and later two more-specific prefixes.

- **China Telecom** (Apr 8, 2010)[5]: in this incident China Telecom (using one of its AS numbers, AS23724) originated 37 000 prefixes for 18 minutes, including some that serve websites of the US governments and military.

In the above description we can see that there are three roles in a route leak: offending AS, victim AS, and other ASes, without any assumption of direct connectivity on any pair of roles. Below we describe in detail a route leak for each of these three vantage points from an operational point of view. Apart from a description of what would happen,

[2]http://research.dyn.com/2005/12/internetwide-nearcatastrophela/

[3]http://research.dyn.com/2006/01/coned-steals-the-net/

[4]http://www.ripe.net/internet-coordination/news/industry-developments/youtube-hijacking-a-ripe-ncc-ris-case-study

[5]http://www.bgpmon.net/chinese-isp-hijacked-10-of-the-internet/

we also discuss how to detect and mitigate a route leak from each perspective. For this discussion we use the above example prefix hijack and let AS D be the offending AS, AS C be the victim AS, and AS A to represent any of the other ASes.

2.2.1 AS D, the offending AS

From the perspective of the offending AS, local traffic is not in immediate danger as the prefixes being hijacked belong to others. From the control plane, it is hard to notice that anything has gone wrong because hardly anyone monitors what prefixes are announced in the egress direction, and if the route leak is triggered by a bug in the router software it may not even be detectable. If the data plane is being monitored, there may be a spike in the traffic load going through the local network if the hijacked prefixes are usually busy. Otherwise, the offending AS may not notice that it has caused a route leak until contacted by other affected ASes.

Assuming the attracted traffic is not beyond the capacity of the offending AS, causing it to fail or creating the need to restart BGP routers, an outside AS (most likely the victim or AS D's upstream) will raise the issue. Surprisingly, this social method involving human operators is the most effective (if not the only) way of communicating about such incidents. The lack of any automation makes problem discovery and resolution a slow, manual process.

We are not aware of any ASes that actively monitor or keep a history of their BGP control plane data, or BGP updates in either direction. The current tools allow operators to monitor if an AS's prefixes are hijacked, but not if the AS is hijacking others' prefixes (an order of magnitude more prefixes would have to be monitored, which is harder to set up). There are also other technical difficulties such as the fact that each BGP neighbour may be seeing a customised version according to local configurations and contracts, making it hard to track multi-dimensional history. The time it takes to set up and maintain such a history may be hard to justify for the occasional route leak that directly affects them; the ASes that have the motivation to keep track of their control plane are also less likely to make mistakes that cause route leaks. The incentives exist, however, in the forms of social pressure, threats of upstream provider disconnection, customer dissatisfaction, unrealised service level agreements, and bad reputation.

To fix an existing route leak, AS D's operator needs to identify the changes made by BGP speakers that caused the leak. A common cause is the removal of a filter which lets a group of prefixes through. A network with good change procedures should keep a log for review of incidents. Once an offending action is identified, operators can revert the change, and, if needed, restart relevant routers. BGP will then take over and AS D will withdraw the hijacked routes.

Operational challenges can make mitigation hard: smaller ASes may not keep a proper change log, and thus take much longer to track down the offending filter. In this case, the upstream or AS D itself could disconnect the relevant router before the problem is solved. Another challenge is the possible excessive traffic affecting routers to the extent that they need to be fully restarted (as opposed to a soft restart which incurs little downtime), taking minutes before they are able to start forwarding packets. In this situation, all traffic passing through the affected routers in AS D will be affected, not just the hijacked prefixes.

So it is prudent to follow best practice and formulate well-defined procedures for working on the network. By doing so, much of the risk can be mitigated. Even if something does happen, a properly maintained network can recover much quicker and avoid downtime.

2.2.2 AS C, the victim AS

From the point of view of AS C, the victim AS, the effect of a prefix hijack is very obvious if it starves the victim of traffic. Even though BGP is asymmetrical in that the egress route may not be the same as the ingress route, a functional outwards link does little work if no response packets can be received. It is even possible that the phone lines, if they are IP-based, will suffer outage, further complicating recovery efforts.

More specifically, it is possible to detect this type of attack from both the control plane and the data plane. From the control plane, there are online services, such as BGPmon [182], which let operators specify a set of prefixes to monitor. A simple anomaly check is done on those prefixes and email alerts are raised for suspicious updates (nothing is certain because BGP does not convey intent). From the data plane, we can set up monitors either inside the local AS or externally, to `ping` or `traceroute` to or from set vantage points of the Internet. Many commercial services already provide this, but mostly for application-level reachability (which would also fail if the prefixes are unreachable). RIPE Atlas is a recent project that monitors reachability for ASes. It remains to be seen if this service can get operators to start monitoring at scale.

In the YouTube/Pakistan Telecom incident, YouTube reacted by first advertising the same /24 prefix, and later two /25 prefixes that make up the hijacked /24. The recovery was also helped by the fact that the victim was prominent, so the event became widely known quickly, and Pakistan Telecom's upstream AS dropped their BGP session. Indeed, apart from advertising more specifics, another effective mitigation method is by asking the hijacker's upstream to stop the bogus announcements at source. However validation of ownership can be hard without a reliable resource allocation registry. The upstream of the remote AS may also be reluctant to work with a third party against its customer, or fear legal liability. In more obvious cases or with ASes with a known bad reputation, upstream coordination is generally possible. However, hijacks (intentional or otherwise) by smaller players may be self-resolving. If too much traffic is attracted, the AS that causes the incident may DDoS itself and its routers will be forced to shut down. When the control plane reacts to this data plane black hole, BGP will work the way it is supposed to, and the next best route — likely the original correct route — is selected by affected neighbours.

2.2.3 AS A, and any of the other ASes

For an outside observing AS that is affected by a prefix hijack, the experience is not as catastrophic as for the offender or the victim . Depending on the number of hops from AS A to AS D, and the scale of the route leak, AS A's border router may see a few or many BGP announcement updates in a very short time period, up to the whole table (500 000+ prefixes currently). Most likely, the border router will process the updates without any problem both in terms of processing speed and local routing selection updates. The speed of incoming updates is no faster than that of a newly booted router which re-announces its local table. The size of the overall table usually does not change. However, if an incident similar to the AS7007 one occurred today, where a large number of prefixes are deaggregated by the router, routers at AS A may experience memory shortage and behave abnormally.

After the control plane changes, some new traffic may flow in since AS A's neighbours now pick this route as their best one. The traffic increase depends on the network topology around AS A and its AS-hop distance from AS D. There might be little new traffic, or a lot if AS A is very close to AS D or if AS D usually handles a lot of traffic.

Route leaks are hard for an outsider to detect. There are many legitimate reasons for route changes, as discussed earlier, and BGP does not reflect intent. Given the lack of a log of control plane changes, AS A may be unable to recognise the issue; and it does not have a direct incentive because its own prefixes and customers are not affected.

Once the problem and its cause become clearer — from community mailing lists for example — AS A may manually revert the erroneous updates by blacklisting an AS or prefixes. Another option is to wait for AS D and AS C to sort out the issue and let the corrected BGP updates propagate over. If the latter option is chosen, there may be another surge of control plane traffic as AS D withdraws many of the routes it has previously announced. The performance issue of this mass withdrawal is comparable to that of the mass announcement phase. Many smaller route leaks probably go unnoticed, especially if AS D corrects them in time or if AS C and AS D successfully communicate privately without escalating to a public forum.

2.3 Real-world BGP

The way BGP works in the real world is rather different from how its design looks on paper. Apart from the basic reachability information that the protocol exchanges, AS-side logic has grown more complex. The need for more than just vanilla BGP comes from the commercial incentives faced by networks that run BGP: incoming routes need to be in order and sorted by preference, and the outgoing routes need to be exposed for reachability while following a reasonable traffic balance. Equipment vendors have so far provided many filtering and traffic engineering tools in their routers.

One important feature that is recommended as best practice is *prefix filters* [44]. BGP design says that ASes should accept what is advertised to them but select the best route according to local selection and preference criteria. Prefix filters allow operators to group prefixes or sources of prefixes and assign priorities to each group, or deny specific groups altogether. As a result, commercial relationships, such as preference for one's own customers, are easily expressed and reflected in the routing decision process.

Another feature that has seen some use is the *maximum-prefix* counter. It is a simple idea: watch for the number of prefixes that this neighbour advertises to me. But it helps in catching the most likely mistake in AS interconnections: a large-scale route leak. In such an event, a neighbour would advertise many more prefixes than it normally does, and without a threshold setting it is very easy for the local router to accept them and propagate them as truth. Of course, a malicious AS can evade this counter if an attack is launched on purpose.

On the egress side, AS path padding and deaggregation are two common tools provided in routers. Traffic engineering of incoming traffic is performed when an AS wishes to load-balance traffic across multiple connection points to the Internet. AS path padding is used to manually create a route that will probably be treated as less preferred by others (because it has many AS hops and thus interpreted as being farther away), by repeating the final AS number multiple times. Deaggregation gives the owner finer control of the traffic-load origin by splitting a larger prefix block into several more-specific blocks. These methods are usually backup routes or accompanied by other backup routes, and create unnecessary load on the full routing table size of other networks. APNIC, the Asia Pacific counterpart of NANOG, runs a weekly report whose last update shows a total of 590 116 table entries that can be aggregated to 217 220 prefixes, a deaggregation factor of 2.72[6].

2.3.1 Running an AS

The maintenance of an AS's connection to the outside is a continuous task; it requires constant monitoring to ensure no downtime occurs. Although BGP is a relatively simple

[6] http://thyme.rand.apnic.net/current/data-summary

protocol, its permissive attitude to many behaviours, such as its allowing one AS to modify data received from others before forwarding, means the ecosystem can break in many ways. Operators also add many local policies which together result in BGP getting more complex [24]. There are many procedural tasks (such as adding a customer, or taking a router offline to be serviced) that, if not done carefully, can lead to catastrophic failures of the local AS and possibly its neighbours. Many ASes therefore have set processes and procedures for changing the network; but these processes tend to be homegrown, because of the lack of good tools to support BGP operations. Examples of potentially hazardous common procedures include adding/removing a new customer or a peer, updating a filter, and the provisioning of a new network device.

2.3.2 Peering

AS peering is the fundamental building block of the Internet. It allows the exchange of reachability data with a physically reachable neighbouring AS. In practice, business relationships come into play to make peering more complicated.

An exchange of data and traffic is a business transaction and incentives need to be considered. Peerings can comprise of an equal one without money changing hands, or one with an unequal level of traffic, possibly with financial compensation in the opposite direction. Both sides also need to agree on a measurement method to determine the volume of traffic. Service level agreements, too, can be a factor of negotiation, as different ASes may have different levels of performance and reachability. These are just some of the factors to be discussed in negotiating a peering contract, which is typically confidential.

Then there is the implementation of a peering contract. Network engineers need to make sure that the configuration reflects the business intentions behind contract clauses. As it is ultimately the actual configuration that provides the negotiated services with the right levels of service, such configurations (and especially special cases) need to be tested ahead of time to make sure contracts are followed.

Some ASes provide a 'looking glass' service to others. These are BGP tables as seen from the vantage points of these ASes, and show potentially valuable reachability information for network troubleshooting. Alternatively, two public repositories exist to collect such information as well: Oregon's Route Views[7] and RIPE RIS[8]. Apart from a live view, these two repositories also make all historical data available.

Even with the existence of looking glasses and public repositories of routing data, many peerings are not reported. For example, a backup link may not have any routes advertised on it until they are needed. Companies exporting a looking glass view to the public can filter out routes they do not want to disclose. Public repositories do not always get the full

[7]http://www.routeviews.org

[8]https://www.ripe.net/analyse/internet-measurements/routing-information-service-ris/ris-raw-data

picture either. The tree-like nature of routes that span out from one particular vantage point also inherently miss connections between two branches further away. Larger content providers, such as Google and Yahoo, usually peer with anyone at an exchange point, in order to minimise transit traffic charges, and these routes do not generally show up in public databases either.

2.3.3 Internet governance

Even though the Internet is a distributed confederation of networks, there are still a few recognised governance bodies. One of the main responsibilities in the BGP world is the centralised allocation of AS numbers and IP address prefixes. This falls under the care of Internet Assigned Numbers Authority (IANA), which in turn delegates some relevant work to Regional Internet Registries (RIR), each with the power to delegate further. Apart from special private or reserved blocks, IP address prefixes are unique across the Internet, and each has an assigned owner who may or may not actively use the whole assigned range. Similarly, AS numbers are also unique across the Internet. Although prefixes and ASes were not digitally signed or otherwise protected by cryptography until now (as we will discuss in the context of RPKI and BGPSEC), neighbourhood watch and the honour system ensure that nost people are honest and only use what they are assigned most of the time.

The assignment of AS numbers is permanent, as the same number will not be re-assigned to another entity, and the assignment of a prefix to a specific AS is generally permanent; we have not heard of a prefix block being reclaimed into the global available pool. Naturally, we would want a registry of AS-prefix associations for record keeping and for other operators to know what to accept from their neighbours. It turns out that this task is very hard. On the AS end, although AS numbers are permanent, companies can merge, go bankrupt or change hands, all of which can lead to a change in AS number ownership. On the prefix end, some companies have downstream customers to whom they sell or lend prefix blocks, which can be reclaimed by the owner if needed. We have even seen a major sale of prefix blocks, from Nortel to Microsoft, essentially treating these as assets (but as they were assigned before ARIN, more lenient rules applied). Some registries exist, and include voluntary registrations such as RADb and more official ones such as the one maintained by RIPE. In both cases, the transient nature of businesses and the failure of proper handover in case of ownership change makes ground truth very hard to obtain. The problem will become more important as RPKI, an infrastructure needed by BGPSEC, gains more traction. It will become worse still when cryptography is added by BGPSEC deployment, as this will create s a very much stronger requirement for data to be correct and up to date.

From a day-to-day perspective, network operators rely not only on the registry data mentioned above to sniff out anomalies, but also on instinct and self governance. For example, a common way to coordinate and avoid possible misunderstandings is through network operators' mailing lists, of which the one by NANOG is the biggest. It is an open forum where affected operators can quickly share and seek data points from others to assess a problem. Where an offender does not cooperate, victims can also complain to their upstream provider — usually a bigger AS who can provide additional assistance or clarification. In this way, daily operations benefit from self-governance of the crowd, and social pressure can often secure compliance.

2.4 Observations in practice

Following a month of internship at a medium-sized ISP, we were able to observe day-to-day operations within an AS. The ISP has a few hundred BGP peers; it has a presence in many countries and at a few major exchange points. Its upstream and transit providers are tier-1 ASes, so it is very close to the core of the Internet. While different networks have different standard modes of operation, and a procedure forbidden in one network may be completely acceptable in another, what we have observed is not far from the mainstream mode of operations according to informal discussions with operators from other ASes.

We also monitored the operations mailing list of a large Internet exchange point (IXP) for 13 months. Operators would use this channel to communicate with other ASes when actions from many others are required. In total, we observed 23 cases involving 20 ASes reaching out to the community (three were involved in two cases each) to confirm an anomaly or to clarify honest intent. We find two very common failure cases: tripping a maximum-prefix due to business growth, which had 11 reported cases, and configuration errors causing a route leak, which had 11 cases, all of which also tripped some maximum-prefixes (according to self-reported emails). The cost of manual operator intervention averages to almost three per month just to peer at this particular IXP. We note that the maximum-prefix setting is always tripped in large enough mishaps, as well as causing more chaos; while manually configuring routers is often the root cause.

2.4.1 Maximum-prefix settings

The maximum-prefix setting is a software feature found in many routers. When set, it indicates the maximum number of prefixes a router accepts from a remote router via a specified link. Either side of the link can set a maximum-prefix before the session is established. It is not part of BGP itself but rather an add-on to help ASes avoid being affected by a misbehaving neighbour. It limits damage from a neighbour who suddenly announces many more prefixes than it normally does — a classic symptom of a route leak.

But this feature has its downside too: it is too coarse, and often punishes the innocent. Such failures have immediate consequences, such as session termination, and cannot be fixed unilaterally. Sometimes it takes days before the disconnection is realised and correct connections are re-established.

Maximum-prefix value

The maximum-prefix value is set unilaterally by one end of a BGP connection to limit the other. Usually this value is set at around two times the current number of prefixes received from the link. However, one cannot see the value the other end has chosen or if this setting is even used at all (unless communicated out of band).

The maximum-prefix setting is usually used by a provider on its customers, and sometimes between two peers. What then may happen is that one AS (the downstream one in the first case) slowly grows over time and acquires more customers or prefixes, eventually triggering the remote maximum-prefix value. Operationally these values are not always reviewed to account for business growth.

What some ASes do, especially when they acquire a large customer or number of prefixes, is to email neighbours and request an adjustment of the maximum-prefix value (which can be done unilaterally). However, this method requires manual work and not all mailing list recipients act on such requests. In at least one case we have observed an AS being automatically dropped by many neighbours as a result of reaching the max-prefix, even with an advanced email warning.

Dealing with a reached prefix

Different actions can be executed automatically by the router when the maximum-prefix count is reached. In all three major router implementations — Cisco, Juniper, and Quagga — we can set a warning threshold before the actual limit. When the number of prefixes eventually reaches the hard upper limit, Cisco and Quagga allow session drop or log only, while Juniper can either reject new routes or log only.

All three options can potentially cause problems. Session drop is the most severe, as it cuts the neighbour off completely, including previously received routes. Re-establishing the connection after a session drop can be done manually or automatically after a preset time interval, sacrificing either timeliness or flexibility (and there may be other issues since the maximum-prefix value cannot increase by itself even with automatic retries). Rejecting only new routes but keeping the old ones is more reasonable, but legitimate routes may be rejected silently and unilaterally, affecting the local view of the global routing table. Finally, a logging-only reaction can be of little operational value, as human intervention will probably not arrive in time.

2.4.2 Filtering and registries

Maximum-prefix settings are the most coarse example of prefix management. In reality, more fine-grained methods exist. They are done at the router or AS level to optimise or ensure the sanity of routing data received from or sent outside. In the general case, any part of the BGP update attributes can be matched or manipulated. In reality, most filters relate to prefixes or AS-paths.

Filters are used extensively by operators. While simpler setups mostly use filters to ignore paths to RFC-specified private prefix blocks, others may have one or more filters for each BGP link, filtering either egress or ingress advertisements, or both. As a result, filters have a direct impact on local policy, and therefore on routing decisions.

Filter management

Even with just a couple of filters per customer, the total number of filters can quickly grow out of hand, and managing them effectively becomes a real problem. Unfortunately, the usual way to manage filters is through the command line interface (CLI) of the router, which suffers from many usability issues. For example, manual entry of AS numbers or prefix addresses (especially in IPv6) is vulnerable to typos, the CLI commands require experience to drive well, and their outputs are not informative.

Many better-managed ASes have a private collection of scripts that they use to work with their networks. Though a slight improvement, these scripts also suffer from many of the drawbacks of direct CLI: usability can be poor with uninformative outputs, manual entry of some data (i.e., filter name, prefix addresses) is still required and can be error-prone; and the robustness of these scripts are uncertain. They may be good at what they are designed to do, but become complex and fragile over time. The usual software development pitfalls apply to custom scripts as to everything else.

While filters are an effective tool to stop route leaks, we learned that they are also the cause of route leaks. It is not uncommon for operators to remove filters for maintenance or replace them. Since BGP does not support atomic operations and router support for filtering is primitive, human errors are common to encounter. If a wrong filter is removed or if something unexpected occurs, routes that are previously blocked by a filter can propagate to an undesired neighbour. For example, there are some routes that are only meant for downstream customers or the local network; mistakes in filter management can leak them to upstream providers and peers. Worse, the CLI does not provide contextual information such as the number of routes 'in effect', or affected, by each filter, which can warn the network engineer to be more cautious when working with a large filter. The CLI also does not provide the number of routes that are unblocked after the removal or modification of a filter. As ingress filtering becomes too complicated to manage towards the core of the Internet, ASes are expected to self-filter their egress updates. A well-positioned

AS near the core of the Internet can seriously affect the Internet as a whole, all from a single mistyped filtering command.

Registries and ground truth

Registering a prefix block at a registry demonstrates ownership and responsibility, facilitates filter construction as owners can indicate some preferred policy in the database, and provides a way for others to reach the operator in time of need. In reality many factors hinder the reliability and accuracy of registries. Registries not in the hierarchical chain of Internet governance such as RADb have no means of verifying the legitimacy of a claim of ownership. Legacy address blocks that were assigned before IANA, which is now in charge, do not fall into the hierarchy. Operators may not keep their prefix lists up to date. Companies are merged, bought, or dissolved without removing their registered prefixes, leaving the later owners of the same prefixes unable to register without going through lengthy bureaucratic processes. A recent study has found varying quality of registry data, showing a mix of acceptance among operators [98].

Despite these issues, registration is still considered good practice. Some ASes even require, as part of their peering policies, that their neighbours register. In part, it is because filters can be generated automatically with a consistent online database, thus reducing human errors when working with filters; registration also allows timely propagation of updated information without the need for human intervention in each neighbour.

2.4.3 General observations

Our general observation is that current management software is inadequate in supporting what operators wish to accomplish. From the mistakes we have observed, operators cannot easily express their intentions in terms of a set of router commands. As a result, day-to-day management of filters and routes can cause advertisements to unintended BGP neighbours, even if the triggering task is only concerned with internal routes or specific neighbours. Filters are often the only mechanism stopping such mistakes, yet they are also rapidly changing, thus leaving room for error.

We believe a top-down mandatory access control mechanism can help operators by creating a new class of long-term route advertising and filtering policies, and making them harder to modify to prevent accidental changes.

2.5 Adding Mandatory Access Control (MAC)

Our goal is to separate day-to-day operational procedures from long-term invariants. The need for this separation was suggested by our observations in Section 2.4. Even though filters and routes are often changed, some general invariants hold. For example, routes from one upstream provider should not be advertised to a different upstream provider, and deaggregated routes of customers should not be advertised to the rest of the Internet without being aggregated first except for special routes. In other words, the neighbour from which a route is accepted dictates where the same route can be advertised.

The relationship of which egress neighbour can receive routes from which ingress neighbour does not easily change, and is the long-term invariant we want to enforce. When these long-term invariants are violated it generates BGP updates that are announced to other neighbours, affecting others and potentially requiring their cooperation to restore to a previous sane state. In general, day-to-day procedures, such as changing internal paths through which routes are received and distributed, tend not to involve other ASes. When internal paths are changed, few outside neighbours are affected.

It is natural to group neighbours into roles depending on business relationships. Three most common roles are *provider*, *customer*, and *peer*. Other groups can exist for ASes with more complex business relationships, and special arrangements can be in their own roles. Thus, Role-Based Access Control (RBAC) appears the natural security policy to use. Top-down enforcement makes the policy a kind of mandatory access control, or MAC. Strictly speaking, the operator has full power over the network, and is not limited by this 'MAC': but we propose to make it harder to change what our MAC enforces; and while the operator can still modify or override existing invariants, it takes deliberate effort, and cannot be done accidentally. So we feel justified in using the term 'MAC' for it.

2.5.1 Design and implementation

We designed RBAC as a MAC mechanism and implemented it by modifying Quagga, an open-source BGP speaker[9]. The design consists of two parts: classifying incoming routes into their respective roles, and enforcing outgoing announcements based on these roles and the relevant egress neighbour. When a route is first received, it is processed through filtering functions before being added to the main Loc-RIB table for selection. Just before it is added into Loc-RIB, we add an extra attribute to each route — its 'role' according to the policy's neighbour-role assignment. All routes from the same neighbour are tagged with the same role, but the values are repeated for each role as they are stored on a per-route basis.

Enforcement comes at announcement time: when a route is identified as the best, it is advertised to neighbours. For each neighbour to which Quagga is about to advertise, we insert a check on the role of this particular route, and verify whether it is permitted to be announced to this neighbour. If, due to a filter error, routes to one upstream provider are advertised to another upstream provider, a very large amount (up to the full table) of

[9]http://www.nongnu.org/quagga/

routes will undergo this process within a short window. As these routes may have different role attributes, they are processed independently.

A high-level overview of the Quagga update logic is shown below. In our prototype, we add steps 5 and 8.

```
1. Receive from BGP neighbour
2. Store in Adj-RIB-In
3. Parse route and attributes
4. Basic checks, apply incoming filter and modifier
5. RBAC role assignment
6. Insert or update into Loc-RIB
7. Route selection
8. Check newly selected route against RBAC permissions
9. Store in Adj-RIB-Out
10. Send to BGP neighbour
```

One advantage of our mechanism is that it is deployable by an AS unilaterally. Unlike protocol upgrades such as BGPSEC, we improve on the network management workflow, and do not require outside collaboration. This gives a strong economic incentive for deployment to ASes because they are direct beneficiaries by adopting our software and do not need to wait for the network effects to catch on.

Within the current Quagga design, it is not possible for our access control mechanism to defend against exploits against the Quagga software itself. We cannot minimise our code into a reference monitor and depend on it alone to restrict what Quagga can and cannot do. Consequently, our threat model does not include software bugs within Quagga.

It is also not possible to guard against intentional attacks from the same operators we are trying to help. As we just make it harder, but not impossible, to announce updates beyond RBAC policy, there are still many legitimate needs for special updates that require operators to override RBAC checks. In the end, we focus on careless errors rather than targeted attacks. A complementary class of attack is intentionally misleading (and therefore malicious) BGP updates from other ASes. While we do not guard against it either, BGPSEC considers it in its scope. We discuss the extent of its success in Section 2.6.

Another shortcoming is the action we can take after a route is blocked from being announced by RBAC checks: currently we drop the route with an alarm, but because a route is prevented from being announced *after* the BGP decision process, the neighbour on the other end of the BGP link would be deprived of this announcement and thus become inconsistent with our view for this particular prefix.

This shortcoming is actually inherent in BGP design. In the route selection process, a best route for a given prefix is selected for all neighbours instead of on a per-neighbour basis,

BGP link (incoming)	Role
d	provider
e	peer
local	self

BGP link (outgoing)	Role	Decision
d	customer	`permit`
d	self	`permit`
e	customer	`permit`
e	self	`permit`

Table 2.4: RBAC assignment table (left) and permission table (right) of AS D. In the assignment table, roles are assigned based on BGP link; in the permission table, decisions are made based on the BGP link and the role. Only the `permit` entries are present in the permission table as the default decision is `deny`.

which is where our RBAC operates. Without a way to automatically force the routing selection again, we can only rely on operators to correct the issue manually. However, it should be noted that the inconsistent state only affects relevant neighbours and does not affect the local AS. It also only affects these neighbours by announcement starvation, and all their existing routes will still function meanwhile. Also, this inconsistent state can only be generated by a local configuration change, an external AS cannot cause it. Therefore it cannot be used as an attack vector against users of our mechanism.

2.5.2 Evaluation

Let us illustrate how our RBAC extension stops an accidental prefix hijack as discussed in Section 2.2 (see Figure 2.1). The misconfiguration leads to undesired behaviour because link e is preferred by business agreement between AS C and AS D. Suppose D unilaterally deploys our RBAC version of Quagga, and labels each (incoming) link with roles as in the left part of Table 2.4. Note that these roles are directional: should B also deploy our Quagga it would label link d as *customer*. We also add a *self* role for routes of prefixes owned by D itself. Here D is a typical AS receiving a full BGP table from B, its upstream, and it has a peering with C. It is likely that neither C nor D wishes the advertisement of a full table from D to C, as C has its own upstream provider which is faster and more reliable. Therefore link e should only carry routes to C and D's own prefixes and those of their (possibly non-BGP) customers. When link e is *first set up*, D configures the RBAC permission table as shown in the right part of Table 2.4.

Now suppose an operator of D makes a mistake in filter management, and removes a filter which in turn causes it to advertise its own full table to C via link e. In Section 2.2 we see this is how a route leak occurs. However with our RBAC addition, step 8 stops the advertisement because link e and role *provider* evaluates to `deny` (as the default decision). The operator is then notified and can reverse the relevant operations. The advantage of our version of Quagga lies in the fact that it is impossible for an operator in D to overcome this permission check accidentally, unless the RBAC assignment table or permission table

	Mean	Std. dev.	Median
Stock Quagga	2.25 s	0.119 s	2.21 s
RBAC-Quagga	2.30 s	0.145 s	2.24 s
Overhead	2.54%	21.8%	1.13%

Table 2.5: Performance of stock Quagga versus RBAC-Quagga in loading 579 155 routes from a BGP neighbour. Units are in seconds except for overhead ratio.

is modified. The extra steps required to perform such modifications makes accidents less likely.

With the introduction of steps 5 and 8, we add a small penalty in Quagga performance. So we wish to measure the performance overhead during large updates such as a full table. We take the BGP snapshot RIB as seen by RIPE RIS at London Internet Exchange (LINX) at start-of-day of 2016-03-01, and load it into a stock Quagga to simulate an upstream provider. In all 579 155 prefixes are present, and each prefix has one route to it (post-selection). We configure an RBAC-Quagga to peer with it and bring up the connection, forcing it to receive with role assignment all routes as fast as possible. This process is repeated 10 times, and is then repeated 10 times with a stock Quagga in place of RBAC-Quagga. The results are shown in Table 2.5.

In terms of memory overhead, with stock Quagga 40 bytes are used for each route. Additionally, each prefix has additional pointers to structured objects that may be shared among different routes. For example, if an AS path is the same for multiple prefix routes, then it is shared in the same memory (called *interning* in Quagga code). Our current modification allocates an additional one byte into each route, allowing for 256 possible roles. The same full-table snapshot from LINX, including selected and all unselected but eligible candidate routes, contains 5 299 916 routes in total. With one additional byte for each route we require approximately an extra 5.3 MB of memory. If we were to modify the code further to take advantage of interning, there would be only one extra byte for each *role* present, a minimal impact on memory.

We noted above the difficulties caused by simple mandatory rules such as max-prefix and filtering, in that rejected updates can require manual intervention and even lead to inconsistency. Our view is that will have to be some hard rules, but they had better be designed properly and tested thoroughly in real environments. Using more general mandatory access control mechanisms such as RBAC creates the design space in which this can be done; we do not claim that the design task itself will be trivial. (Indeed, the deployment of mandatory access control systems such as SELinux in Android and elsewhere has required serious engineering effort). However there is one reason why this task cannot be avoided much longer, and that is the forthcoming introduction of BGPSEC.

2.6 RPKI and BGPSEC operational challenges

Recently, there has been much work on a proposal to secure the semantics of BGP, in particular the AS-path attribute. As noted above, there is an active Secure Inter-Domain Routing (SIDR) group working on specifications as part of the IETF[10]. The proposal comes in three steps: first, a Resource Public Key Infrastructure (RPKI) is to be designed to enable the storage of signed AS to prefix block mappings, and this is already an RFC [108]; then, prefix origins (the last AS number in the AS-path) will be checked against this RPKI; finally, with BGPSEC[11] the whole AS-path will be checked for validity, and each AS will sign the paths they announce, wrapping AS-path in an onion-type signature scheme. In contrast with the mandatory access control mechanism proposed above, where we make sure an AS cannot make mistakes within its own domain, BGPSEC ensures the integrity of routes *after* they are announced by an AS.

The main class of benefits that BGPSEC brings is to vastly reduce the number of ASes that can manipulate a given prefix. Whereas any AS could do this previously, routing will now only be vulnerable to ASes that are legitimately in the path of route propagation. Because signed updates also protect the order of ASes on a path, fewer ASes can now claim to be close by the origin of a prefix, and the farther they are the less influence they will have should they do a prefix hijack. The prefix originator can now know that its prefix is harder to manipulate, and an AS receiving a route can now be more confident that the announcement is genuine.

However BGPSEC does not fix where an AS sends a route after it legitimately receives it. A route leak is still possible even with full RPKI and BGPSEC. The MAC mechanism we proposed above fixes this exact problem for the case of accidental configuration errors. It is complementary to BGPSEC in that it protects what happens inside an AS. However, because deployment of our MAC is voluntary, a malicious or compromised AS can still fail to use it and mount an attack.

With the introduction of cryptography in RPKI and BGPSEC, new sources of error also arise. It has been shown that a regular computer user cannot work with PGP encryption competently [178], and most network operators are not much better trained cryptography. BGPSEC management software need to be sufficiently usable to ensure that the already fragile BGP may not be subject to a new class of mistakes. Our MAC mechanism can help ensure that those thing that should not go out do not go out, but cannot monitor for the opposite and ensure that everything that should go out will. For example, if a signatures are not done properly, for any one of a number of technical reasons, then novel failure modes can arise that lead to denial of service and black holes.

Similarly, RPKI is a cryptographically-strong source of AS-prefix ownership, and will

[10]http://datatracker.ietf.org/wg/sidr/charter/
[11]Draft RFC: https://www.ietf.org/id/draft-ietf-sidr-bgpsec-protocol-15.txt

be relied on to generate and check routes within BGPSEC. Whereas various current routing registry databases are optional, and therefore not maintained by ASes to be up-to-date, an outdated RPKI will seriously disrupt route generation and propagation. It is a new task for network operators to keep up, and a new source of potential error; the introduction of cryptography will require a step change in the quality of AS administration. RBAC goes with the grain of this.

The road to BGPSEC, or any other improved protocol, is long and difficult. New technology will likely propagate across the Internet slowly. Smart operators will recognise the potential benefit but many will be reluctant to be the first to adopt, due to the lack of incentives that these protocols give to first movers and the difficulty of doing a proper implementation without mature tools. Luckily, we have seen some technology-motivated ASes allocating some engineers to give RPKI and BGPSEC a try, while sharing their experiences among other ASes[12].

2.7 Conclusion

In this chapter we take a practical view of BGP operations. Through performing fieldwork at a medium-sized ISP and hanging out on operators' mailing lists, we discovered the major BGP failure cases faced by operators. We found that current tools are inadequate for supporting BGP operations; many erroneous configurations can easily be made that affect both the local AS and others nearby.

We propose a mandatory access control extension to BGP management software using RBAC. By assigning different BGP links into roles, and by requiring operators to denote neighbour-role relationships at setup time, we can ensure this information is enforced in a way that is unaffected by transient local changes. A policy can thus be set to limit accidental announcement of routes into non-permitted neighbours. We prototyped this using Quagga and tested it; we found minimal overhead.

While BGP is migrated to BGPSEC over the coming years, it will be insufficient to protect only the protocol. A lot of work will have to go into how operators actually use BGPSEC. If accidents are just as easy to make, then the BGPSEC infrastructure will not add much value, and ASes may be reluctant to invest in it, as it does not give much in the way of incremental local benefit. Fortunately, our RBAC extension does offer such benefits and works with BGPSEC as well, complementing it in ways that may make BGPSEC more usable and more attractive. It can also mitigate the increased workload and fragility that BGPSEC may introduce.

[12]`https://www.nanog.org/sites/default/files/wednesday_george_adventuresinrpki_62.9.pdf`

Chapter 3

Access control for SDN

Software Defined Networking (SDN) has enabled significant network innovation and cost reduction, but network control in the face of competing uses remains a major challenge. Where competing tenants share a data centre network, network control must reconcile conflicting policies and enforce activity access controls in a complex virtualised environment. In this chapter we introduce AXE, a prototype attribute-based access control system that enables SDN controllers at both the tenant and fabric levels to manage SDN applications (apps) in a clean and consistent way. Experiments show that our prototype introduces modest overhead for a scalable design that is demonstrated to support a range of different SDN applications. Clean virtualisation enables our access control system to support hierarchical virtual networks with separation of policy and mechanism, policy conflict resolution, and separation of the stateless and stateful aspects of access control. We discover that stateless access controls for virtual SDN controllers map canonically to stateless firewalls, which enables us to draw interesting and more general lessons about both access control and firewall design. Not only can the stateless aspects of firewalls migrate to the SDN infrastructure, giving greater flexibility, manageability and assurance, but we can manage security state better by breaking it out into apps where its logic, dependencies and possible conflicts become much clearer.

3.1 Introduction

Software-defined networks replace specialist network devices such as routers and load balancers with combinations of commodity network switches based on low-cost merchant silicon. These switches provide a set of common, well-defined interfaces (e.g., the OpenFlow protocol [123]) for their control. Control-plane processing can in turn be implemented on any general-purpose computer; although rarely a commodity beige box, SDN controllers operate on a range of systems from the built-in low-power CPU chips common in switches themselves, through speciality network-control hosts with various levels of redundancy and reliability, and even to virtual machines in a data-centre environment. The success of SDN is largely due to flexibility and the ability to operate across a range of equipment from a variety of vendors. The flexibility of SDN and its inherent reduction of complexity [192] have been a significant attraction for environments where network functionality was a side effect of combining routers, switches, and middleboxes from a range of different vendors. In an SDN, the complexity becomes software within a controller, cutting the cost of network equipment, while providing more efficient utilisation of network capacity and more effective delivery of network functions.

By replacing proprietary network hardware with *white box* switches supporting an open standard, SDN enables developers to compete by offering network functions as *SDN apps*. The vision is that SDN apps can be developed, distributed, downloaded, installed, and used in much the same way as mobile phone apps. An example SDN app is a load balancer, distributing incoming web traffic among a number of servers on a tenant network. In a large multi-tenanted data centre, this might be provided as a service by the centre operator; but tenants might also buy such an app from their vendor of choice and run it on the controllers of their tenant network. This is good for competition and innovation, but quickly raises the question of access control. We need to prevent a range of attacks, as Alice's load balancer can interfere with Bob's, or with some other functionality in Bob's tenant network, or even with the underlying data-centre fabric itself.

Current SDN systems are designed to support multiple concurrent apps. However, when a controller runs multiple apps within a single network it assumes orthogonality or failing that, a strict (pre-declared) ordering [125]. The sharing of stateful information, resource and control among different apps within the same SDN fabric, or indeed instances of the same app operating on different (virtual) networks within the same SDN fabric, remains an unsolved problem, and is the problem that motivated the work reported in this chapter. It has been ignored until now because early SDN deployments have been commonly in the data centres of the big service firms where everything is under the control of the same network administrators, the deployment environment is relatively homogeneous and any conflicts that do arise can largely be resolved by *ad hoc* mechanisms, such as reference to the lowest common manager. Additionally, within such deployments there are relatively few apps and those tend to have been written by highly-skilled engineers.

However, we imagine this industry taking a similar path to mobile phones, with app stores offering a multitude of products. The many tenants of a large data centre might each be provided with their own dedicated *tenant* networks and expect to run all sorts of apps for monitoring, load balancing, intrusion detection and new functions yet to be invented. We need only look to Android and iOS for some guidance on what we could expect from such policy enforcement mechanisms. Each app might have a manifest stating what kind of paths it expected to be allowed to set up, tear down, manipulate, or monitor; any policy conflicts might be resolved either when the app is first authorised (as in older

Androids) or when a capability is first used (as with iOS and the latest Androids). And just as a phone app needs to be explicit about what resources it needs to access (e.g., the phone, the contacts list, GPS location or the camera), an SDN app may only require access to a certain subnet, a limited tuple of ports and hosts or only to certain middleboxes and end hosts. However, it is no longer enough to assume that one app will not interfere with another whether maliciously or accidentally. The evolution seen in mobile app access control over the past ten years appears inevitable for SDN apps too.

In the remainder of this chapter we will articulate aspects of our vision for policy and control in such an environment, and what our experiments teach us. The rest of this chapter is organised as follows. In Section 3.2 we present a use case, of an airport network with several thousand tenants including airlines, service companies and government agencies. We develop from this a threat model for large multi-tenanted SDN networks. We assume that such networks are implemented using a hierarchy of controllers, each of which contains a number of virtual controllers that each controls a tenant network, and which issue commands through the hypervisor to the relevant switches. We then describe the separation requirements of three SDN apps currently available in the SDN app store. In Section 3.3 we describe AXE, our network access control engine. Our prototype is coded in Python to run with the Ryu open-source controller[1]. We then evaluate its effectiveness for several use cases. In Section 3.4, we observe that the rules our access control engine must enforce map canonically to the rules implemented in stateless firewalls since the 1990s. We discuss the implications; rather than controlling flows by filtering already-generated packets, it is faster, cheaper, and more flexible to prevent unauthorised flows being created in the first place. In Section 3.5 we discuss the extra functionality needed to resolve policy conflicts. In our discussion of related work (Section 3.6) we draw out many interesting parallels between the access controls and firewalls; their convergence in the SDN case brings together 50 years' worth of access control research with 25 years' worth of research on firewalls, which we believe will be a very fertile source of new research directions. Finally, Section 3.7 draws our conclusions.

3.2 Access control in SDN

Almost all the deployment of SDN to date has been in large corporate data centres (e.g., [89].) These are the easy use case: environments with uniform network function requirements, relatively homogeneous network equipment, a high level of operational skill and unified administrative control. There are also some islands of use in government, at Internet Exchange Points, and in local parts of corporate networks. The next challenge is to provide SDN advantages to more complex networks, particularly those with serious separation requirements. Examples include different government departments, or between

[1]`https://osrg.github.io/ryu/`

an investment bank and its retail banking arm, where regulators may require a *Chinese wall*. We must provide the necessary isolation.

In a complex network, the power to arbitrarily change connectivity will no longer be under the complete control of all or even most operators. As different departments or different tenants start sharing the same physical network, controllers will run external code whose goals represent different and possibly conflicting user interests. For a host of regulatory and dependability reasons, different users, departments, and tenants within the same network will need to be isolated.

As a worked example of a network requiring such isolation, consider a large international airport such as JFK, SFO or Heathrow. There, physical, business and regulatory constraints mean that multiple tenants must share the same physical network. For example, Heathrow airport has some 3000 firms sharing its network, with about 180 000 badged staff; in addition, millions of members of the public use WiFi access points. We will discuss this environment in much more detail in Chapter 4, but for now the salient points are that the tenants include airlines who compete with each other, some (such as El Al and Iran Air) being associated with states that are at war with each other. There are emergency communication services with which no tenant may be allowed to interfere, and shared services such as ACARS which are almost as important (ACARS enables airlines to communicate with aircraft and file flight plans). There are common broadcast services, giving information such as which aircraft is at which gate. Finally, there are government bodies such as the UK Borders Agency which inspects the passports of arriving passengers and needs access to classified systems.

The mechanisms currently available to provide isolation and indirection in an SDN are insufficient. At present, there are principally two tools: slicing and abstraction. With slicing, the host creates a partial view of the network and allows a user to work within it. With abstraction, the user does not know the underlying topology but can create logical switches which are then interpreted and mapped on to the actual physical network. We review each type in greater detail in Section 3.6. However, neither model can tackle the need for shared services and resources such as ACARS, a service which needs to interface with every airline. The complexity of control is not simply limited to a physical port; or a network tuple, we might also need to make access a function of time, as check-in desks may change allocation from one airline in the morning and another in the afternoon.

3.2.1 An SDN network threat model

In a pre-SDN network device the code that provides the control plane comes principally from the hardware vendor, and is customised by the host network operator using router scripts. These install or uninstall specific paths and operate at the level of an assembly language. With a complete understanding of the network, the network operator may exercise perfect control. However the usability is poor and maintenance is hard. It is difficult in practice to get the flexibility that customers want, and dependability is also an issue. The lure of SDN is not just escaping the lock-in of proprietary hardware, but the promise of using proper computer science abstractions to manage networks flexibly. But with this Eden comes the serpent. The network is potentially exposed to all the many things that can go wrong with software.

In particular, SDN potentially brings external software. Network operators set up a fabric of SDN controllers which in turn set up virtual networks for tenants, and the tenants start, run, and manage SDN apps. These apps may be written by third-party developers and are developed to be deliver a wide range of different network functions, including (with luck) many new ones we have not thought of yet. The operator wants their tenants to manage their own network resources and apps, and wants the fabric to provide suitable isolation so that they can do this without getting in each others' way. Figure 3.1 illustrates a typical SDN setup, while Figure 3.2 is a multi-tenant version.

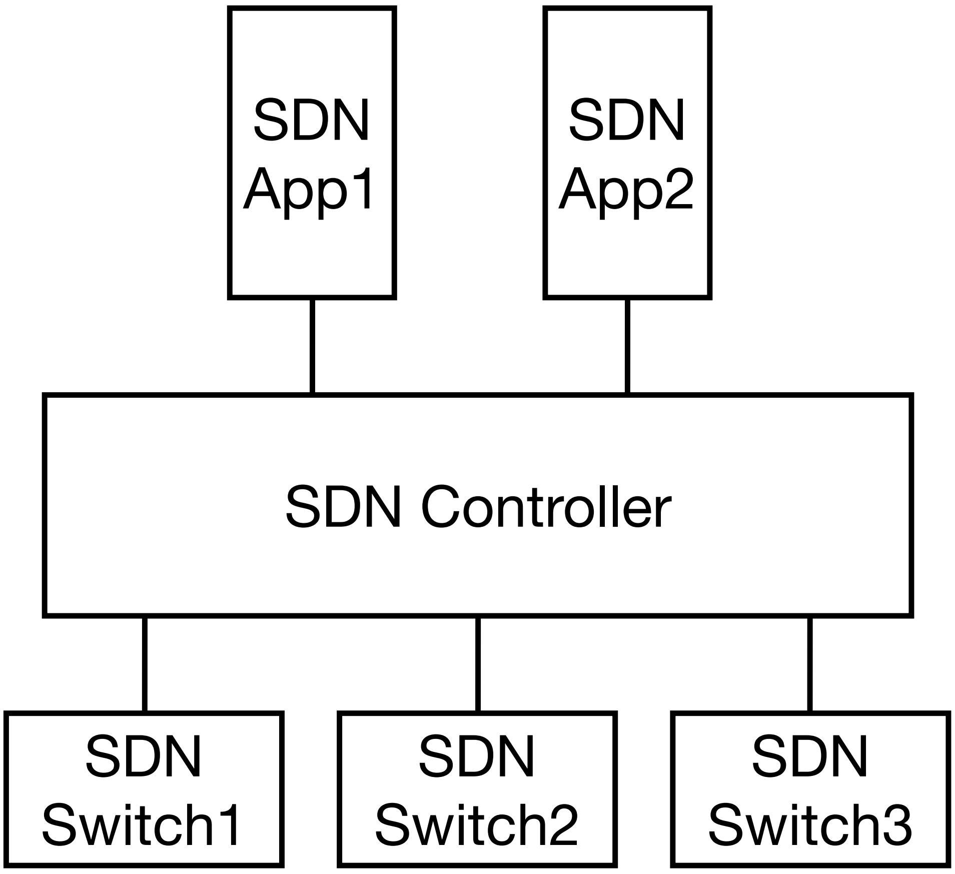

Figure 3.1: A typical SDN setup: a single network with two apps operating across multiple switches.

The switch from vendor code to third-party code is the key in our threat model. A

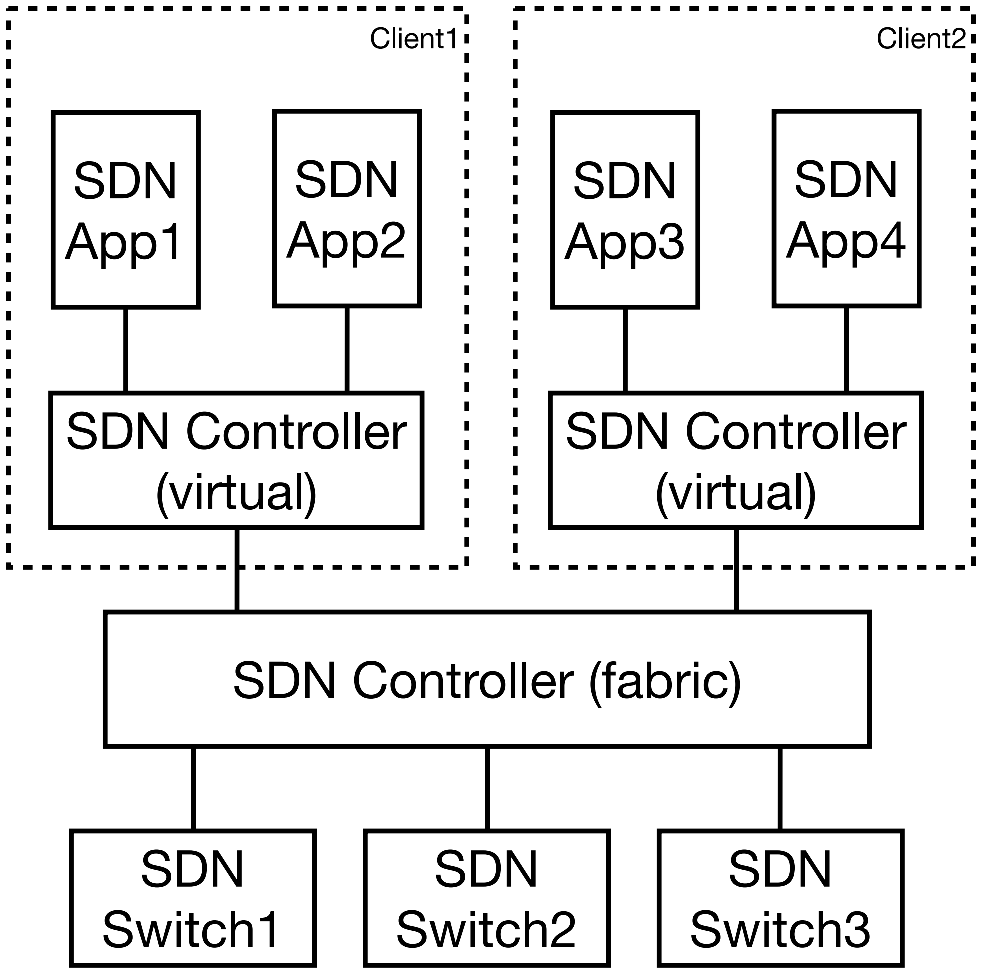

Figure 3.2: An SDN setup illustrating multiple tenants, each tenant similar to Figure 3.1.

traditional network permits any command entered into a router to have full access to the control plane. With SDN apps, we can no longer assume network control software to be rigorously tested, as it is no longer written by either the router vendor or the host network operator. As with mobile apps, developers will range in quality and trustworthiness. Even in the absence of malice, SDN apps may be harder to test rigorously because of the different ways tenants set up their networks, which will not generally be known in advance to the app author. They may contend for control of the same resources or generate conflicts in subtle ways; and as different developers have different incentives, they may be assumed to behave strategically rather than altruistically.

Current SDN controller implementations essentially assume trustworthy app code. They provide common helper functions and OpenFlow protocol translation, and will perform any operation the app requests. In a multi-tenant environment this is no longer appropriate, and code must be assumed to be malicious. In Needham's words, *'Going all the way back to the early time-sharing systems, we systems people regarded the users, and any code they wrote, as the mortal enemies of us and each other. We were like the police* *force in a violent slum.'* [4]

Apps, and tenants, need to be isolated robustly from each other, and from the network fabric. The host infrastructure will have to provide the appropriate isolation and sandboxing mechanism, like any other computer system shared by mutually hostile users. Our threat model follows from the use case of an airport network whose tenants include El Al and Iran Air.

3.2.2 Virtual controllers

We illustrate an SDN architecture where a physical controller may host multiple virtual controllers (VCs): one for each tenants network. The virtual controllers interface to the physical controller manipulating the pre-agreed tenant ports across the appropriate switches. Utilising the classical hypervisor definition, we extend it beyond a single machine and out to the network in its entirety [139]. In this way the VCs are managed by a network hypervisor; thus the controllers in each tenant network are virtual entities that can be spawned, replicated, and killed.

The network hypervisor enforces a reference monitor in the form of a shim between each VC and the physical network. This checks that the VC is abiding by the relevant access control rules and enforce them if required by blocking any forbidden OpenFlow commands issued by an app on that VC. This means that the virtual controller software does not have to be fully trusted; and because the hypervisor has a dedicated role, we can try to make its code small and simple so as to minimise its attack surface and to make it verifiable.

In a future large SDN network, we expect a hierarchy of virtualised controllers. At the top, the network owner and operator sets its policy. The rules are then propagated downwards to the lower-level hypervisors and virtual controllers. As a further conceptual model, imagine a network service provider whose customers are banks in a financial centre. On signing up a new bank, the operator will want to add new nodes to a number of the bank-controlled *internally-administered* VPNs (Virtual Private Networks) within its tenant network, to support access to a number of different stock markets, news wires and other services that may have special quality-of-service requirements or separation policies, such as between retail and investment banking. Similarly, if a bank signs up to a new market or service, the operator will want to extend it to the bank's relevant VPN within the tenant network, perhaps to multiple bank branches. Managing such complexity in a manner that is straightforward, transparent and (especially) auditable is one of the attractions of SDN compared to manipulating long batches of router commands, or making network configuration change requests of a centralised hosting facility.

In addition to providing security properties such as class of service and separation, our approach maintains the flexibility of the SDN approach. We can enable customers to ‘bring-your-own-controller’, and configure and use it entirely orthogonally of any other tenant, by choosing whatever VC software will best support their choice of apps. This freedom of choice makes it easier to attract and onboard new customers; the network operator’s trust does not reside in the VC but in the hypervisor. In the event that a user’s VC starts behaving abnormally or even maliciously, the network operator can kill or restart it with minimal impact on other users.

A capable controller hypervisor can therefore add real business value by providing SDN access control independently of tenants’ choice of VC. How can we go about building one?

3.2.3 Current apps in an SDN app store

We investigated the one currently functioning SDN app store, which is run by HP[2]. We surveyed the apps on offer and identified three distinct types: optimisation (e.g., a load balancer), monitoring/visualisation (e.g., a network visualiser), and host management (e.g., a honeypot or host isolator). We investigated what sort of access control each of these might require.

An optimisation app uses existing knowledge and real-time updates to provide high availability given existing network and host resources. There may be additional requirements such as affinity, so that an individual flow or host will see consistent behaviour. An example is a load balancer for web servers. Here, incoming requests are diverted to one of N web servers to share the load. The load balancer app receives TCP connections destined for a web server and forwards it to the server with the least load at the time, or perhaps to the server that that user most recently used, in the case that the tenant wants to use affinity to boost performance.

In the monitoring and visualisation case, apps gather traffic data from the network, run analytics of various kinds and report the results. What they monitor may be a specific service, traffic-breakdown by user or application, or more general indicators of network or system health. An example is a network topology visualisation tool that displays the traffic volume on each link in near real time.

Our third class is host management tools, which identify and deal with hosts that need some kind of special treatment within the network. For example, in a network with WiFi access points, an app may require a new host to log in via a web page and then be allocated to a specific VLAN, so that VLANs get reconfigured as machines move about. Another example is a malware containment tool that looks for hosts sending email other than through a designated mail server and responds by quarantining suspected cases of infection into a *walled garden* subnetwork.

Not one of these apps needs to be omnipotent. A load balancer needs to distribute incoming web traffic among a fixed set of devices; a network visualiser needs real-time

[2]`https://saas.hpe.com/marketplace/sdn`

read-only access to each relevant switch; and a WiFi manager needs to be able to set up and tear down specific VCs at relevant network nodes. If the load balancer has the ability to insert, modify, or delete *arbitrary* flow rules, it can easily issue commands to intercept or modify non-web traffic, such as SSH. An evil app could coordinate with a malicious intranet host to exfiltrate sensitive data. In a topology visualisation tool, even a single write or modify operation on topology is quite unnecessary. A host management app can put insecure hosts into a top secret network by giving it a corresponding VLAN tag; restricting an app to its own clearance level is necessary. The hardest case is the malware containment tool, but this merely needs to be able to move hosts to a specific subnet.

The common theme is that while apps make it easier to manage the network, there are currently no real bounds on what they can do. Current SDN security is at the same place mobile phones were ten years ago in the days of Symbian UIQ2, or where PC operating systems were with Windows 95. In these environments apps had to be trusted and the accidental or intentional malice of co-resident applications had to be accepted as an operational hazard. Such systems cannot cope well with malicious or even strategic users. Moving SDN to a modern world of app stores will take significant engineering, comparable to the work done on both PC and phone operating systems over the past decade.

3.3 An access control engine for SDN

3.3.1 Security model

Our concept of operations is basically a reference monitor that sits between the hypervisor and a virtual controller, able to look at the commands it issues and check if they comply with the policy set by the relevant authority, which is the network operator in the case of the underlying SDN fabric and the tenant in the case of the tenant's VC.

Initially we conceive of access control policy in terms of tuples of subject, object, action and context, a notion we will refine later. The subjects are individual apps; the objects are switches, subnets, individual IP addresses and port/protocol tuples; and the actions are reads and writes. We deal with higher-level objects such as OpenFlow messages by forming a mapping from a higher-level object to a pre-determined set of lower-level objects. Read-only access means only allowing commands to move data from the switch to the app without accepting a reply, while write access allows flows in both directions.

A classic way of defining policy is using an access control list (ACL) stating for each resource whether it may be read or written by its owner, a relevant group, or by all. A more general mechanism is Role Based Access Control (RBAC), where most users can be assigned to one of a small number of roles (e.g., bank clerk, branch accountant) and each role has a set of permissions. RBAC is more suitable for managing large complex installations but suffers in its original form from several drawbacks. First, there is not any mechanism to restrict access by a context such as time of day (e.g. 'El Al may run its VPN to checkin desks 64–70 from 6pm until 11pm each day'); it also does not offer the ability to separate instances of an app (e.g. 'The airline El Al can run its network to checkin desks 64–70 and the airline Iran Air can run its network to desks 17–22'). For this we need a more recent and more general model of RBAC, namely Attribute Based Access Control (ABAC).

ABAC provides a framework in which we can assign attributes to each subject (apps), object (network devices), action (read, modify, etc.), and context attribute (time of day, traffic load, etc.) to generate a permission decision. When roles are mutually exclusive, this essentially becomes RBAC. Where ABAC is more powerful is its ability to assign multiple possibly overlapping attributes to subjects, objects and actions. This extra power must of course be used with care. For example, a classical separation-of-duty policy subject might insist that all large money transfers be authorised by a branch accountant and a branch manager; if someone can assume both attributes, then the desired separation is lost.

We may also benefit from a separation of security policy from mechanism. In terms of mechanism, we believe it may be convenient to store policy as ABAC rules but implement it at runtime as capabilities, a typical one being 'app A is allowed to set up flows reading traffic from nodes in set X'. We will typically want to set up permissions in one or more central systems that manage the ecosystem of virtual networks; access policies then have to be transmitted to remote locations where we will typically have a set of virtual controllers running on a single hypervisor and managing a set of switches. At that level, it is convenient to give a set of capabilities to each VC and have the supervisor monitor their use.

Using capabilities has an added advantage: we can provide tenants and other network users with a way to manage their own app permissions or to delegate them to another user. A user running an app she does not fully trust can delegate a subset of her own capabilities to it, so as to limit the damage it can do to her own tenant network should it turn out to be buggy or malicious. For example, she might set up a subnetwork with its own VC to test new apps, or test them on one of the branches of her business.

There are a few vectors that are out of scope of the threat model. Similar to other access control and firewall implementations, the policy itself needs to be managed. A detailed policy language may grow to be very large and difficult to manage by a human. A good change procedure and user interface is needed to make sure no subtle conflicts occur within one policy. Another vector is bugs within the reference monitor. Although this can be addressed by keeping the code base of the reference monitor sufficiently small and thus easy to verify for correctness. A small code base is also easy to update without possible side effects once a bug is found.

Another class of attacks that we do not cover is malicious apps. Even though we can stop malicious apps from affecting others, they can still do damage within resources they can legitimately access. For example, a CDN app can insert malware to serve to users without needing to acquire extra access permissions. A more proper place to address this attack is perhaps at a higher layer, such as the application layer, where monitors and defence mechanisms are more effective in catching these activities. Once misuse of trust is discovered, removing the offending app is simple.

3.3.2 Design

To explore the design space for such an access control engine, we implemented AXE. Our AXE prototype runs as a shim between a virtual controller and its switches. The engine interacts directly with the underlying switches and forwards, in either direction, only those command messages the policy allows. The advantages of implementing the access engine separately instead of modifying an existing implementation of an SDN controller are first so that it is compatible with all controllers, whether open source or proprietary; and second, that the access control code is small enough to be closely scrutinised. The shim, running as a separate OS process, thus acts as a classic reference monitor.

We put each controller and any apps that it runs into a virtual environment. In our implementation we use Linux namespaces so that each controller has a separate set of tenant network interfaces, disconnected from the underlying network and monitored by the access control engine. For stronger separation (including for example of the memory and file system), we would put them in different virtual machines. In the terminology of RFC 2904 [46], the access control engine is both the Policy Decision Point (PDP) and the Policy Enforcement Point (PEP). In what follows we will describe the PDP as the 'policy component' and the PEP as the 'capabilities component'.

There are three main subsystems in our prototype: a policy parser which works with a simple policy language that understands OpenFlow, a decision component that loads a policy and handles incoming requests, and a capabilities part that issues and validates capability tokens along with any necessary cryptographic operations. Requests are routed either through the policy component (for requests without capability tokens) or through the capabilities component (for requests with capability tokens). In order to ensure a consistent response, there should only be one policy decision point in a network (or part of a network with a single jurisdiction). This policy component consists of the parser and the decision logic. The capability component is replicated as needed as the network scales, and derives its decisions from the policy component via capability tokens. A diagram of the access control engine is presented as Figure 3.3.

The policy component is where the network operator interfaces with the access control engine. We devised a simple policy language based on a combination of keyword and value mappings. For now we work within the keyword space of OpenFlow protocol, but

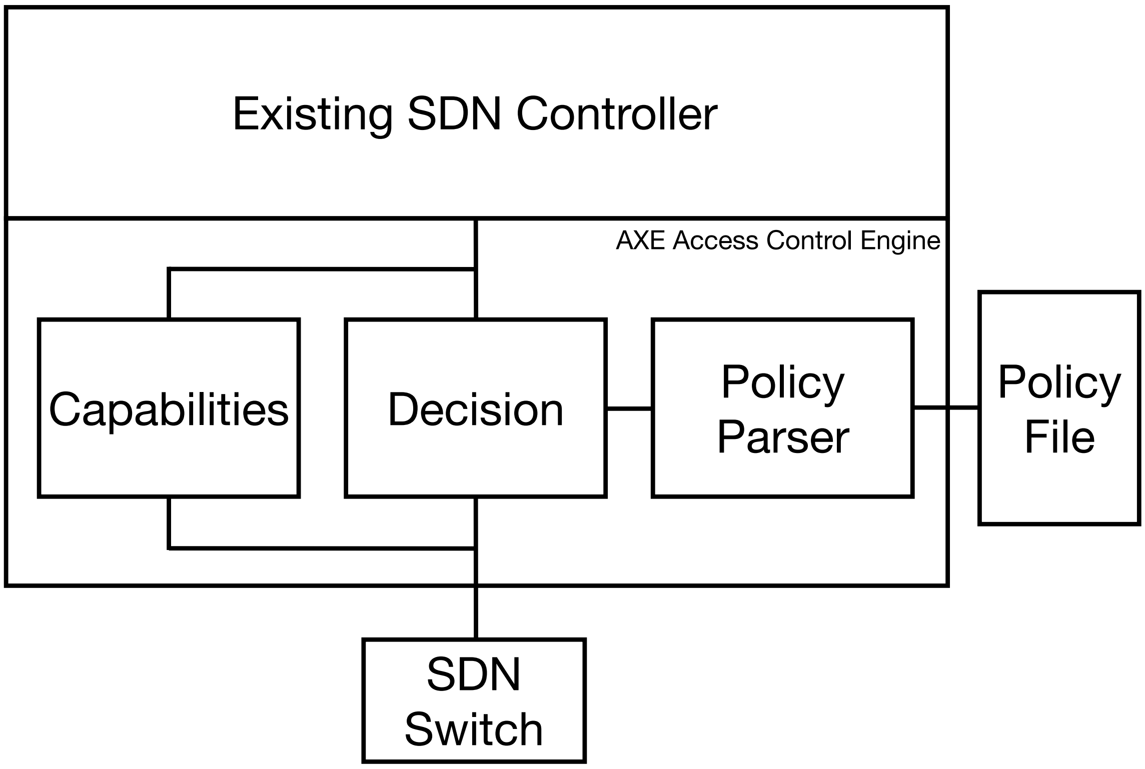

Figure 3.3: The three main components of the access control engine. The parser interacts with network operators through policy files. The decision component and the capabilities component handle incoming requests depending on whether a capability token is present.

the policy language is protocol-neutral and can accept other protocols as well. A policy is composed of rules, where each rule is a series of matches with a decision of `permit` or `deny`. If (and only if) all of the matches in one rule are satisfied is the decision used. Policies default to deny if no matches are found, resulting in a white-list approach..

Apart from OpenFlow keywords, the policy language also supports some overloaded keywords that have an expanded meaning. For example, `ip_addr` can be expanded into matching either `nw_src` or `nw_dst` in the transport layer. Similarly, TCP or UDP ports can match either the source or the destination as needed. This part is extensible to be application-specific, too. Suppose a network runs a service that relies on a specific set of traffic signature patterns; it can add its own overloaded keyword to expand into the more primitive keywords, to make the policy easier to understand with this customised shorthand.

The decision component is the main driver of the program. Upon startup, this component parses and loads the policy, and stores rules into a lookup table. For each app-to-switch or switch-to-app access request it receives, the decision component searches the relevant policy for matches and works out the decision, and if permitted forwards the command onwards. We can cache at the switch level (by installing a flow rule) or at the shim to improve performance for subsequent packets that match the same signature.

Requests with capability tokens are not handled by the policy component but instead by the capabilities component. The capabilities component is responsible for managing capability tokens available to apps on virtual controllers so that the policy component need not be consulted again until after a predefined expiration time. This is essential to give adequate performance and scalability; without it the policy component would be a bottleneck. The capability token can also be transferable by the app (or the VC operator, each of which has his own policy component). When the token expires, the request goes to the relevant policy component, and a new capability token can be issued if relevant, authorised and needful.

3.3.3 Evaluation

There are two areas of focus in evaluating our implementation of the access control engine: effectiveness and efficiency. First, it needs to be effective at what we set out to do and to make correct decisions. We seek to show that our prototype implementation can work in the use cases already discussed (Section 3.2.3) Second, we assess the performance overhead of our prototype.

The experimental setup was as follows. An instance of an unmodified Ryu SDN controller (v3.21) was run in a PC, in a separate Linux namespace, running a minimal app for basic switching functions. In a separate process, the access control engine was run and connected to both the controller and switch via separate network interfaces. Flow rules were not issued so every packet that the switch received was forwarded to the access control engine. An instance of Open vSwitch (OVS) (v2.3.1) was set up in the same PC. We used Mininet (v2.2.1) to drive the experiment, by providing and managing lightweight virtual hosts to connect to OVS. The network topology can be found in Figure 3.4.

Effectiveness

Let us start with a simple scenario and consider the following policy:

```
[id=1,name=policy1]
subject.app=p1,resource.OFPPacketIn.tp_src=80,decision=permit
subject.app=p1,resource.OFPPacketIn.tp_dst=80,decision=permit
subject.app=p1,resource.OFPPacketOut.buffer_id,decision=permit
```

This policy denies all traffic unless a packet has either a source port number or a destination port number of 80 in the transport layer (this implies the network layer is an IP packet). The `OFPPacketOut` command is permitted only when there is a `buffer_id`,

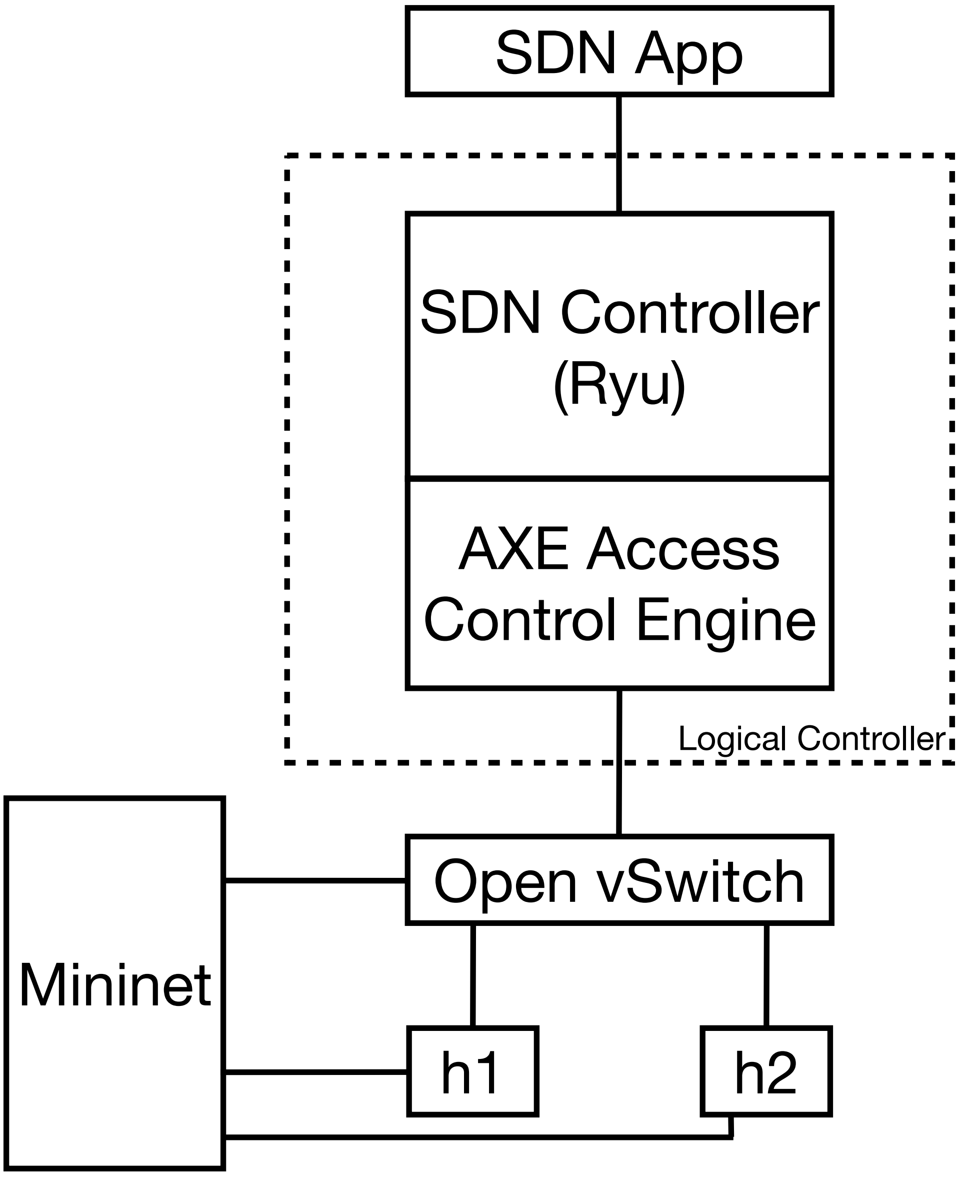

Figure 3.4: Topology diagram of the experimental setup. Mininet is used to setup the virtual topology and drive the experiment.

indicating the app is responding to a previous `OFPPacketIn` request. If the app crafts a new packet and pushes it out independently, it will be denied. Using Mininet, we generate two hosts, *h1* with IP address `10.0.0.1` and *h2* with `10.0.0.2`. From *h1* we run `nmap`, a network scanner, on *h2*. Specifically, we probe ten ports around 80 using UDP packets. The results are as follows:

```
Nmap scan report for 10.0.0.2
Host is up (0.015s latency).
PORT   STATE         SERVICE
75/udp open|filtered priv-dial
76/udp open|filtered deos
```

```
77/udp open|filtered priv-rje
78/udp open|filtered vettcp
79/udp open|filtered finger
80/udp closed        http
81/udp open|filtered hosts2-ns
82/udp open|filtered xfer
83/udp open|filtered mit-ml-dev
84/udp open|filtered ctf
85/udp open|filtered mit-ml-dev
```

Column 1 of the results shows the transport layer port number and protocol. Column 3 indicates the most likely service that is running in each respective port. The interesting result is in Column 2, where we see two types of states: 'open|filtered' and 'closed'. In `nmap`'s terminology, 'open|filtered' means no responses are received, in our experiment this is due to packets being dropped; the latter means a response from the remote host is received, and the port is accessible, but no applications are listening on it. This output shows that the controller and access control engine together successfully stopped all packets except for those that relate to port 80.

Let us now revisit the three use cases that are currently present in HP's SDN app store: optimisation, monitoring/visualisation, and host management. Since an optimisation app might focus on one service only, it is not very different from the example above. In fact, a complete policy for a web traffic load balancer (for both HTTP and HTTPS traffic) is fairly short:

```
[id=1,name=web_balancer]
subject.app=web,resource.OFPPacketIn.tp_src=80,decision=permit
subject.app=web,resource.OFPPacketIn.tp_dst=80,decision=permit
subject.app=web,resource.OFPFlowMod.match.OFPMatch.tp_src=80,decision=permit
subject.app=web,resource.OFPFlowMod.match.OFPMatch.tp_dst=80,decision=permit
subject.app=web,resource.OFPPacketIn.tp_src=443,decision=permit
subject.app=web,resource.OFPPacketIn.tp_dst=443,decision=permit
subject.app=web,resource.OFPFlowMod.match.OFPMatch.tp_src=443,decision=permit
subject.app=web,resource.OFPFlowMod.match.OFPMatch.tp_dst=443,decision=permit
subject.app=web,resource.OFPPacketOut.buffer_id,decision=permit
```

The second type, monitoring and visualisation, is not hugely different. In addition to identifying the resources it needs to access, these type of apps needs only read access. In OpenFlow, read access is reflected in the `OFPPacketIn` type of messages, where it indicates the arrival of a new packet. Previously we allowed `OFPPacketOut` messages to pass if and only if it contains a reference to a buffer within the switch, removing this rule will make sure that the app cannot reply at all to any packets. Continuing the example above, if we were to run a web traffic deep packet inspector app to comply with a police warrant, a policy might simply be the following:

```
[id=1,name=web_inspector]
subject.app=web,resource.OFPPacketIn.tp_src=80,decision=permit
subject.app=web,resource.OFPPacketIn.tp_dst=80,decision=permit
```

The third use case, host management, can vary in scope and the types of functions it needs to perform. An active malware scanner will need a dynamic corpus of signatures to match, so might have a DPI component that copies traffic to a host running suitable analytics, while an app that forces new hosts to log in via a web page needs to match specific MAC addresses and VLAN tags. The app in each case might have a small trusted component that is permitted to move individual hosts on to a specific VPN within the tenant network, such as one with rate limiting. A developer can also suggest a set of policy to be reviewed by the network operator at install time. This would be similar to the process of a user reviewing permissions on an Android app prior to installation.

Efficiency

A key performance metric is the extra overhead that the access control engine imposes on the controller. We break down this overhead into four aspects: initialisation, lookup, capability token generation, and capability token validation. In each experiment run we compare the time taken for running the Ryu controller alone to the time running Ryu with AXE in each of the four cases. We further vary the number of rules in the policy (from 10 to 100 000) to study the impact of rule-base size on performance.

The time it takes to initialise the controller and load the policy into the access control engine is presented in Table 3.1. Each scenario is repeated 10 times and the mean and standard deviation measured.

After initialisation, we measure the per-request overheads. A trivial app runs on top of the Ryu controller and tries to issue network access requests. There is no application-level caching in the app, the controller, or the access control engine. As we are only interested in the overhead introduced by the access control engine, the cost of executing app logic is not included in the measurements, nor is any network roundtrip should the switch respond after each command. In effect, we take our measurements at the controller-shim interface and the shim-switch interface. For capabilities, we use `SHA256` to first hash the message then sign with `PKCS1_PSS`. Again, we repeat each request 10 times for the mean and standard deviation measurements. The measurements can be found in Table 3.2.

There is a small but noticeable increase in the time it takes to process an access request. Fortunately, this increase does not seem to depend on the number of rules in the policy. This gives a worst-case latency of around one millisecond when establishing a new flow.

	Mean (μ)	Std. dev. (σ)
Ryu controller	16.88	3.05
AXE with 10 rules	1.25	0.16
AXE with 100 rules	2.49	0.22
AXE with 1000 rules	14.88	1.10
AXE with 10 000 rules	145.87	4.62
AXE with 100 000 rules	2561.87	431.58

Table 3.1: Initialisation times under various scenarios. The cases with AXE denote only the time it takes to start and load policy and do not include the time taken for the controllers themselves to start, as the initialisation stage for the controller remains the same whether or not AXE is present. All time units are in milliseconds.

	(1) μ	(1) σ	(2) μ	(2) σ	(3) μ	(3) σ	(4) μ	(4) σ
Ryu controller	0.10	0.03						
AXE with 10 rules	—	—	0.62	0.07	0.95	0.10	0.70	0.09
AXE with 100 rules	—	—	0.70	0.12	0.91	0.21	0.75	0.15
AXE with 1000 rules	—	—	0.57	0.06	0.80	0.09	1.09	0.16
AXE with 10 000 rules	—	—	0.65	0.12	0.88	0.17	0.96	0.18
AXE with 100 000 rules	—	—	0.65	0.10	0.84	0.17	0.99	0.18

Table 3.2: Packet processing times under various scenarios. All time units in milliseconds. Column cases: (1) is Ryu controller without access control, (2) is rule table lookup only, (3) is rule table lookup and generation of capability token, (4) is capability token decision only.

3.4 Access control and firewalls

After implementing AXE, the access control engine, we observed that in SDN an access control mechanism interposed between users and the underlying network equates to a firewall in the data plane. Rather than filtering packets in a separate box, one can just as well prevent undesired flows being generated in the first place. The same effects can be achieved by both approaches and operations can be mapped across them. This opens up a number of possible gains. At the level of engineering, we might hope to save money, improve performance, increase flexibility, coordinate management better and obtain higher levels of assurance by moving from a fleet of discrete appliances to a reference monitor in a network hypervisor. At the scientific level, we might hope to understand both firewalls and access controls better now that we have a concrete example of a system in which they are doing the same job. We will discuss the scientific aspects further in Section 3.6.

Let us review the three types of firewalls [17, 33]. The simplest form is a *packet-*

filtering firewall, which drops packets based on source or destination address or service, without maintaining track of context from one packet to the next. For example, a packet filter might allow a host to browse the web but not be a web server itself. The slightly more complicated *circuit-level firewall* tracks connection state (such as that of TCP) and distinguishes inbound connections from outbound ones; usually, unsolicited inbound TCP connects are denied unless an inside host has recently initiated a request. The most complex type is an *application-level firewall*, which understands packets in the context of one or more applications.

Historically, the application-level firewall was the first type to be built: in 1989 Mogul described, and in 1991 Vixie and Ranum delivered, the DEC Gatekeeper system which ran proxies for email, usenet, telnet, ftp, WHOIS and X windows, among other services; this was followed by Presotto and Cheswick's much more lightweight AT&T firewall which worked at the transport layer and finally work by Mogul, Carbridge, Chapman and others brought the benefits of packet filtering to wide attention in the early 1990s [84]. The tradeoffs between filtering at different levels are now fairly well understood and extend from firewalls to other middlebox applications from intrusion detection to surveillance. Below we outline how access control can arrive at the same outcome.

Packet-filtering firewall. Rather then choosing to drop or forward each packet as a firewall does, AXE can simply decide whether or not to forward each packet to the appropriate app according to the access policy. This decision can be taken at two levels: by the policy engine in the guest network, which enforces access control between a tenant's apps, and by the policy engine in the host network whose task is to separate guest networks from each other. If permitted, the flow may be set up using OpenFlow rules and the affected packets are forwarded onwards from and through the relevant switches.

Circuit-level firewall. The same mechanisms will support a circuit-level firewall but we need to provide some extra mechanism. For example, we allow a flow from an internal host A to an external host B and copy packets to an app C that then raises a request to allow a response flow from B to A. The implicit assumption of state being tracked is made explicit by vesting this state in an app, which must be trusted to the extent that it can change security state. Making this trust explicit is beneficial as it makes the trust dependencies more transparent.

Application-level firewall. In this case, the access control mechanism in the shim can parse traffic by port and forward it to the appropriate service proxy. This privileged app can perform deep packet inspection and may also be stateful. It is likely to function much like a conventional application firewall in that it will either drop or output packets (whether unmodified or modified) back into the data plane to be forwarded.

We observe that the main benefits to be obtained in assurance and performance terms are likely to be realised in the packet forwarding case, as this is the cleanest of the three and the one that can move completely from a network middlebox to an access control mechanism in a network hypervisor. In this case, we can capitalise on decades of access control research and experience, and we have a direct economic incentive to replace middleboxes with SDN access control. In addition to capital expenditure savings we might reasonably hope to make operational savings too; the ability to manage thousands of firewalls from a single control point, and to upgrade and reconfigure then at will, is an important objective of security managers with whom we spoke.

Once these savings have been banked, the SDN approach we sketch here can bring the further benefit that circuit-level and application-level devices can be reimplemented as SDN apps rather than requiring physical middleboxes. This should also bring improvements in capital cost, maintainability and flexibility, and permit different guest networks to make quite different arrangements, increasing both competition and flexibility. As a result, tenants could now consider deploying firewalls everywhere in the network [80], supporting a strategy of deperimeterisation for tenants who prefer this, while allowing other tenants to opt for reperimeterisation instead. However, delivering this sort of flexibility at scale raises some new problems — or, rather, an old problem which access control researchers have largely ignored up till now.

3.5 Policy conflicts

We have known for decades that policies can conflict; dealing with these conflicts is variously known as the multipolicy problem or the metapolicy problem [78, 79]. Sloman and Lupu discussed policy specification and conflict resolution in the context of programmable networks, which predate SDN [163, 164]. This has been discussed in the literature for a quarter of a century but so far there has been little incentive to work on a systematic solution. SDN may change that.

As a motivating example, consider two mobile phone apps: one for an emergency medical communications such as paging hospital staff, and one for banking. The emergency app requires access to GPS, voice and movement sensors, while the banking app forbids any other app accessing the microphone, accelerometer, gyro or GPS while it is running (for fear of side-channel attacks). This leads to a clear conflict. Suppose a doctor is doing his online banking when an emergency call arrives; what happens then? The desirable outcome is that the banking app is immediately terminated. But how is this to be managed in practice? With older versions of Android, permissions are requested in the app's manifest and granted on installation, so presumably a doctor trying to install this banking app would be warned that it would be terminated whenever his pager went off. That is quite hard to engineer, so a more likely outcome would be a warning that the pager and the bank were making incompatible demands, so the user either had to prioritise one of them or reject one of them outright. (As every philosopher knows, you cannot have an unstoppable cannon ball and an immoveable post in the same universe at the same time.) In the case of the iPhone and of later Androids, where capabilities are authorised on first use, things could get more messy still.

In short, policy conflict detection is not straightforward, and conflict resolution can be even harder. Both have usually ignored in systems built up till now, on the assumption that they can be handled using ad hoc means, such as by the user shooting one of the apps. For example, the NIST standards document on ABAC [81] mentions metapolicy briefly but offers no ideas about how actual conflicts are to be detected or resolved.

One obvious detection (and resolution) approach is through assigning priority levels to each app. But this is not always possible. For example, during the Cold War, a UK subject could either get a Top Secret clearance, or travel to a Warsaw Pact country such as the USSR or East Germany. People were not permitted to do both; a student trip to Moscow could exclude someone from subsequent employment in the defence or security parts of government (on the theory that the student might have been recruited by the KGB, or at least exposed to blackmail), while people with high-level clearances were forbidden from crossing the Iron Curtain. This is a stateful access control policy: you can go to Moscow or Cheltenham but not both.

This case is interesting for network access control as it is an instance of the Cascade problem [77]. If an EAL4 evaluated mail guard is sufficient to bridge a Secret system to either a Top Secret system or an Unclassified one, then whether a Secret system is allowed to connect to systems of the former type or the latter depends entirely on which type of system it connected to first. Such path-dependent security policies are not in general reducible to an ordering [4]. Cascade conflicts are also interesting because the state changes they entail are global and permanent rather than local and transient, unlike most reactive flow control mechanisms. We will discuss this in more detail in the following section.

It follows that a policy conflict resolution mechanism must in general keep track of which host has talked with whom. If it does not do so, then it can at most manage conflicts that are detected at install time and that can be resolved by an ordering. If the basic capability mechanism is going to be stateless for performance reasons, then the state will have to be managed at the appropriate policy control point, whether the guest network's or the hypervisor's.

3.6 Related work and new directions

Firewalls appeared in the late 1980s, with one of the spurs being the 1987 Internet worm which suddenly made network security salient [84]. As noted above, Mogul described the first implementation of a firewall in 1989 [126]. This firewall is an application-level proxy that allows an explicit set of services and denies the rest. Its drawbacks included needing to update the firewall for each new application and requiring client programs to be configured to use it. Cheswick the described a simpler firewall design for AT&T that worked at the transport layer [32]. However it still needed clients to be configured to use it and the performance penalty was still notable. IP-level packet filtering also existed but with various configuration difficulties; these were alleviated with the arrival of the socks proxy. Other early work described in increasing detail what firewalls could do and how to implement them [13, 127, 17]. Since then, research has focused on policy and deployment, such as constructing [72], analysing [122], and verifying [187] policies, or scaling firewall functions into a distributed system [16].

A particularly influential paper by Ioannidis and others described a distributed firewall designed by UPenn and AT&T [86]. As firewalls became mainstream, a security manager could struggle to maintain consistent state across hundreds or thousands of devices with complex and changing policies. Ioannidis et al set out to separate policy from mechanism, with the Keynote policy engine creating security contexts that are signed and distributed to policy daemons that implement them. They noted that such mechanisms might be extended to application level firewalls as well. Their system was stateless, and best described packet filtering, although commercially available firewall systems increasingly supported state, most notably to ensure that if Alice the insider sent a packet to Bob outside on the Internet, then Bob would be able to send reply packets to the appropriate port.

Firewall mechanisms developed in parallel with other ‘middlebox’ services such as intrusion detection, of which early and notable examples were Snort [144] and Bro [136]. While firewalls exclude activities that are known to be unnecessary and believed to be possibly malicious, intrusion detection systems police the greyer areas of activities that we do not want to exclude a priori but which bear monitoring and reporting. The technologies range from signature-based systems that look for ‘known bad’ patterns, to behaviour-based systems that look for the unusual [28, 114]. Regardless of the internal logic, intrusion-detection systems look to the network like stateful firewalls; they accumulate information about network state and may then make a decision to terminate a flow, with or without human intervention. And then there are complex systems such as the Great Firewall of China; Clayton reports it to consist of a packet filter that blocks known subverse IP addresses, coupled with an intrusion detection system that uses deep packet inspection to look for forbidden substrings and then tears down the offending TCP connections [38].

We noted that stateless access control and firewalls perform the same functions: access control specifies who can access what file, and firewalls do the same to flows in networks.

Academic papers on both access controls and firewalls tend to disregard state, but real-world systems usually cannot. In banking it is common for large transactions to be authorised by two or more people; this was formalised in the Clark-Wilson model, but without much discussion of the mechanics. In practice, the security state may consist of a file of transactions authorised by Alice, and awaiting confirmation by either Bob or Charlie. This state is local, and transient; we do not want to carelessly replicate the file lest the same transaction is authorised by both Bob and Charlie and mistakenly paid twice.

Similarly, if Alice is an analyst at an intelligence agency and sends an http request to a subversive website maintained by Bob, she wants the firewall to permit return packets to come from Bob to her machine only, along that path only, and for a short period of time only. The great majority of other uses of state in both access controls and network flow controls are also local and transient, from commands such as `sudo` to affinity requirements in web traffic load balancers. Keeping access controls stateless and putting security state into apps is more or less what existing systems do anyway; we implement dual controls in the real world by having an app that inputs unauthenticated transactions and outputs payment instructions that have been authenticated by both Alice and Bob. The SDN case may help by forcing us to think about such mechanisms carefully, and to reason about them more generally.

There are however a few cases where security state must persist and propagate globally, such as the Cascade problem discussed earlier. If a Secret network has been connected to the Internet via an EAL4 mail guard, even once, then it might be forever forbidden from being similarly connected to a Top Secret network. This might seem a contrived and specialised example, but persistent security state changes are likely to become more common with the adoption of SDN.

SDN app security and access control has already attracted some interest, but in a flat role-based model. FortNOX [140] introduced a security-oriented controller that puts apps into one of three roles (administrator, security, and other) and assigns permissions accordingly. The rationale is that apps should receive access privileges according to their functions. It also uses these roles to help resolve conflicts when conflicting flow rules are detected, and an app with a role of a higher privilege overrides another with a lower privilege. However the conflict resolution process does not address two apps within the same role. Many of these ideas were developed further into SE-Floodlight [141], a modification of the Floodlight controller incorporating app authentication, role-based access control, conflict resolution, and auditing. Role-based access control, however, is single-dimensioned and does not offer the flexibility of matching on packet fields. It also does not consider environmental variables such as time of day, or a combination of packet fields and these external factors. While a new conflict detection algorithm is presented, the resolution process is still the same as its predecessor. PermOF [176] provides 18 different permissions for apps to interact with the underlying network, which are then checked and enforced. It also calls for a shim between apps and the OS, but is more restrictive by requiring apps to be reactive-only. The fixed set of permissions also does not allow for an easy upgrade when OpenFlow is expanded or if another protocol is to replace OpenFlow. AXE is further different in allowing a network to be divided into subnets and managed separately. OperationCheckPoint [152] builds upon PermOF including using its permission set, but provides an implementation to also allow permission checking to be scalable. FlowGuard [80] is an SDN firewall framework that detects rule violations and offers a graceful way to resolve conflicts by keeping the non-conflicting parts in place. The detection method works by converting firewall rules into two disjoint partition spaces, each containing all packets that should be allowed or denied under the current rule. This approach does not address the lower level details of how policies are enforced, but can potentially be complementary to our AXE design.

From a different angle, VeriFlow [100] performs access control checks with network invariants as its policy framework. VeriFlow checks control plane update messages before they arrive at the forwarding plane, and intercepts them if network invariants are violated, thus having similar powers and control as AXE's shim. It provides efficient lookup algorithms to make sure that near real-time checking can be achieved, minimising extra overhead. Through checking for network invariants, we no longer need to give privilege levels or prioritise among flow rules with this approach. Similarly, Flover [165] can formally check if OpenFlow rules abide a network policy using an SMT solver, and can guard against bypassing of security properties even in the presence of `goto` actions, which are trickier to handle.

The need for consistent updates was discussed by Reitblatt et al [143]. They point out that SDN networks need to propagate state consistently when network upgrades occur, and describe a prototype system to support powerful multi-switch updates with strong semantic guarantees. This is in the context of a research programme to get away from low-level install/uninstall operations towards a higher-level language. (OpenFlow is more a macroassembler than a high-level network-function language.) They therefore provide two-phase commit as a primitive. Their policy mechanisms are flat, but they do mention as extensions the possibility of doing pure extensions, retractions or subset operations, which can be used to scale their update strategy to large systems, for example when we want to isolate a host that is infected or that is being attacked [130, 151]. And network upgrade is not the only new case where SDN will require access control coordination across controllers and switches; another example is when implementing multipath TCP. However upgrade can serve as our test case for security state propagation.

This is important, because as SDN moves out of the data centre, SDN controllers reacting to rules on a network flow-by-flow basis will be particularly attractive. At present a typical data-centre SDN system may preassign all flows — routes within and among data centres. Assigning local and transient new flows can often be left to local controllers and apps, implementing a simple circuit-level firewall as discussed above; another possible mechanism is out-of-band, as where VMWare supports an instrumented API to generate connection data for guest operating systems. However in the long term we need robust mechanisms that work with virtualisation so that, to the greatest extent possible, the same controllers and the same access policy mechanisms can be used on tenant SDN controllers as on the underlying fabric controllers. And we also need to consider whether apps themselves will run in virtual machines; Shin et al. report doing this to prevent apps crashing to bring down the controller on which they run [161]. Such issues have been studied extensively in the world of operating system access controls and it is natural to expect that the robust sandboxing mechanisms already being deployed in phones and elsewhere will in time see adoption for SDN as well.

In passing, we note that Network Functions Virtualisation (NFV), launched in 2012 [34], provides an alternative approach to tenant networks that can be based on legacy network technologies and virtualised hosts but can provide a migration path to SDN. As this overlaps with SDN, it raises many of the same issues; there, too, middlebox functions are likely to be implemented as apps.

Finally we note that many of the problems with which access control researchers have wrestled over the past fifty years are potentially applicable to SDN, ranging from tranquillity principles through non-interference to covert channels. In some respects the problems may be simpler; if the TCB can be made stateless, or nearly so, and security state located in apps with defined permissions, then hopefully the TCB can be made much smaller. In other respects it may be harder, if (as we argue) we have to deal systematically with policy conflicts for the first time.

3.7 Conclusion

In this chapter we presented AXE, an access control approach for SDN that provides an attribute-based access control model for SDN controllers to manage apps. We assume apps cannot be trusted, either as they may be designed to behave in a strategic or potentially malicious way, or that they may one day be compromised. From this threat model it follows that the controller must be ready and able to stop apps when they start misbehaving. Our model is sufficiently flexible to allow policy control to be imposed on SDN network tenants by treating their control requirements as apps, and we outline an approach that permits a hierarchy of such virtual controllers to also deploy policy control for SDN apps operating within their tenant-networks. In contrast to past work modifying existing controllers, AXE is a new approach, implementing the access control engine as a reference monitor. This permits a low complexity of implementation and opens the door for operational correctness to be validated. In our evaluation we demonstrate an unoptimised prototype that has modest fixed per-packet overheads; these overheads are independent of the complexity of policy. Our prototype also implements a mechanism of capability tokens permitting higher performance through more sophisticated rule handling and rule caching.

Finally, in the course of this research we realised that access controls in SDN virtual controllers map canonically to stateless firewall rules. This allows us to lay fifty years' worth of access control research alongside twenty five years' worth of firewall research, look for the gaps, and work out directions for future research. We believe that this confluence between two related research fields may prove very fertile soil for future research, as well as helping us to solve a real engineering problem that is holding up the spread of SDN from the data centres of the large service firms to multi-tenanted networks and to more general use.

Chapter 4

Scaling SDN and access control in real cyber-physical systems

As we already noted, SDN is already deployed in data centres, where it offers advantages of cost and flexibility over traditional routers, within a single controlling organisation with unified administration. For SDN to fulfil its potential, it needs access control mechanisms to accommodate mutually mistrustful principals. If this is done right, there could be a third class of potential benefits, namely security. In the last chapter we described a prototype access control system, AXE, which implements attribute-based access control in virtual controllers (and/or the hypervisors on which they run) so as to provide essentially the same separation properties as stateless packet-filtering firewalls. However the next question that arises is whether a lab prototype can reasonably be expected to scale to a major operational system. In order to do so, we will return to our use case, of a major international airport, and examine it in more detail, so as to get a more realistic and detailed threat model. We will then investigate how an access control system such as AXE could scale to such an environment.

4.1 SDN deployment: cyber-physical systems

As part of the work for a DARPA seedling project [70], we identified five possible deployment scenarios that can benefit from SDN. They are data centres, large ISPs, Internet exchange points (IXPs), cyber-physical systems, and battlefields. In this chapter we analyse the security aspect of cyber-physical systems and how an SDN deployment can better secure them.

A cyber-physical system is an interacting network of devices and the physical environment where they exist. These are often found in infrastructures where there is more than networking, such as electricity control systems, smart factories and warehouses, or airports.

A large international airport is a complex physical environment that relies on robust networking for many of its daily operations, and is so constrained that the tenants cannot be allowed to do their own networking. Thus a single network has to support and interact with with a large number of diverse users. We could discuss this in the context of any large airport (JFK, O'Hare, LAX, Hong Kong) but for simplicity and concreteness will refer to this setting as *Heathrow*.

4.1.1 Operators and stakeholders

Heathrow's infrastructure serves many organisations ranging from its operator BAA through the major contractors and airlines to small franchise operators. Some organisations are arms of the British state and deal with classified information, such as the Borders Agency and HM Revenue and Customs; others are arms of other states, such as El Al and Iran Air. The only common background check performed on UK staff is that they are not on various blacklists of terrorist suspects or convicted of serious crimes; in the event of the foreign employees of foreign organisations, real checks are not possible.

All of these organisations must use a single network; it would not be acceptable for 3000 firms to have their own cabling and other infrastructure. The network must also serve the general public, via WiFi hotspots where passengers can pay for service via dozens of providers and roaming agreements; it also serves emergency services by supporting mobile radio of various kinds. There are more specialised shared services, such as a broadcast data channel which announces which plane is at which gate, and an ACARS system that relays data to aircraft. Many of these shared services are critical; without ACARS, for example, airlines would take an extra 15–45 minutes to turn round aircraft as flight plans would have to be filed manually.

These numerous users and stakeholders need the network for different purposes and thus have different requirements. We identify four main groups of users: infrastructure-owner, main customers, auxiliary customers, and general public.

The owner, or its lead contractor, lays out the network infrastructure. Many things must be got right to give users the dependability guarantees they need. The owner is the key decision-maker on network requirements, design, installation, and maintenance. Without proper planning, a failure at this layer will affect the whole operation. In the case of Heathrow, the government also has a vested interest as continued operations are of national importance. An attack that significantly disrupted airplane or passenger movement could have macroeconomic effects even if it did not lead to casualties.

The second group is the 'anchor' tenants such as the major airlines, and the police, border and customers services. These customers may have similar needs, but the airlines (for example) are in competition with each other. In more extreme cases two state-owned airlines may represent two countries at war — so the case for secure separation between users is strong. These customers not only need to connect their machines and devices together at the location and to servers offsite, but also to take over shared facilities on a temporary basis (as when an airline uses a dozen check-in desks for a few hours before one of its planes departs). They may want the network to roll out special technologies to support their operations. Both airlines and law-enforcement agencies need the network to run their core business: if an airline cannot get through to its booking system (typically run by a third-party contractor such as Amadeus) then it does not know which passengers may board which plane and its operations cease (or at best continue in highly-degraded form, relying on paper documents that more and more customers simply do not carry).

Auxiliary customers such as restaurants, shops and third-party wireless Internet providers are not critical to the airport's operations, in the sense that passengers could board and planes could fly without them. However the rent they pay is a substantial part of the airport's operating income, and if their network service were interrupted for long, there would be consequences. For example, high-value stores could not operate profitably without the ability to do online payment card transactions. Furthermore, as auxiliary networks are used by many staff who have had only cursory background checks, they can be an entry point for malicious actions.

The general population is the largest group of users by headcount.They interact with the network as passengers, hotel guests, shoppers, and even as criminals — from baggage thieves to people who set up bogus WiFi connections to conduct phishing and pharming attacks. So we have to consider general public network access as a possible entry point for tactical or strategic attacks.

4.1.2 Technical operations

The applications and services in a cyber-physical network demand several properties: low latency, high bandwidth, resilience, virtualisation, and flexibility. Each application and service may require one or more of these properties to function.

Latency is most important in infrastructure-type functions. Timely feedback of individual sensors throughout the building, such as CCTV cameras and fire alarms, is essential to continue operations. Emergency radio equipment is also critical. Latency is achieved by good network design and the ability to prioritise critical traffic.

The many staff, firms and passengers also use a lot of bandwidth. Advertising, CCTV, flight information and business transactions occupy a large portion of the available bandwidth.

Resilience, too, is important as an airport depends on a large number of applications, from obvious ones such as checkin and baggage handling to the less visible such as the RFID locks on thousands of doors, the ACARS systems used to send and receive data from aircraft and the classified government systems that connect border agency staff with visa and blacklist systems.

Virtualisation is important for individual airlines, shops and other airport tenants as they need to connect to their corporate networks over the network operated by BAA. Virtualisation must also support separation, to prevent virtual networks interfering with each other and denying service. It is not desirable for any of the airport's tenants to be able to deny service to critical flight-operations or national-security systems, or to any other tenant.

Lastly, the network needs to accommodate the many new devices and protocols that will emerge over the lifespan of a business such as an airport (which has been in operation for over 60 years). The deployment of new networks and services must be possible without interfering with legacy communications and applications.

In order to give some idea of the scale, one single terminal at Heathrow has [42]

- 1000 fixed and 500 mobile video cameras (10 Gb/s)
- 500 displays (10 Gb/s)
- Biometric scanners (10 Gb/s)
- Private and Public Fixed and Wireless LAN (20 Gb/s)
- Cellular services (10 Gb/s)
- TETRA and private radio (0.5 Gb/s)
- Passive RFID (0.2 Gb/s)
- Active locatable RFID (5 Gb/s)

The assumed aggregate peak rate is in the tens of Gb/s.

4.2 Security analysis

4.2.1 Vulnerabilities

At a high level, we are concerned with both strategic and tactical attacks. Strategic attacks involve an attempt to close down or cripple the airport, perhaps as a service-denial move in wartime. In the Heathrow case, apart from the most obvious case of cutting electrical power for a complete network outage, a targeted attack on the Amadeus passenger booking system can result in total chaos as airline employees need to fall back to paper passenger lists, while a takedown of the ACARS system can delay plane turnaround time by forcing flight plans to be filed manually.

Tactical attacks have smaller goals. Mutually mistrusting parties such as Iran Air and El Al may try to sabotage each other's operations from within the same network. In fact, we find this mistrust everywhere: not only do participants of the same function not trust each other (as with airlines); different applications within the same network also distrust each other. The network delivers traffic for safety critical systems (e.g., emergency radio), operational critical systems (e.g., ACARS), and public WiFi, all with the same physical infrastructure. In fact, a simple attack would be to use as much bandwidth as possible, hoping critical services are degraded or denied.

Gaining physical access to devices can potentially allow an adversary to replace a device with one of his own. In this attack, the attacker can do much more than eavesdropping and can actively participate in the network. By taking over a device, such as a switch, it is possible to cause havoc, such as by repeatedly advertising and tearing down bad routes, leading to a network collapse. More subtle attacks can involve altering network configuration so that a competitor's traffic passes a machine controlled by the attacker, to facilitate eavesdropping, traffic analysis or targeted service denial. If carried out competently, such an attack may be hard to detect. SDN can create many new vulnerabilities, if deployed thoughtlessly, but it also has the potential to support robust virtualisation that would make such attacks harder.

Similarly, a software vulnerability exploitable over the network can potentially allow an attacker to take over one or more routers or other devices. Such exploits can potentially allow access to the device to be escalated into an attack on the network. Standard software vulnerability classes and attack surfaces apply here, as routers and switches are based increasingly on commodity software and standard attacks may be used in the time window between a vulnerability being disclosed in Linux or FreeBSD and its being patched in network hardware based on this software.

What we do not cover in scope are the parts of the network that do get compromised. The goal is to ensure the overall resilience of the infrastructure, and making sure attacks cannot easily propagate sideways within the network. However, hardware, subnet, app, and other components that are successfully attacked will become compromised, along with any information and access permissions they own. Therefore, we accept partial outages or degradation of service levels, but aim to keep it small and separated from the rest of the network. In a more extreme scenario, it is possible for a sophisticated attack to take over a large portion of the network piece by piece and reconnecting them together, rendering the surviving (small) part effectively handicapped and succeeding in controlling a major part of the network despite our security controls.

4.2.2 Requirements for a resilience architecture

The required resilience of the network is to ensure network connectivity as well as the quality of service for a number of critical services in decreasing priority order, starting with safety and emergency communications, down through flight operations, through airline systems to the services offered to the general public. Further virtual network separation is desirable to protect airlines against interfering with each others' service, whether accidentally or otherwise. The network managers must be able to measure and monitor changes and faults and modify network topology and behaviour in response as appropriate.

4.2.3 Organisational requirements

Apart from technical and operational requirements, there are three linked business requirements from the network operator: the need for abstractions, the need for automation, and the need to reduce costs.

As already noted, current network management tools are inadequate, being based on router command lines that differ from one vendor to another and which do not support the atomic, consistent, isolated and durable transactions which network operators really need. The lack of appropriate abstractions entails a pervasive lack of contextual information which is not just inconvenient but can easily result in operator errors. Abstractions can hide the parts of the network that are not relevant to the task in hand; this is just basic computer science, and applying it to networking is one of the big promises of SDN.

Abstraction will also support better automation. Current network technologies leave too much room for human error. Some tasks, such as adding a new airline or shop, may be repeated many times and should become routine; dependencies with local implementation detail must be better hidden to make this simple and dependable.

The main driver in day-to-day operations is cost. The air transport industry operates on tight margins; in Heathrow's case the operator is heavily indebted following a takeover and was forced to divest other London airports by the competition authorities. Investments in new technologies will only happen if there is an unavoidable regulatory mandate, or to save money.

4.3 Hierarchical design

Our detailed threat model therefore assumes physical compromise of devices, along with associated attacks that can be done by having physical access. It follows that some devices (the ones 'in the field') are unsafe. Once an attacker gains physical access to a device, he can open, modify, remove, or replace it. It is reasonable to say a number of switches are compromised at a given time, and sometimes controllers at the bottom of the hierarchy are compromised too (since they are deployed near switches). We also assume the communication channels connecting an unsecured switch to be insecure. If an attacker can get access to a device, he also has access to its cabling.

Imagine an ISP needing to change a switch at a customer's site by ordering the equipment from a contractor. There is no guarantee of security at any of the steps from factory and shipping company to contractors and subcontractors and eventually to the customer. Even the customer may modify the equipment if there is incentive to do so. It is not just a matter of occasional access by secret policemen at points of presence in less well-governed countries; recent Edward Snowden files reveal that NSA tampered with Cisco supply chain of hardware exported to other countries[1].

In a time of conflict, the opportunity would exist for an opponent who had taken over one of the routers to use it to cause havoc, by inserting false rules, or removing and replacing legitimate ones, and thus generally disrupting the switching fabric. If a network has a thousand routers and any one of them can disrupt it, as is the case today, then an attacker need compromise a few routers for the whole network to become vulnerable.

4.3.1 Topology

We argue that a move from peer routers and switches, any of which can cause equal havoc if compromised, to a tiered system of switches and controllers with appropriate access control, can result in a more secure and scalable architecture for SDN. The lower-level controllers and switches cannot all be protected; they are involved in running the local nodes but have a limited view of the whole network. The upper-level controllers are responsible for infrastructure tasks, only interact with lower controllers, and can be better protected physically. Overall, the topology looks like a tree with central controllers at the top, and local controllers and switches as leaves on the bottom (the topology may actually be a multitree once redundancy is added; as we show below).

A classical network setup [64] adapted to SDN, as illustrated in Figure 4.1 and currently deployed in data centres, might have a bottom layer of 1000 switches, with each ten switches driven by a level 2 controller, every ten level 2 controllers driven by a level 1 device, and the ten level 1 devices coordinated by a master controller. Thus if we can arrange things so that only controllers can cause widespread outages if compromised, the number of critical components is reduced by a factor of ten. If we can further arrange things so

[1]http://www.theguardian.com/books/2014/may/12/glenn-greenwald-nsa-tampers-us-internet-routers-snowden and

http://arstechnica.com/tech-policy/2014/05/photos-of-an-nsa-upgrade-factory-show-cisco-router-getting-implant/

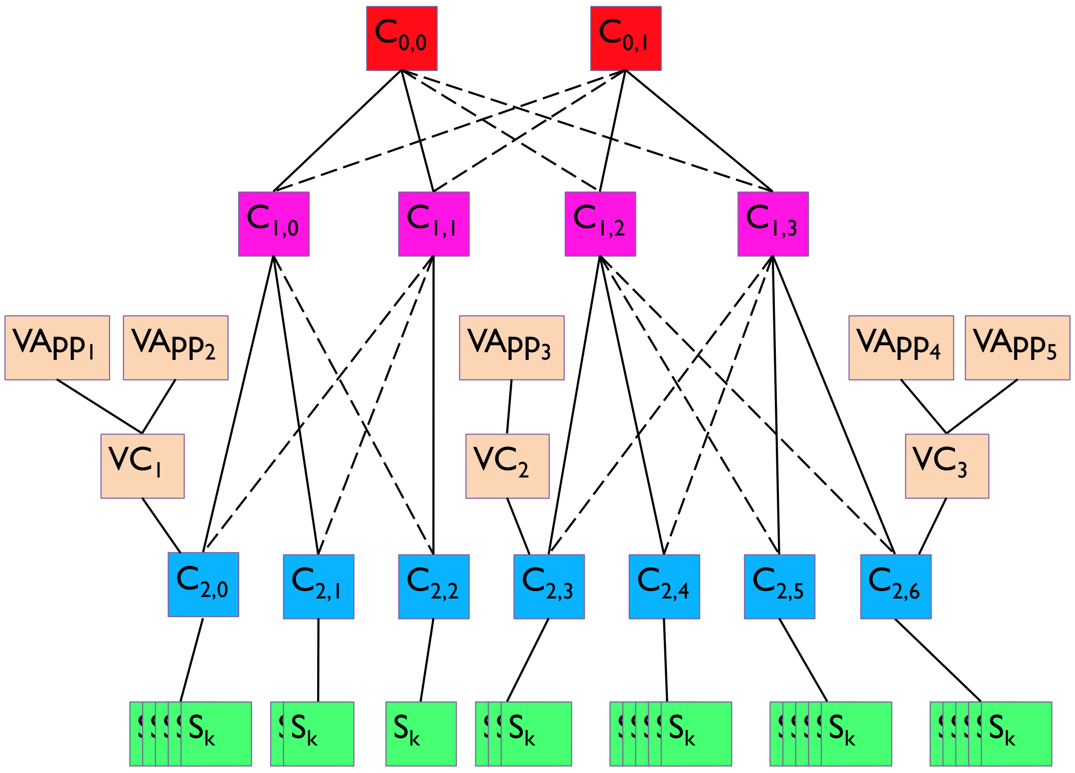

Figure 4.1: An SDN setup with hierarchical controllers and switches. Solid lines denote connections, and dotted lines backup connections. Virtual controllers and virtual apps are denoted by *VC* and *VApp*.

that the compromise of a level 2 controller does little damage outside of its immediate neighbourhood, then we have reduced the number of points of serious failure by another order of magnitude.

Although Figure 4.1 illustrates an SDN hierarchy informed by data centre practices, without much imagination it is plausible to map the components to those of an ISP (Network Operation Centre, Regional Offices, PoPs, etc.) and to the components of our airport example (where there are some central facilities, some in separate terminal buildings and some on different floors of those buildings, connected in a hierarchy).

The virtual controllers and virtual apps as discussed in Chapter 3 apply here as well. Cyber-physical systems often require tenants to share a common set of physical resources. The security and isolation properties from virtualising tenants complement those provided by hierarchy. We propose to host virtual controllers and virtual apps right down to the bottom layer of controllers, allowing rapid interaction with switches. This assumes stateless virtual apps or at least virtual apps that can share state outside of the SDN architecture, for example using shared servers. While stateful virtual apps are possible if the infrastructure provides memory as a service, the potential implications to performance, placement, and security warrant some thought.

The physical controllers will also host (physical) apps, but for infrastructure tasks. Apart from native functions necessary for running cyber-physical systems, infrastructure apps also contain management functions such as setting up and tearing down tenants, managing the topology, and capability token distribution.

4.3.2 Architectural security

We use AXE from Chapter 3 for access control using capability tokens. AXE provides a clean and consistent way to enforce access control between apps and switches. Virtual apps need access control permissions to interact with the underlying network. In AXE, there are two ways to obtain such permission: AXE-unaware apps can issue OpenFlow commands as normal and requests are first captured and decided by the shim in the physical controller, or AXE-aware apps can first request a capability token and use it for future requests. For scalability we take only the capability portion of AXE into our hierarchical setup. Apps need first request their access tokens at initialisation time, then present the token to the controller to be granted the necessary permissions.

In keeping with our isolation principles, the capability token generated at initialisation time does not include all of the permissions the app has. Rather, it is the subset of permissions bounded by the traffic that each instance of the app can see at that particular location. For example, if an app can see traffic of all of a /24 network, but there are four bottom controllers each with a non-overlapping /26 subnet (the four /26 subnets compose the original /24), then four capability tokens are generated. Each token is for one instance of the app, and grants permission for access traffic in that particular /26. In essence, the capability tokens are both *enabling* access as well as *confining* access. This part provides mandatory access control for tenants (*tenant-MAC*).

We also allow apps to declare, at initialisation time, a set of permissions which they declare they will use. This could be a smaller subset than the full permissions that the mandatory access control would give them. This self-limiting feature can be helpful for apps that may themselves be vulnerable. For example, a DPI app that parses incoming packets can be targeted for exploits (just as Wireshark is often the first target in capture-the-flag hacking competitions). Therefore it is prudent to self-limit the scope of its permission, and tell the system to alarm if it uses more. The infrastructure then monitors the app for undesired behaviour, such as a command that it declared it would not issue. Should an event occur, the infrastructure can mark the app as compromised and restart it with a clean copy. This part gives discretionary access control to tenants (*tenant-DAC*).

Finally, the infrastructure itself imposes access control on all controllers, virtual or physical. This is the mandatory access control mechanism of the infrastructure (*infrastructure-*

MAC). This step provides isolation within the hierarchy and ensures that parts can fail without affecting others. For each controller, the root issues a capability token that is tailored to its location within the network, similar to MAC for tenants. Because the root is responsible, through the hierarchy, for directing traffic through the SDN, it knows what each controller and their neighbours should be able to send and receive. Through AXE shims present at each node, each controller uses capability tokens to receive and send OpenFlow commands. If a controller is misbehaving, it cannot do more than it is allowed by the shim. If the controller and shim are both compromised (e.g., through physical attack), it will still be fairly powerless because its immediate neighbours will not have the tokens to *accept* anything outside of the existing defined permissions. By requiring capability tokens at both ends for a connection to be set up, network compromises can be strongly localised.

In summary, the infrastructure uses a set of three capability tokens to ensure isolation and security. The result is the least privilege of the three. The order in which the tokens are checked should be infrastructure-MAC, tenant-MAC, and then tenant-DAC. The set of all three accepts authorised traffic and denies everything else.

We note that many of the lower level security constructs come from AXE, which is described in greater detail in Chapter 3. There are also two previous work in particular that are relevant in this section, as we scale up in deployment size. Header Space Analysis [94] is a framework to statically check for errors in a large network. Its novelty lies in its treatment of packet headers as a set of bits, as opposed to pre-specified protocol meanings. In this way it is able to generalise packet headers and perform algebraic operations on them. An improved version is NetPlumber [93], which can incrementally check for changes online as they are issued, allowing rules to be checked for errors and policy violations as they are issued.

4.4 Implementation and evaluation

We implemented the above design as a proof of concept. While AXE already supplies the necessary access control and capability functions (and we evaluated it above), there are two main additions here: generating tokens for infrastructure-MAC and tenant-DAC. We also apply these performance measurements back to Heathrow's network to see what effects they would bring to a network of such size.

4.4.1 Capability token generation

The whole network has a set of IP address prefixes that it uses, whether in public or private ranges. The most obvious way to generate capability tokens from more powerful ones is by dividing capabilities by prefixes, and ensuring isolation at the address level.

For this method, we replicate all other attributes, such as port numbers, into each subdivided prefix for which the original has permission. For example, a permission for `ip_addr=10.0.0.0/24, port=80` across two controllers would divide into two tokens of `ip_addr=10.0.0.0/25, port=80` and `ip_addr=10.0.1.0/25, port=80`.

Other fancy methods also exist, and an optimisation can be made in minimising the number of tokens while still achieving the required set of permissions. Because root controllers are context-aware and can be made specific to a network, we can implement a more specific instance of capability token generation algorithm than simply using IP prefixes. After generating smaller capability tokens from an original one, it is desirable to check if the original set and the resulting set are equivalent in terms of covering space.

For each controller, the IP addresses of the traffic it sees is composed of the traffic sent to it by its neighbours. Within an SDN, this set depends on the flow rules and can be calculated in advance, since the central controllers know and issue these flow rules in the first place. For controllers interacting with external routers, this can be agreed out-of-band or learned from recent history. For a given controller, once the capabilities of all of its neighbours are known, its local capability is simply the union of the set.

Incremental update can however be made more difficult by a cold boot. Power cuts, for example, force the whole network to restart from scratch. The temptation may be to temporarily disable all AXE shims and allow unlimited access, so that the network can achieve a consistent state quickly, then calculate the capabilities for each controller one-by-one. We can also temporarily sign capabilities covering the whole local network and revoke them later. This just invites attacks involving disruptive service denial, though. A secure reboot takes longer. We generate capabilities starting from edge controllers, where external incoming IP addresses are known, and then propagate inwards. Once we have a working infrastructure, we start to create the tenant virtual networks according to network policy; tenant capabilities can now be generated and propagated across the network. The slow part may be in getting the infrastructure into a consistent state.

4.4.2 Tenant-DAC

The tenant-DAC is implemented in AXE, with an additional step that a virtual app can present its own policy file for AXE to monitor, co-existing with any MAC permissions that AXE already imposes. Because tenant-MAC already filters all disallowed connections from the switch to the virtual app, tenant-DAC only protects commands issued by the virtual app to the switch. At runtime, any violation to the DAC policy, for example a disallowed flow rule issued by the virtual app, will be denied but also trigger a restart of the app from a clean copy. The performance to watch out for here is how fast a virtual app can be restarted.

In AXE, we use Linux network namespaces to contain a virtual controllers and virtual

Setup	Mean	Std. dev.	Median
Real (wall clock)	122 ms	17 ms	125 ms
CPU (sys + user)	31 ms	5 ms	32 ms
Delete	Mean	Std. dev.	Median
Real (wall clock)	39 ms	42 ms	20 ms
CPU (sys + user)	6 ms	3 ms	4 ms
Setup + delete	Mean	Std. dev.	Median
Real (wall clock)	353 ms	8 ms	353 ms
CPU (sys + user)	30 ms	3 ms	28 ms

Table 4.1: Measured time for single Linux namespace setup, deletion, and cycle of setup and deletion. Measured over 100 times, units in milliseconds. The high standard deviation of Delete's wall clock time is due to a few outliers; we have not been able to find a root cause for them.

apps within their own networking scope. Starting a virtual app thus consists of setting up a new network namespace, starting a virtual controller, and starting the virtual app. While the time it takes to perform second and third steps vary depending on the implementation and on the complexity of the software, we can measure the first step for overhead it introduces.

Using a 32-bit Ubuntu 12.04 LTS with Linux kernel 3.2.0-23-generic-pae VM with 1GB of memory running inside a 2.4GHz MacBook Pro with OS X, we performed three sets of operations: setting up a namespace, deleting a namespace, and setting up then deleting immediately after. Each operation is repeated 100 times within each set to obtain mean, standard deviation, and median, and are measured by the `time` command each time. The resulting real and CPU (kernel space plus user space) times are taken, and presented in Table 4.1.

Even though the computer is not doing any other CPU-intensive tasks during the experiment, the 'real' time is much more than CPU time. It seems that Linux namespace is mostly non-CPU operations (waiting for virtual interfaces to be ready, for example). In the worst case we see a cycle of setup and delete taking over 300 milliseconds to complete. However, some optimisations can be performed. For example, a new instance of the virtual app can be instantiated first, and the old one shut down only after the new one is ready to receive commands. Security is not compromised because the shim is still blocking commands for which it has no capabilities.

4.4.3 Reality check

Given the above performance numbers, we can now apply them to realistic requirements to see the extent that security affects performance. In Section 4.1.2 we present the scale

Restart frequency	Bandwidth (Gb/s)	Bandwidth (percentage)
1 day	~50 Gb/s	100%
1 hour	~50 Gb/s	100%
1 minute	49 Gb/s	98%
1 second	21 Gb/s	41%

Table 4.2: Effective bandwidth degradation from 50 Gb/s with 4 of 8 services needing their virtual controllers restarted for each time period.

of current bandwidth supporting the technical operations of one terminal. In this section we estimate how this bandwidth would be affected under different scenarios with varying malicious packet presence.

The goal is to calculate how much Heathrow's bandwidth capacity is reduced with our security mechanisms deployed, assuming no modification to existing infrastructure hardware. In other words, we want to get a general idea of how much 353 milliseconds of mean wall clock time (per cycle of setting up and deleting virtual controllers) affect Heathrow operations in terms of bandwidth. Even though here we calculate the amount of reduction in bandwidth using the same infrastructure but with security added in, it is not difficult to calculate the extra hardware needed in order to maintain the same bandwidth capacity upon deployment of security mechanisms.

From Section 4.1.2 we can reasonably assume an aggregate peak rate of 50 Gb/s across 8 different services. We assume half of them to need their virtual controllers restarted, either proactively or reactively, during a varying range of time period from 1 day to 1 second. We assume that the restarting process is unoptimised and will stop forwarding traffic until it is back online. The resulting effective bandwidth is summarised in Table 4.2.

Alternatively we can also estimate based on packet-level volume. Packets-per-second, as opposed to bits-per-second, is often used for measuring hardware performance. The two measurements differ by a multiple of average packet size. Looking at Heathrow's 8 services, we estimate the first 4 services to use 1000-byte packets as they are more data-oriented, and the other 4 to use 100-byte packets as they are more time-sensitive. Weighted by the bandwidth of each service, the combined network has a bandwidth of 207 Megapackets per second (Mp/s). The effective bandwidth with security in place is summarised in Table 4.3. We note that the ratios here are the same as Table 4.2, this is because we assume equal likelihood of both packet-size profiles to be restarted. If some services are more likely to be attacked, the resulting effective bandwidth and percentages may be different.

Restart frequency	Bandwidth (Mp/s)	Bandwidth (percentage)
1 day	∼207 Mp/s	100%
1 hour	∼207 Mp/s	100%
1 minute	202 Mp/s	98%
1 second	86 Mp/s	41%

Table 4.3: Effective bandwidth degradation from 207 Mp/s with 4 of 8 services needing their virtual controllers restarted for each time period.

4.5 Conclusion

Large multi-tenanted networks, and cyber-physical systems in general, are likely to be more complex than traditional data centres and have larger attack surfaces. In order to be deployed in these environments, SDN needs an architecture that can support separation between mutually mistrustful tenants, and preferably support virtualisation too. In this chapter we propose a hierarchical architecture for SDN which uses the mandatory access controls we developed in AXE to accommodate mandatory access controls for both infrastructure and apps, and discretionary access control for apps as well.

We also implemented key components of this new architecture to illustrate how it can work and how much overhead it imposes. The biggest overhead we measured was from Linux namespace management; the rest of the work done by components such as virtual controllers, capabilities and apps is localised, so we do not foresee performance to be unduly affected by the size of the network.

As attacks and compromises become the norm rather than the exception, we need to design new systems that inherently isolate parts that do not need to influence each other. In our hierarchical design, not only do we isolate tenants by virtualisation, we also isolate the physical infrastructure so that parts of it can fail without affecting others. Capability tokens are used to enable allowed traffic and deny everything else, ensuring continued operation even if an immediate neighbour is compromised and malicious.

Chapter 5

Conclusions

Networking is not trivial. BGP misconfigurations are common and failures may be unreported, especially when they are contained within a network. Often, it is not the network management logic that is at fault, but rather how human operators use and abuse the software. Network management software needs to be more usable and less fragile, and we have argued here for mechanisms that enable operators to tie down certain low-level invariants, in order to support more robust high-level management.

This is intimately tied to network security management, and will become more so. Traditionally networks have one owner operator who has full access and control; proper configuration and management depend on operator experience and local knowledge. High-level intentions, from business contracts or network policies, have to be translated into low level commands; and although both the translation is done by automated scripts, their maintenance is a highly-skilled task, and human errors here are a significant cause of network outages. This traditional approach simply does not scale well, particularly as networks become more complex and virtualised.

The management of BGP is our initial example of this scalability problem, and was the focus of Chapter 2. Router management is still stuck in 1970s command line technology. As multiple protocols coexist, misconfigurations can easily have serious consequences. Current BGP management software simply does not provide adequate support to operators. We introduce access control into BGP management software as an error containment mechanism — as a way of making sure operators' strategic intentions are not as easily violated by tactical routing decisions. We present an implementation of a mandatory access control mechanism where role-based policy labels and enforces the direction of BGP announcements depending on link roles. Operators still have the final say, so these enforcement mechanisms can be overridden if need be — just not by accident.

We then move on to the new technology of software-defined networks, SDN. This saves money by moving from proprietary routers to open software running on commodity PC and switch hardware; it brings flexibility to operators; and it can even allow them to delegate some management authority to tenants within a virtual network. Tenants welcome this as it provides more options for traffic engineering.

However, current software does not adequately restrict what tenants can do. If tenants could run arbitrary third-party apps, they could interfere with other tenants' traffic, whether by accident or on purpose, and use resources without proper accounting. In Chapter 3, we investigate how to provide access controls for tenants and their apps within one physical SDN. We present AXE, an attribute-based access control mechanism for virtual controllers, which acts as a shim between each untrusted component and the trusted network. We discovered that by doing stateless access control within virtual controllers, we were essentially implementing stateless firewalls. This enables us to compare 50 years of research in access control with 25 years of research in firewalls. This gives us a number of insights, such as into the role of state in access control, and ways of managing it. We hope the insights will help SDN be deployed in more complex environments, bringing both cost savings and flexibility.

In Chapter 4, we look in more detail at cyber-physical systems to get some insight into a possible large-scale deployment scenario for SDN. One characteristic that is different is that we also need to consider physical compromises. With a traditional networking device, a physical attack can easily compromise the whole network. With SDN, we can do better. We propose a hierarchical architecture of SDN with AXE. A combination of mandatory access control in the host network, and mandatory and discretionary access control in each virtual network, can limit what each device and tenant can access, effectively limiting the damage that a compromised device can cause.

The goal of this dissertation has been to explore what access control can bring to network management. By integrating the two where they fit, network operators can draw on decades of experience elsewhere in access control (so there is some chance of doing it right) and provide the protection mechanisms needed for SDN to realise its potential in the complex multi-tenanted networks of the future. In this dissertation we have discussed the serious and pervasive threats that will have to be mitigated for that to happen.

As future work, one relevant direction is what the combination of BGP and SDN will look like, and what security issues will arise. We are starting to see BGP speakers that are directly coupled with SDN, so that routes can be processed and pushed to the forwarding plane directly. An instance of the BGP software can be bundled as an SDN app, and a future IXP or ISP may provide a central SDN platform upon which tenant BGP instances, along with their policies and customisations, are hosted. Within SDN, BGP software will receive more freedom in route and attribute handling, or even a completely different policy framework from what we have today. The added flexibility may expose security issues that are not possible within current rigid implementations. The interface between BGP and SDN also needs research: the security assumptions and realisations are different on each side, and vulnerabilities can arise because of their differences.

So we have shown we can use access control to block attacks, and to increase robustness. But can the access control mechanisms we propose actually be deployed in practice? This will come down to incentives; and our mechanisms give local incremental benefit — which may be just what is needed to drive adoption, further development and progress.

Bibliography

[1] Kanak Agarwal, Eric Rozner, Colin Dixon, and John Carter. SDN traceroute: tracing SDN forwarding without changing network behavior. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 145–150. ACM, 2014.

[2] William Aiello, John Ioannidis, and Patrick McDaniel. Origin authentication in interdomain routing. In *Proceedings of the 10th ACM conference on Computer and communications security*, CCS '03, pages 165–178, New York, NY, USA, 2003. ACM.

[3] Ali Al-Shabibi, Marc De Leenheer, Matteo Gerola, Ayaka Koshibe, Guru Parulkar, Elio Salvadori, and Bill Snow. OpenVirteX: make your virtual SDNs programmable. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 25–30. ACM, 2014.

[4] Ross J Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. Wiley Publishing, 2008.

[5] Vitaly Antonenko and Ruslan Smelyanskiy. Global network modelling based on Mininet approach. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 145–146. ACM, 2013.

[6] Katerina Argyraki, Petros Maniatis, David Cheriton, and Scott Shenker. Providing packet obituaries. In *ACM HotNets-III*, 2004.

[7] Josh Bailey, Dean Pemberton, Andy Linton, Cristel Pelsser, and Randy Bush. Enforcing RPKI-based routing policy on the data plane at an Internet exchange. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 211–212. ACM, 2014.

[8] Ilya Baldin, Shu Huang, and Rajesh Gopidi. A resource delegation framework for software defined networks. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 49–54. ACM, 2014.

[9] Hitesh Ballani, Paul Francis, and Xinyang Zhang. A study of prefix hijacking and interception in the Internet. *ACM SIGCOMM Computer Communication Review*, 37(4):265–276, 2007.

[10] Md Faizul Bari, Arup Raton Roy, Shihabur Rahman Chowdhury, Qi Zhang, Mohamed Faten Zhani, Reaz Ahmed, and Raouf Boutaba. Dynamic controller provisioning in software defined networks. In *CNSM*, pages 18–25, 2013.

[11] Sharon Barkai, Randy Katz, Dino Farinacci, and David Meyer. Software defined flow-mapping for scaling virtualized network functions. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 149–150. ACM, 2013.

[12] D Elliott Bell and Leonard J LaPadula. Secure computer systems: Mathematical foundations. Technical report, DTIC Document, 1973.

[13] Steven Bellovin. There be dragons. In *In UNIX Security Symposium III Proceedings*, 1992.

[14] Steven Bellovin, Randy Bush, and David Ward. Security requirements for BGP path validation. https://tools.ietf.org/html/draft-ietf-sidr-bgpsec-reqs-12, July 2014.

[15] Steven M. Bellovin. Security problems in the TCP/IP protocol suite. *Computer Communication Review*, 19(2):32–48, April 1989.

[16] Steven M Bellovin. Distributed firewalls. *Journal of Login*, 24(5):37–39, 1999.

[17] Steven M Bellovin and William R Cheswick. Network firewalls. *Communications Magazine, IEEE*, 32(9):50–57, 1994.

[18] Pankaj Berde, Matteo Gerola, Jonathan Hart, Yuta Higuchi, Masayoshi Kobayashi, Toshio Koide, Bob Lantz, Brian O'Connor, Pavlin Radoslavov, William Snow, and Guru Parulkar. ONOS: towards an open, distributed SDN OS. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 1–6. ACM, 2014.

[19] Kenneth J Biba. Integrity considerations for secure computer systems. Technical report, DTIC Document, 1977.

[20] Kyle Brogle, Danny Cooper, Sharon Goldberg, and Leonid Reyzin. Impacting IP prefix reachability via RPKI manipulations. Technical report, Boston University Computer Science, 2013.

[21] Randy Bush, Olaf Maennel, Matthew Roughan, and Steve Uhlig. Internet optometry: assessing the broken glasses in Internet reachability. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, IMC '09, pages 242–253, New York, NY, USA, 2009. ACM.

[22] Kevin Butler, Toni Farley, Patrick McDaniel, and Jennifer Rexford. A survey of BGP security issues and solutions. *Proceedings of the IEEE*, 98(1):100–122, January 2010.

[23] Matthew Caesar, Donald Caldwell, Nick Feamster, Jennifer Rexford, Aman Shaikh, and Jacobus van der Merwe. Design and implementation of a routing control platform. In *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, NSDI'05, pages 15–28, Berkeley, CA, USA, 2005. USENIX Association.

[24] Matthew Caesar and Jennifer Rexford. BGP routing policies in ISP networks. *Network, IEEE*, 19(6):5–11, 2005.

[25] Marco Canini, Dejan Kostic, Jennifer Rexford, and Daniele Venzano. Automating the testing of OpenFlow applications. In *Proceedings of the 1st International Workshop on Rigorous Protocol Engineering (WRiPE)*, number EPFL-CONF-167777, 2011.

[26] Marco Canini, Daniele Venzano, Peter Peresini, Dejan Kostic, and Jennifer Rexford. A NICE way to test OpenFlow applications. In *NSDI*, pages 127–140, 2012.

[27] Valdivieso Caraguay, Angel Leonardo, Lorena Isabel Barona Lopez, and Luis Javier Garcia Villalba. Evolution and challenges of software defined networking. In *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*, pages 1–7. IEEE, 2013.

[28] Carlos A Catania and Carlos García Garino. Automatic network intrusion detection: Current techniques and open issues. *Computers & Electrical Engineering*, 38(5):1062–1072, 2012.

[29] Haowen Chan, Debabrata Dash, Adrian Perrig, and Hui Zhang. Modeling adoptability of secure BGP protocols. *ACM SIGCOMM Computer Communication Review*, 36(4):279–290, 2006.

[30] Jian Chang, Krishna K Venkatasubramanian, Andrew G West, Sampath Kannan, Boon Thau Loo, Oleg Sokolsky, and Insup Lee. AS-TRUST: a trust quantification scheme for autonomous systems in BGP. In *Trust and Trustworthy Computing*, pages 262–276. Springer, 2011.

[31] Pei-chun Cheng, Jong Han Park, Keyur Patel, Shane Amante, and Lixia Zhang. Explaining BGP slow table transfers. In *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*, pages 657–666. IEEE, 2012.

[32] Bill Cheswick. The design of a secure internet gateway. In *USENIX Summer Conference Proceedings*. Citeseer, 1990.

[33] William R Cheswick, Steven M Bellovin, and Aviel D Rubin. *Firewalls and Internet security: repelling the wily hacker*. Addison-Wesley Longman Publishing Co., Inc., 2003.

[34] Margaret Chiosi, Don Clarke, Peter Willis, Andy Reid, James Feger, Michael Bugenhagen, Waqar Khan, Michael Fargano, C Cui, H Deng, et al. Network functions virtualisation: An introduction, benefits, enablers, challenges and call for action. In *SDN and OpenFlow World Congress*, pages 22–24, 2012.

[35] Luca Cittadini, Wolfgang Muhlbauer, Steve Uhlig, Randy Bush, Pierre Francois, and Olaf Maennel. Evolution of Internet address space deaggregation: myths and reality. *Selected Areas in Communications, IEEE Journal on*, 28(8):1238–1249, 2010.

[36] David D. Clark. Policy routing in Internet protocols. RFC 1102, May 1989.

[37] Richard Clayton. Internet multi-homing problems: Explanations from economics. In *Economics of Information Security and Privacy*, pages 67–78. Springer, 2010.

[38] Richard Clayton, Steven J Murdoch, and Robert NM Watson. Ignoring the great firewall of china. In *Privacy Enhancing Technologies*, pages 20–35. Springer, 2006.

[39] Rami Cohen, Katherine Barabash, Benny Rochwerger, Liran Schour, Daniel Crisan, Robert Birke, Cyriel Minkenberg, Mitchell Gusat, Renato Recio, and Vinit Jain. An intent-based approach for network virtualization. In *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, pages 42–50. IEEE, 2013.

[40] Lorenzo Colitti, Steinar Gunderson, Erik Kline, and Tiziana Refice. Evaluating IPv6 adoption in the internet. In Arvind Krishnamurthy and Bernhard Plattner, editors, *Passive and Active Measurement*, volume 6032 of *Lecture Notes in Computer Science*, pages 141–150. Springer Berlin / Heidelberg, 2010.

[41] Danny Cooper, Ethan Heilman, Kyle Brogle, Leonid Reyzin, and Sharon Goldberg. On the risk of misbehaving RPKI authorities. In *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, page 16. ACM, 2013.

[42] Jon Crowcroft, Malcolm Scott, Paul Brennan, Jaafar Elmirghani, Richard Penty, and Ian White. The intelligent networked airport (TINA). 2007.

[43] Brian Dickson. Route leaks — definitions. draft-dickson-sidr-route-leak-def-03, October 2012.

[44] Jerome Durand, Ivan Pepelnjak, and Gert Doering. BGP operations and security. BCP 194 RFC 7454, February 2015.

[45] ENISA. Inter-X: Resilience of the Internet interconnection ecosystem. Technical report, ENISA, April 2011.

[46] J. Vollbrecht et al. AAA authorization framework. RFC 2904 (Informational), August 2000.

[47] Seyed Kaveh Fayazbakhsh, Vyas Sekar, Minlan Yu, and Jeffrey C Mogul. FlowTags: enforcing network-wide policies in the presence of dynamic middlebox actions. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 19–24. ACM, 2013.

[48] Nick Feamster, Jaeyeon Jung, and Hari Balakrishnan. An empirical study of “bogon” route advertisements. *SIGCOMM Comput. Commun. Rev.*, 35:63–70, January 2005.

[49] Nick Feamster, Jennifer Rexford, Scott Shenker, Russ Clark, Ron Hutchins, Dave Levin, and Josh Bailey. SDX: A software-defined Internet exchange. *Open Networking Summit*, 2013.

[50] Nick Feamster, Jennifer Rexford, and Ellen Zegura. The road to SDN: An intellectual history of programmable networks. *Queue*, 11(12):20, 2013.

[51] Nick Feamster, Jared Winick, and Jennifer Rexford. A model of BGP routing for network engineering. In *Proceedings of the joint international conference on Measurement and modeling of computer systems*, SIGMETRICS ’04/Performance ’04, pages 331–342, New York, NY, USA, 2004. ACM.

[52] Anja Feldmann, Olaf Maennel, Z Morley Mao, Arthur Berger, and Bruce Maggs. Locating Internet routing instabilities. *ACM SIGCOMM Computer Communication Review*, 34(4):205–218, 2004.

[53] Andrew D Ferguson, Arjun Guha, Chen Liang, Rodrigo Fonseca, and Shriram Krishnamurthi. Hierarchical policies for software defined networks. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 37–42. ACM, 2012.

[54] Marcial P Fernandez. Comparing OpenFlow controller paradigms scalability: Reactive and proactive. In *Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on*, pages 1009–1016. IEEE, 2013.

[55] Nate Foster, Arjun Guha, Mark Reitblatt, Alec Story, Michael J Freedman, Naga Praveen Katta, Christopher Monsanto, Joshua Reich, Jennifer Rexford, Cole Schlesinger, Alec Story, and David Walker. Languages for software-defined networks. *Communications Magazine, IEEE*, 51(2):128–134, 2013.

[56] Alex Galis, Stuart Clayman, Lefteris Mamatas, Javier Rubio Loyola, Antonio Manzalini, Slawomir Kuklinski, Joan Serrat, and Theodore Zahariadis. Softwarization of future networks and services-programmable enabled networks as next generation software defined networks. In *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*, pages 1–7. IEEE, 2013.

[57] Murat Can Ganiz, Sudhan Kanitkar, Mooi Choo Chuah, and William M Pottenger. Detection of interdomain routing anomalies based on higher-order path analysis. In *Data Mining, 2006. ICDM'06. Sixth International Conference on*, pages 874–879. IEEE, 2006.

[58] Lixin Gao and Jennifer Rexford. Stable internet routing without global coordination. In *ACM SIGMETRICS Performance Evaluation Review*, volume 28, pages 307–317. ACM, 2000.

[59] Joseph Gersch and Dan Massey. Characterizing vulnerability to IP hijack attempts. In *Technologies for Homeland Security (HST), 2013 IEEE International Conference on*, pages 328–333. IEEE, 2013.

[60] Phillipa Gill, Michael Schapira, and Sharon Goldberg. Let the market drive deployment: A strategy for transitioning to BGP security. In *ACM SIGCOMM Computer Communication Review*, volume 41, pages 14–25. ACM, 2011.

[61] Phillipa Gill, Michael Schapira, and Sharon Goldberg. A survey of interdomain routing policies. *ACM SIGCOMM Computer Communication Review*, 44(1):28–34, 2013.

[62] Sharon Goldberg, Michael Schapira, Peter Hummon, and Jennifer Rexford. How secure are secure interdomain routing protocols. In *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, pages 87–98, New York, NY, USA, 2010. ACM.

[63] Geoffrey Goodell, William Aiello, Timothy Griffin, John Ioannidis, Patrick McDaniel, and Aviel Rubin. Working around BGP: An incremental approach to improving security and accuracy of interdomain routing. In *Proceedings of Network and Distributed System Security Symposium (NDSS)*, February 2003.

[64] Albert Greenberg, James R Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: a scalable and flexible data center network. In *ACM SIGCOMM computer communication review*, volume 39, pages 51–62. ACM, 2009.

[65] Arpit Gupta, Robert MacDavid, Rudiger Birkner, Marco Canini, Nick Feamster, Jennifer Rexford, and Laurent Vanbever. An industrial-scale software defined internet exchange point. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, Santa Clara, CA, March 2016. USENIX Association.

[66] Arpit Gupta, Laurent Vanbever, Muhammad Shahbaz, Sean P. Donovan, Brandon Schlinker, Nick Feamster, Jennifer Rexford, Scott Shenker, Russ Clark, and Ethan Katz-Bassett. SDX: A software defined Internet exchange. In *Proceedings of the*

2014 ACM Conference on SIGCOMM, SIGCOMM '14, pages 551–562, New York, NY, USA, 2014. ACM.

[67] Stephen Gutz, Alec Story, Cole Schlesinger, and Nate Foster. Splendid isolation: a slice abstraction for software-defined networks. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 79–84. ACM, 2012.

[68] Andreas Haeberlen, Ioannis C Avramopoulos, Jennifer Rexford, and Peter Druschel. NetReview: Detecting when interdomain routing goes wrong. In *NSDI*, pages 437–452, 2009.

[69] Chris Hall, Ross Anderson, Richard Clayton, Evangelos Ouzounis, and Panagiotis Trimintzios. Resilience of the Internet interconnection ecosystem. In *WEIS '11: Proceedings of the 10th Workshop on the Economics of Information Security*, 2011.

[70] Chris Hall, Dongting Yu, Zhi-li Zhang, Jonathan Stout, Andrew Odlyzko, Andrew W. Moore, Jean Camp, Kevin Benton, and Ross Anderson. Collaborating with the enemy on network management. In *Security Protocols XXII*. 2014.

[71] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, Bob Lantz, and Nick McKeown. Reproducible network experiments using container-based emulation. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pages 253–264. ACM, 2012.

[72] Scott Hazelhurst, Adi Attar, and Raymond Sinnappan. Algorithms for improving the dependability of firewall and filter rule lists. In *Dependable Systems and Networks, 2000. DSN 2000. Proceedings International Conference on*, pages 576–585. IEEE, 2000.

[73] Ethan Heilman, Danny Cooper, Leonid Reyzin, and Sharon Goldberg. From the consent of the routed: improving the transparency of the RPKI. *Proceedings of the ACM SIGCOMM 2014*, 2014.

[74] Brandon Heller, Rob Sherwood, and Nick McKeown. The controller placement problem. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 7–12. ACM, 2012.

[75] Timothy Hinrichs, Natasha Gude, Martìn Casado, John Mitchell, and Scott Shenker. Expressing and enforcing flow-based network security policies. Technical report, University of Chicago Technical Report, 2008.

[76] Rahul Hiran, Niklas Carlsson, and Phillipa Gill. Characterizing large-scale routing anomalies: A case study of the China Telecom incident. In *Passive and Active Measurement*, pages 229–238. Springer, 2013.

[77] Joseph Douglas Horton, RH Cooper, WF Hyslop, Bradford G. Nickerson, OK Ward, Robert Harland, Elton Ashby, and WM Stewart. The cascade vulnerability problem. *Journal of Computer Security*, 2(4):279–290, 1993.

[78] Hilary H Hosmer. Metapolicies I. *ACM SIGSAC Review*, 10(2-3):18–43, 1992.

[79] Hilary H Hosmer. Metapolicies II. In *Proceedings of the 15th National Computer Security Conference*, pages 369–378, 1992.

[80] Hongxin Hu, Wonkyu Han, Gail-Joon Ahn, and Ziming Zhao. FLOWGUARD: building robust firewalls for software-defined networks. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 97–102. ACM, 2014.

[81] Vincent C Hu, David Ferraiolo, Rick Kuhn, Arthur R Friedman, Alan J Lang, Margaret M Cogdell, Adam Schnitzer, Kenneth Sandlin, Robert Miller, Karen Scarfone, et al. Guide to attribute based access control (ABAC) definition and considerations (draft). *NIST Special Publication*, 800:162, 2013.

[82] Yiyi Huang, Nick Feamster, Anukool Lakhina, and Jim Jun Xu. Diagnosing network disruptions with network-wide analysis. In *ACM SIGMETRICS Performance Evaluation Review*, volume 35, pages 61–72. ACM, 2007.

[83] Geoff Huston, Mattia Rossi, and Grenville Armitage. Securing BGP — a literature survey. *Communications Surveys & Tutorials, IEEE*, 13(2):199–222, 2011.

[84] Kenneth Ingham and Stephanie Forrest. A history and survey of network firewalls. *University of New Mexico, Tech. Rep*, 2002.

[85] Trusted Network Interpretation. NCSC-TG 005. *National Computer Security Center*, 1990.

[86] Sotiris Ioannidis, Angelos D Keromytis, Steve M Bellovin, and Jonathan M Smith. Implementing a distributed firewall. In *Proceedings of the 7th ACM conference on Computer and communications security*, pages 190–199. ACM, 2000.

[87] Junaid Israr, Mouhcine Guennoun, and Hussein T. Mouftah. Credible BGP – extensions to BGP for secure networking. In *ICSNC '09: Proceedings of the 2009 Fourth International Conference on Systems and Networks Communications*, pages 212–216, Washington, DC, 2009. IEEE.

[88] Junaid Israr, Mouhcine Guennoun, and Hussein T. Mouftah. Mitigating IP spoofing by validating BGP routes updates. *IJCSNS International Journal of Computer Science and Network Security*, 9(5):71–76, May 2009.

[89] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jon Zolla, Urs Hölzle, Stephen Stuart, and Amin Vahdat. B4: Experience with a globally-deployed software defined WAN. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 3–14, New York, NY, USA, 2013. ACM.

[90] Nanxi Kang, Zhenming Liu, Jennifer Rexford, and David Walker. Optimizing the one big switch abstraction in software-defined networks. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, pages 13–24. ACM, 2013.

[91] Josh Karlin, Stephanie Forrest, and Jennifer Rexford. Pretty Good BGP: Improving BGP by cautiously adopting routes. In *Network Protocols, 2006. ICNP'06. Proceedings of the 2006 14th IEEE International Conference on*, pages 290–299. IEEE, 2006.

[92] Josh Karlin, Stephanie Forrest, and Jennifer Rexford. Autonomous security for autonomous systems. *Computer Networks*, 52(15):2908–2923, 2008.

[93] Peyman Kazemian, Michael Chang, Hongyi Zeng, George Varghese, Nick McKeown, and Scott Whyte. Real time network policy checking using header space analysis. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 99–111, 2013.

[94] Peyman Kazemian, George Varghese, and Nick McKeown. Header space analysis: Static checking for networks. In *NSDI*, pages 113–126, 2012.

[95] S. Kent, C. Lynn, J. Mikkelson, and K. Seo. Secure border gateway protocol (S-BGP) — real world performance and deployment issues. In *Proceedings of the Network and Distributed System Security Symposium*, 2000.

[96] S. Kent, C. Lynn, and K. Seo. Secure border gateway protocol (S-BGP). *IEEE Journal of Selected Areas in Communications*, 18(4):582–592, April 2000.

[97] Stephen Kent and Andrew Chi. Threat model for BGP path security. RFC 7132 (Informational), February 2014.

[98] Akmal Khan, Hyun-chul Kim, Taekyoung Kwon, and Yanghee Choi. A comparative study on IP prefixes and their origin ASes in BGP and the IRR. *ACM SIGCOMM Computer Communication Review*, 43(3):16–24, 2013.

[99] Varun Khare, Qing Ju, and Beichuan Zhang. Concurrent prefix hijacks: Occurrence and impacts. In *Proceedings of the 2012 ACM conference on Internet measurement conference*, pages 29–36. ACM, 2012.

[100] Ahmed Khurshid, Wenxuan Zhou, Matthew Caesar, and P Brighten Godfrey. Veriflow: Verifying network-wide invariants in real time. *ACM SIGCOMM Computer Communication Review*, 42(4):467–472, September 2012.

[101] Teemu Koponen, Martìn Casado, Natasha Gude, Jeremy Stribling, Leon Poutievski, Min Zhu, Rajiv Ramanathan, Yuichiro Iwata, Hiroaki Inoue, Takayuki Hama, et al. Onix: A distributed control platform for large-scale production networks. In *OSDI*, volume 10, pages 1–6, 2010.

[102] Vasileios Kotronis, Xenofontas Dimitropoulos, and Bernhard Ager. Outsourcing the routing control logic: better Internet routing based on SDN principles. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, pages 55–60. ACM, 2012.

[103] Diego Kreutz, Fernando Ramos, and Paulo Verissimo. Towards secure and dependable software-defined networks. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 55–60. ACM, 2013.

[104] Craig Labovitz, Scott Iekel-Johnson, Danny McPherson, Jon Oberheide, and Farnam Jahanian. Internet inter-domain traffic. In *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, pages 75–86, New York, NY, USA, 2010. ACM.

[105] Mohit Lad, Dan Massey, Dan Pei, Yiguo Wu, Beichuan Zhang, and Lixia Zhang. PHAS: a prefix hijack alert system. In *Proceedings of the 15th conference on USENIX Security Symposium-Volume 15*, page 11. USENIX Association, 2006.

[106] Butler W Lampson. Dynamic protection structures. In *Proceedings of the November 18-20, 1969, fall joint computer conference*, pages 27–38. ACM, 1969.

[107] Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, page 19. ACM, 2010.

[108] Matt Lepinski and Stephen Kent. An infrastructure to support secure Internet routing. RFC 6480 (Informational), February 2012.

[109] Matt Lepinski and Sean Turner. An overview of BGPSEC, July 2014.

[110] M. Lepinski (Ed.). BGPSEC protocol specification, July 2014.

[111] Dan Levin, Marco Canini, Stefan Schmid, and Anja Feldmann. Panopticon: Reaping the benefits of partial SDN deployment in enterprise networks. *TU Berlin/T-Labs, Tech. Rep*, 2013.

[112] Dan Levin, Andreas Wundsam, Brandon Heller, Nikhil Handigol, and Anja Feldmann. Logically centralized?: state distribution trade-offs in software defined networks. In

Proceedings of the first workshop on Hot topics in software defined networks, pages 1–6. ACM, 2012.

[113] Jun Li, Michael Guidero, Zhen Wu, Eric Purpus, and Toby Ehrenkranz. BGP routing dynamics revisited. *ACM SIGCOMM Computer Communication Review*, 37(2):5–16, 2007.

[114] Hung-Jen Liao, Chun-Hung Richard Lin, Ying-Chih Lin, and Kuang-Yuan Tung. Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, 36(1):16–24, 2013.

[115] Thomas A. Limoncelli. OpenFlow: A radical new idea in networking. *Queue*, 10(6):40:40–40:46, June 2012.

[116] Yaoqing Liu, Syed Obaid Amin, and Lan Wang. Efficient FIB caching using minimal non-overlapping prefixes. *ACM SIGCOMM Computer Communication Review*, 43(1):14–21, 2013.

[117] Yi Lu, Mei Wang, Balaji Prabhakar, and Flavio Bonomi. ElephantTrap: A low cost device for identifying large flows. In *High-Performance Interconnects, 2007. HOTI 2007. 15th Annual IEEE Symposium on*, pages 99–108. IEEE, 2007.

[118] Andra Lutu, Marcelo Bagnulo, and Olaf Maennel. The BGP visibility scanner. In *Computer Communications Workshops (INFOCOM WKSHPS), 2013 IEEE Conference on*, pages 115–120. IEEE, 2013.

[119] Robert Lychev, Sharon Goldberg, and Michael Schapira. BGP security in partial deployment: Is the juice worth the squeeze? In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 171–182, New York, NY, USA, 2013. ACM.

[120] Ratul Mahajan, David Wetherall, and Tom Anderson. Understanding BGP misconfiguration. In *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '02, pages 3–16, New York, NY, USA, 2002. ACM.

[121] Stephanos Matsumoto, Samuel Hitz, and Adrian Perrig. Fleet: defending SDNs from malicious administrators. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 103–108. ACM, 2014.

[122] Alain Mayer, Avishai Wool, and Elisha Ziskind. Fang: A firewall analysis engine. In *Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on*, pages 177–187. IEEE, 2000.

[123] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, March 2008.

[124] Marouen Mechtri, Ines Houidi, Wajdi Louati, and Djamal Zeghlache. SDN for inter cloud networking. In *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*, pages 1–7. IEEE, 2013.

[125] J. Medved, A. Tkacik, R. Varga, and K. Gray. Opendaylight: Towards a model-driven sdn controller architecture. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on a*, pages 1–6, June 2014.

[126] Jeffrey C Mogul. Simple and flexible datagram access controls for unix-based gateways. In *In USENIX Conference Proceedings*. Citeseer, 1989.

[127] Alec Muffett. Proper care and feeding of firewalls. In *In Proceedings of the UKERNA Computer Security Workshop. United Kingdom Education and Research Networking Association, Atlas.* Citeseer, 1994.

[128] S. Murphy. BGP security vulnerabilities analysis. RFC 4272 (Informational), 2006.

[129] Shishir Nagaraja, Virajith Jalaparti, Matthew Caesar, and Nikita Borisov. P3CA: Private anomaly detection across ISP networks. In *Privacy Enhancing Technologies*, pages 38–56. Springer, 2011.

[130] Ankur Kumar Nayak, Alex Reimers, Nick Feamster, and Russ Clark. Resonance: dynamic access control for enterprise networks. In *Proceedings of the 1st ACM workshop on Research on enterprise networking*, pages 11–18. ACM, 2009.

[131] Martin Nicholes and Biswanath Mukherjee. A survey of security techniques for the Border Gateway Protocol (BGP). *Communications Surveys Tutorials, IEEE*, 11(1):52 –65, 2009.

[132] Ola Nordström and Constantinos Dovrolis. Beware of BGP attacks. *SIGCOMM Comput. Commun. Rev.*, 34:1–8, April 2004.

[133] Bruno Astuto A Nunes, Marc Mendonca, Xuan-Nam Nguyen, Katia Obraczka, and Thierry Turletti. A survey of software-defined networking: Past, present, and future of programmable networks. *Communications Surveys Tutorials, IEEE*, 16(3):1617–1634, 2014.

[134] Tomohiko Ogishi, Yuichiro Hei, Shigehiro Ano, and Toru Hasegawa. Empirical study on inferring BGP routing instability and its location based on single point

observation. In *Communications, 2007. ICC'07. IEEE International Conference on*, pages 131–138. IEEE, 2007.

[135] Eric Osterweil, Terry Manderson, Russ White, and Danny McPherson. Sizing estimates for a fully deployed RPKI. Technical report, Technical report, Verisign Labs Technical Report, 2012.

[136] Vern Paxson. Bro: a system for detecting network intruders in real-time. *Computer networks*, 31(23):2435–2463, 1999.

[137] Debbie Perouli, Olaf Maennel, Iain Phillips, Sonia Fahmy, Randy Bush, and Rob Austein. An experimental framework for BGP security evaluation. *it-Information Technology Methoden und innovative Anwendungen der Informatik und Informationstechnik*, 55(4):147–154, 2013.

[138] Ben Pfaff, Justin Pettit, Keith Amidon, Martìn Casado, Teemu Koponen, and Scott Shenker. Extending networking into the virtualization layer. In *Hotnets*, 2009.

[139] Gerald J. Popek and Robert P. Goldberg. Formal requirements for virtualizable third generation architectures. *Commun. ACM*, 17(7):412–421, July 1974.

[140] Philip Porras, Seungwon Shin, Vinod Yegneswaran, Martin Fong, Mabry Tyson, and Guofei Gu. A security enforcement kernel for OpenFlow networks. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 121–126. ACM, 2012.

[141] Phillip Porras, Steven Cheung, Martin Fong, Keith Skinner, and Vinod Yegneswaran. Securing the Software-Defined Network Control Layer. In *Proceedings of the 2015 Network and Distributed System Security Symposium (NDSS)*, February 2015.

[142] Zafar Ayyub Qazi, Cheng-Chun Tu, Luis Chiang, Rui Miao, Vyas Sekar, and Minlan Yu. SIMPLE-fying middlebox policy enforcement using SDN. In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, pages 27–38. ACM, 2013.

[143] Mark Reitblatt, Nate Foster, Jennifer Rexford, Cole Schlesinger, and David Walker. Abstractions for network update. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '12, pages 323–334, New York, NY, USA, 2012. ACM.

[144] Martin Roesch. Snort: Lightweight intrusion detection for networks. In *LISA*, volume 99, pages 229–238, 1999.

[145] Christian Esteve Rothenberg, Marcelo Ribeiro Nascimento, Marcos Rogerio Salvador, Carlos Nilton Araujo Corrêa, Sidney Cunha de Lucena, and Robert Raszuk. Revisiting routing control platforms with the eyes and muscles of software-defined

networking. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 13–18. ACM, 2012.

[146] Charalampos Rotsos, Nadi Sarrar, Steve Uhlig, Rob Sherwood, and Andrew W. Moore. OFLOPS: An open framework for OpenFlow switch evaluation. In *Passive and Active Measurement*, pages 85–95. Springer, 2012.

[147] Matthew Roughan, Walter Willinger, Olaf Maennel, Debbie Perouli, and Randy Bush. 10 lessons from 10 years of measuring and modeling the Internet's autonomous systems. *Selected Areas in Communications, IEEE Journal on*, 29(9):1810–1821, 2011.

[148] Javier Salido, Masanori Nakahara, and Yinhai Wang. An analysis of network reachability using BGP data. In *Internet Applications. WIAPP 2003. Proceedings. The Third IEEE Workshop on*, pages 10–18. IEEE, 2003.

[149] Jerome H Saltzer and Michael D Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.

[150] Ravi Sandhu, David Ferraiolo, and Richard Kuhn. The NIST model for role-based access control: towards a unified standard. In *ACM workshop on Role-based access control*, volume 2000, 2000.

[151] Takayuki Sasaki, Yoichi Hatano, Kentaro Sonoda, Yoichiro Morita, Hideyuki Shimonishi, and Toshihiko Okamura. Load distribution of an openflow controller for role-based network access control. In *Network Operations and Management Symposium (APNOMS), 2013 15th Asia-Pacific*, pages 1–6. IEEE, 2013.

[152] Sandra Scott-Hayward, Christopher Kane, and Sakir Sezer. Operationcheckpoint: Sdn application control. In *2014 IEEE 22nd International Conference on Network Protocols*, pages 618–623. IEEE, 2014.

[153] Sandra Scott-Hayward, Gemma O'Callaghan, and Sakir Sezer. SDN security: A survey. In *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*, pages 1–7. IEEE, 2013.

[154] Sakir Sezer, Sandra Scott-Hayward, Pushpinder-Kaur Chouhan, Barbara Fraser, David Lake, Jim Finnegan, Niel Viljoen, Marc Miller, and Navneet Rao. Are we ready for SDN? implementation challenges for software-defined networks. *Communications Magazine, IEEE*, 51(7), 2013.

[155] Nick Shelly, Ethan J Jackson, Teemu Koponen, Nick McKeown, and Jarno Rajahalme. Flow caching for high entropy packet fields. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 151–156. ACM, 2014.

[156] Rob Sherwood, Michael Chan, Adam Covington, Glen Gibb, Mario Flajslik, Nikhil Handigol, Te-Yuan Huang, Peyman Kazemian, Masayoshi Kobayashi, Jad Naous, et al. Carving research slices out of your production networks with OpenFlow. *ACM SIGCOMM Computer Communication Review*, 40(1):129–130, 2010.

[157] Rob Sherwood, Glen Gibb, Kok-Kiong Yap, Guido Appenzeller, Martìn Casado, Nick McKeown, and Guru Parulkar. Flowvisor: A network virtualization layer. *OpenFlow Switch Consortium, Tech. Rep*, 2009.

[158] Xingang Shi, Yang Xiang, Zhiliang Wang, Xia Yin, and Jianping Wu. Detecting prefix hijackings in the Internet with Argus. In *Proceedings of the 2012 ACM conference on Internet measurement conference*, pages 15–28. ACM, 2012.

[159] Seungwon Shin and Guofei Gu. Attacking software-defined networks: a first feasibility study. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 165–166. ACM, 2013.

[160] Seungwon Shin, Phillip A Porras, Vinod Yegneswaran, Martin W Fong, Guofei Gu, and Mabry Tyson. FRESCO: Modular composable security services for software-defined networks. In *Proceedings of Network and Distributed System Security Symposium (NDSS)*, April 2013.

[161] Seungwon Shin, Yongjoo Song, Taekyung Lee, Sangho Lee, Jaewoong Chung, Phillip Porras, Vinod Yegneswaran, Jiseong Noh, and Brent Byunghoon Kang. Rosemary: A robust, secure, and high-performance network operating system. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 78–89. ACM, 2014.

[162] Sajad Shirali-Shahreza and Yashar Ganjali. FleXam: flexible sampling extension for monitoring and security applications in OpenFlow. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 167–168. ACM, 2013.

[163] Morris Sloman and Emil Lupu. Policy specification for programmable networks. In *Active Networks*, pages 73–84. Springer, 1999.

[164] Morris Sloman and Emil Lupu. Security and management policy specification. *Network, IEEE*, 16(2):10–19, 2002.

[165] Sooel Son, Seungwon Shin, Vinod Yegneswaran, Phillip Porras, and Guofei Gu. Model checking invariant security properties in OpenFlow. In *Communications (ICC), 2013 IEEE International Conference on*, pages 1974–1979. IEEE, 2013.

[166] Brent Stephens, Alan Cox, Wes Felter, Colin Dixon, and John Carter. PAST: Scalable ethernet for data centers. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pages 49–60. ACM, 2012.

[167] Jonathan Philip Stringer, Qiang Fu, Christopher Lorier, Richard Nelson, and Christian Esteve Rothenberg. Cardigan: Deploying a distributed routing fabric. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 169–170. ACM, 2013.

[168] David L. Tennenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall, and Gary J. Minden. A survey of active network research. *IEEE Communications Magazine*, 35:80–86, 1997.

[169] Amin Tootoonchian, Sergey Gorbunov, Yashar Ganjali, Martìn Casado, and Rob Sherwood. On controller performance in software-defined networks. In *USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE)*, volume 54, 2012.

[170] Vytautas Valancius, Cristian Lumezanu, Nick Feamster, Ramesh Johari, and Vijay V. Vazirani. How many tiers?: Pricing in the Internet transit market. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, pages 194–205, New York, NY, USA, 2011. ACM.

[171] Jacobus E Van der Merwe, Sean Rooney, Ian Leslie, and Simon Crosby. The tempest-a practical framework for network programmability. *Network, IEEE*, 12(3):20–28, 1998.

[172] Dennis Volpano, Xin Sun, and Geoffrey G Xie. Toward systematic detection and resolution of network control conflicts. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 67–72. ACM, 2014.

[173] Matthias Wählisch, Fabian Holler, Thomas C Schmidt, and Jochen H Schiller. Updates from the Internet backbone: An RPKI/RTR router implementation, measurements, and analysis. In *NDSS*, 2013.

[174] Feng Wang and Lixin Gao. On inferring and characterizing Internet routing policies. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pages 15–26. ACM, 2003.

[175] Feng Wang, Zhuoqing Morley Mao, Jia Wang, Lixin Gao, and Randy Bush. A measurement study on the impact of routing events on end-to-end internet path performance. In *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '06, pages 375–386, New York, NY, USA, 2006. ACM.

[176] Xitao Wen, Yan Chen, Chengchen Hu, Chao Shi, and Yi Wang. Towards a secure controller platform for openflow applications. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 171–172. ACM, 2013.

[177] Russ White. Securing BGP through secure origin BGP. Technical report, Cisco Internet Protocol Journal, September 2003.

[178] Alma Whitten and J. D. Tygar. Why Johnny can't encrypt: a usability evaluation of PGP 5.0. In *Proceedings of the 8th conference on USENIX Security Symposium - Volume 8*, SSYM'99, pages 14–14, Berkeley, CA, USA, 1999. USENIX Association.

[179] Jian Wu, Zhuoqing Morley Mao, Jennifer Rexford, and Jia Wang. Finding a needle in a haystack: Pinpointing significant BGP routing changes in an IP network. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 1–14. USENIX Association, 2005.

[180] Jian Wu, Ying Zhang, Z Morley Mao, and Kang G Shin. Internet routing resilience to failures: analysis and implications. In *Proceedings of the 2007 ACM CoNEXT conference*, page 25. ACM, 2007.

[181] Kuai Xu, Jaideep Chandrashekar, and Zhi-Li Zhang. A first step toward understanding inter-domain routing dynamics. In *Proceedings of the 2005 ACM SIGCOMM workshop on Mining network data*, pages 207–212. ACM, 2005.

[182] He Yan, Ricardo Oliveira, Kevin Burnett, Dave Matthews, Lixia Zhang, and Dan Massey. BGPmon: A real-time, scalable, extensible monitoring system. In *Conference For Homeland Security, 2009. CATCH'09. Cybersecurity Applications & Technology*, pages 212–223. IEEE, 2009.

[183] Curtis Yu, Cristian Lumezanu, Yueping Zhang, Vishal Singh, Guofei Jiang, and Harsha V Madhyastha. FlowSense: Monitoring network utilization with zero measurement cost. In *Proceedings of Passive and Active Measurement Conference (PAM)*, pages 31–41. Springer, 2013.

[184] Dongting Yu, Andrew W. Moore, Chris Hall, and Ross Anderson. Authentication for resilience: The case of SDN. In Bruce Christianson, James Malcolm, Frank Stajano, Jonathan Anderson, and Joseph Bonneau, editors, *Security Protocols XXI*, volume 8263 of *Lecture Notes in Computer Science*, pages 39–44. Springer Berlin Heidelberg, 2013.

[185] Harlan Yu, Jennifer Rexford, and Edward W Felten. A distributed reputation approach to cooperative Internet routing protection. In *Secure Network Protocols, 2005.(NPSec). 1st IEEE ICNP Workshop on*, pages 73–78. IEEE, 2005.

[186] Ye Yu, Chen Qian, and Xin Li. Distributed and collaborative traffic monitoring in software defined networks. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 85–90. ACM, 2014.

[187] Charles C Zhang, Marianne Winslett, and Carl A Gunter. On the safety and efficiency of firewall policy deployment. In *Security and Privacy, 2007. SP'07. IEEE Symposium on*, pages 33–50. IEEE, 2007.

[188] Ying Zhang and Makan Pourzandi. Studying impacts of prefix interception attack by exploring BGP AS-PATH prepending. In *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*, pages 667–677. IEEE, 2012.

[189] Jinjing Zhao and Yan Wen. Evaluation on the influence of Internet prefix hijacking events. *Computer Science and Information Systems/ComSIS*, 10(2):611–631, 2013.

[190] Xiaoliang Zhao, Dan Pei, Lan Wang, Dan Massey, Allison Mankin, S Felix Wu, and Lixia Zhang. An analysis of BGP multiple origin AS (MOAS) conflicts. In *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pages 31–35. ACM, 2001.

[191] Jiafeng Zhu, Weisheng Xie, Li Li, Min Luo, and Wu Chou. Software service defined network: Centralized network information service. In *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*, pages 1–7. IEEE, 2013.

[192] N Zilberman, P. M. Watts, C Rotsos, and A. W. Moore. Reconfigurable network systems and software-defined networking. *Proceedings of the IEEE*, 103:1102–1124, Jul 2015.