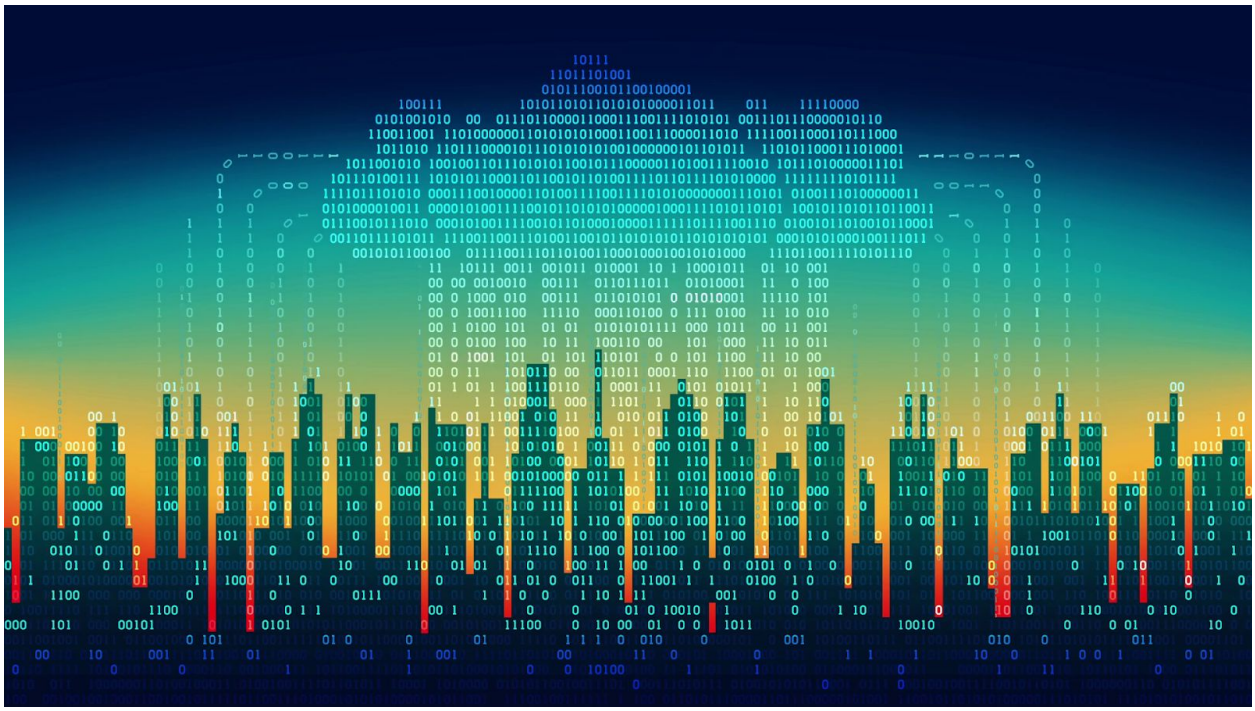
Network Security - Peter Garamvoelgyi 2018280070

# Paper Reading Assignment #2

## Abstractions for Network Update



[1]

## 1. Introduction

This report is a summary and discussion of the paper *Abstractions for Network Update* [1].

The report is organized as follows. Section 2 presents a brief historical overview of policy-based management techniques. Section 3 is a discussion of the main insights and contributions of [1]. In Section 4, I present my personal insights into policy-based network management gained while reading this paper. Finally, Section 5 concludes by giving an overview of the topics discussed.

[1] Image source: https://www.videoblocks.com

# 2. Policy-based management

*All information in this chapter is based on [2] and my personal insights*

It is not an easy task to summarize what people mean when they refer to the abstract concept of *policy*. One way to view it is that a policy is *a rule that defines a choice in the behavior of a system that is intended to reflect objectives of the system managers* [3].

Generally speaking, policy-based management (PBM) systems help people reason about the functionality of a complex system in a high-level, abstract way. PBM standards, languages, and tools can help us create systems that behave correctly by automatically implementing the tedious, error-prone low-level tasks.

Policy-based management approaches first appeared in the late 60s in the realm of information security. Ranging from simple access control matrices [4] to complex role- and domain-based control policies [5], these techniques formed the way we view security today.

Another important area of study that followed soon after was the application of PBM to networking policies. These techniques range from enforcing SLAs to intra- and inter-organizational control of routes and access to resources, and even network configuration updates, the main topic of this report.

Finally, later development in PBM mostly focused on how to formalize (specification languages), optimize (refinement), and check (consistency analysis) these policies.

# 3. Paper overview

*All information in this chapter is based on [1] and my personal insights*

The problem the paper *Abstractions for Network Update* by Reitblatt et al. is tackling is that of consistent network updates. Computer networks tend to have a large number of rules and policies distributed among numerous nodes (routers, firewalls, switches). Typical rules include security policies like dropping packets or refusing connections under certain conditions, complex routing instructions, and access control mechanisms.

Network update is the process of switching from one configuration to a new one. This might include adding and removing rules, rerouting packets, etc. Complex network updates typically consist of multiple steps that need to be executed on numerous routers throughout the network. Such updates, the authors argue, often introduce incorrect intermediate network states that might potentially lead to instability, including outages, performance disruptions, and security vulnerabilities.

## Consistent network updates

Reitblatt et al. introduce a system for consistent network updates. Their analysis includes two distinct consistency-levels:

1. **Per-packet consistency**: Each packet in the network is either processed using the old configuration, or the new configuration, but never a mixture of the two.
2. **Per-flow consistency**: Each packet belonging to the same flow is handled by the same configuration.

An example of the latter is the case of HTTP load balancers. A load balancer must route all TCP packets from the same connection to the exact same host. This can be guaranteed using per-flow consistent update mechanisms.

Network update processes, the authors argue, should be formalized in an abstract way, leaving figuring out the exact steps to the system.[2] The goal is to hide the complexities of network updates from the developers, making the whole process less tedious and more secure.

The authors propose a *2-phase update* mechanism that guarantees per-packet consistency. This mechanism consists of two steps:

1. **Install new configuration on all nodes**. This is a so-called *unobservable update*; nodes will not start using the new configuration yet, so this update has no effect on network traffic flow.

[2] "*We believe that any energy the programmer devotes to navigating this space would be better spent in other ways. The tedious job of finding a safe sequence of commands that implement an update should be factored out, optimized, and reused across many applications.*" [1]

2. **Unlock new policy on ingress ports**. This is a so-called *one-touch update*; ingress ports (basically the entry nodes of the network) start *stamping* incoming packets with a new version number. Other nodes know that packets with this version number should be handled using the new policy, while previous packets should use the old policy.

*2-phase updates* offer a simple yet powerful mechanism for consistent network updates. The performance impact of this scheme can be reduced using various tactics and heuristics, as discussed in section 5 of the paper.

*Per-flow consistent updates* can be implemented combining packet versioning and rule timeouts. Once flows associated to an old configuration finish, the associated rules timeout automatically, and new flows will be handled by the new configuration. The authors discuss two other approaches (*wildcard cloning* and *end-host feedback mechanisms*), but these are not yet available in OpenFlow.

The proposed update mechanisms are implemented in a framework called Kinetic built on top of OpenFlow. This system offers a high-level interface (two functions `per_packet_update` and `per_flow_update`), that implement consistent updates with automatic optimizations. An evaluation is also presented using the Mininet network simulation system and various commonly used network topologies (fat-tree, small-world, and random).

## A mathematical model of computer networks

The authors of [1] not only propose the network update mechanisms discussed above, but also prove the correctness of their statements. To do this, they formulate an abstract mathematical model of computer networks that allows them to capture configuration changes and packet traces in a mathematically rigorous manner.

Having defined the syntax and semantics of their model, the authors proceed to prove many of the properties the proposed update mechanisms have.

$$
\begin{array}{lll}
\text{Bit} & b & ::= 0 \mid 1 \\
\text{Packet} & pk & ::= [b_1, ..., b_k] \\
\text{Port} & p & ::= 1 \mid \cdots \mid k \mid Drop \mid World \\
\text{Located Pkt} & lp & ::= (p, pk) \\
\text{Trace} & t & ::= [lp_1, ..., lp_n] \\
\text{Update} & u & \in LocatedPkt \rightharpoonup LocatedPkt\ list \\
\text{Switch Func.} & S & \in LocatedPkt \to LocatedPkt\ list \\
\text{Topology Func.} & T & \in Port \to Port \\
\text{Port Queue} & Q & \in Port \to (Packet \times Trace)\ list \\
\text{Configuration} & C & ::= (S, T) \\
\text{Network State} & N & ::= (Q, C)
\end{array}
$$

An important discussion in [1] is that of trace properties. Trace properties characterize the paths that individual packets can take through the network. Using the proposed mathematical model, it can be proven that per-packet consistent updates preserve trace properties. As many common network properties can be expressed as trace properties (access control, connectivity, routing correctness, loop- freedom, etc.), these are also preserved.

Trace properties of networks can be checked and analyzed using several existing model checker and static analysis tools. The authors offer a library with several canonical rules that can be used to verify planned configuration changes or debug existing ones.

# 4. Insights and ideas

- **Abstraction is a powerful tool**

  Abstraction is one of the most powerful tools used by computer scientists and software engineers. By adopting an appropriate high-level view, we can disregard many distracting details of complex systems and focus on the essence. This in turn helps us build correct and performant systems.

  [1] gives a very interesting application of the principle of abstraction to the practical problem of consistent network updates. Seeing that a high-level model built on solid mathematical foundations can help us avoid many mistakes and make developers' lives easier was very inspiring for me.

- **Mathematical models help us build correct systems**

  The authors of this paper built their approach on a mathematical abstraction. Their model reminded me of many principles commonly associated with functional programming (immutability, list processing). This was refreshing to see after reading many publications with imperative algorithms and models.

  As with any abstraction, the challenge of defining a useful mathematical model is to decide which details to include and which ones to ignore. In general, we should strive for simplicity while preserving the key properties of the system. The model presented in [1] did a good job at this. As a positive side-effect, this model not only helps us prove facts about the system but also makes it easier to understand the problem of network updates and complex concepts like trace properties.

- **Good performance is essential for adoption**

  The adoption of any new technology relies on a cost-benefit consideration. If people feel that adding extra security to their system will have a significant performance impact, they are less likely to do it. The authors of this paper considered this and offered some ways to optimize their original method.

# 5. Conclusion

Useful, reusable abstractions help us build build correct systems and make better use of our time as managers, developers, and system administrators. The paper *Abstractions for network update* [1] offers a good application of these principles to the problem of consistent network updates. The mathematically rigorous, yet still clear approach employed by the authors is a good example to follow in the area of policy-based management and beyond.

# References

[1] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, “Abstractions for network update,” in Proceedings of ACM SIGCOMM, 2012.

[2] R. Boutaba, and I. Aib, “Policy-based management: A historical perspective,” Journal of Network and Systems Management, vol. 15, no. 4, pp. 447-480, 2007.

[3] Damianou, N., Dulay, N., Lupu, E.C., Sloman, M.: Ponder: A language for specifying security and management policies for distributed systems. Imperial College Research Report DoC 2000/1, 2000

[4] Lampson, B.W.: Dynamic protection structures. In: Proc. AFIPS Conf.35 (1969 FJCC), pp. 27–38 (1969)

[5] Ferraiolo, D., Kuhn, R.: Role-based access controls. In: 15th NIST-NCSC National Computer Security Conference, pp. 554–563 (1992)