# **AuthedIP: Authenticated IP Layer**

Daniel Chin. nq285. April 2022.

Project report for Networks and Mobile Systems with professor Anirudh Sivaraman.

## **Introduction**

Ethan was proposed by Casado et al. in 2007, and it provides a programmable control plane for an enterprise network. Ethane routers, when encountering new flows, ask a centralized Controller for routes to install. The Controller can thus implement policies for the enterprise network on a software level. Although Ethane offers significant controllability improvement for the enterprise network, its original proposition has several security limitations. Here I address two of them.

The first problem is with Ethane’s binding between (a) the user, (b) the MAC address, and (c) the IP address. When a user appears in the network, they first authenticate with the Controller with their credentials. This endorses their IP and MAC addresses. However, multiple users may share the same addresses, e.g. with multi-user OSes and multi-VM end hosts. Ethane is blind to multi-user address sharing. The biding between the user and the addresses is therefore weak. The fundamental cause is that IP has no per-packet *user* identity.

The second problem is with Ethane’s binding between (a) the packet, (b) its first Ethane router ingress port, and (c) its source addresses. Via MAC spoofing, a packet can easily lie about its source addresses, so it is impossible for Ethane to verify which end host sent the packet. The only secure information is which Ethan router ingress port the packet first went through. That is insufficient to hold packets accountable, unless the enterprise has deployed Ethane routers to all edges and each end host is connected directly to a different port on an Ethane router. Therefore, the binding between the packet and its source is weak. An attacker, as long as they share the same ingress port with a valid user, can steal the user’s MAC address and send packets in their name. From Ethane’s perspective, a stream of packets with the same source address arrive at the ingress port after the user authenticates successfully with the Controller. No anomaly is noticeable. A user’s authentication thus only proves “such a user is behind such an ingress port” and nothing else. The user endorses the ingress port, and not their packets. In other words, the core problem is that Ethane has no *per-packet* user verification.

Observe that the above two problems share the same root: the lack of per-packet user verification. In this project, I design and study Authenticated IP (“AuthedIP”) where every packet contains user endorsement. In an enterprise network using AuthedIP, non-users cannot send anything, and registered users cannot send anything anonymous. Anomalies can thus be traced back to registered users.

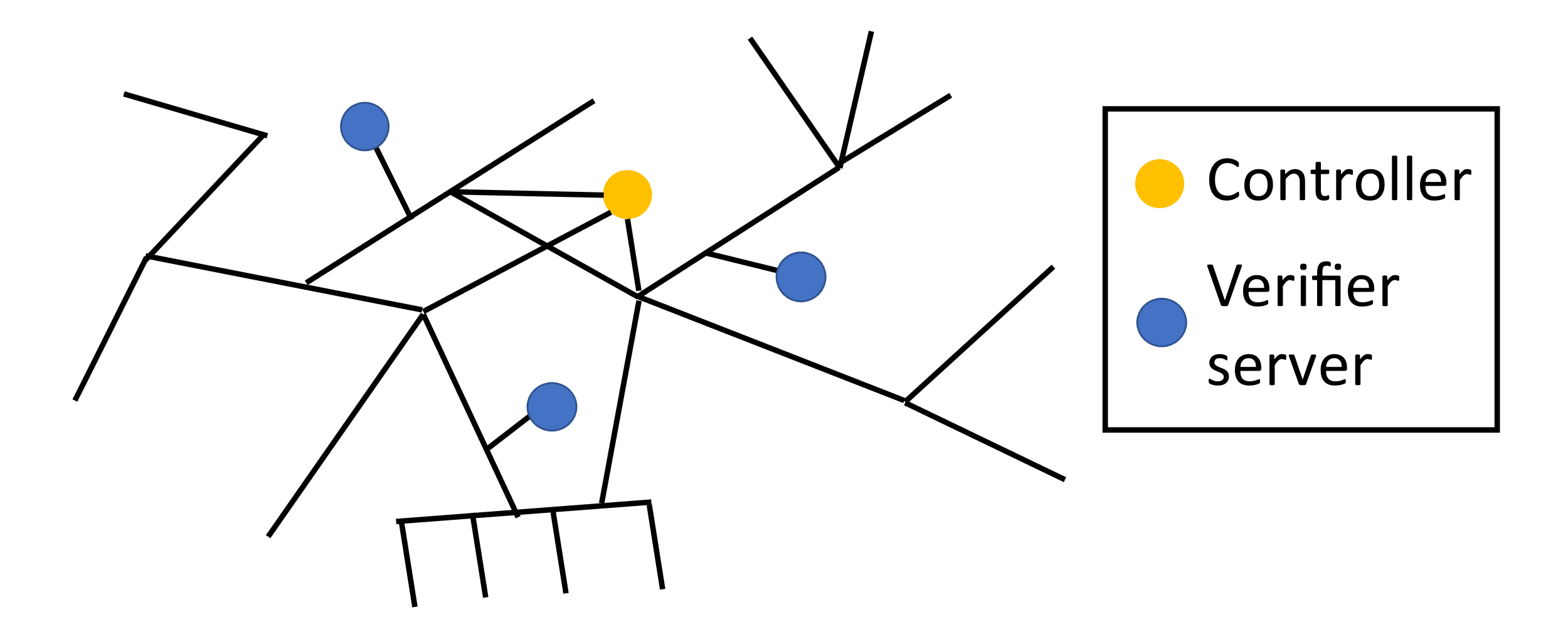
## **Threat Model**

Network-layer security is concerned with many types of threats, but in this project, I only focus on the Denial of Service (DoS) attack originating from within the network. DoS aims to deplete network resources and overwhelm the victim server via sending high-rate traffic to the victim. One can defend against DoS on the application layer, but that already surrenders CPU resources, and cannot solve link congestion near the victim. On the contrary, network-layer security aims to stop the DoS traffic near its source, therefore localizing the damage.

In the context of enterprise networks, DoS attacks can be categorized according to whether the attacker is a registered user. A non-user may connect to the enterprise network and perform a DoS. A corrupted end host may perform a DoS against its user’s will. A misaligned employee may willingly perform a DoS. These cases shall all be treated.

I assume a subset of the enterprise network is physically secure. Denote this subset by “the Inside”. On the Inside, no link is hijacked, no router is corrupted, and no port is exposed. Everything else is “the Outside”. Any data from the Outside may be fabricated by an attacker. The Inside must be a connected graph, i.e., it cannot have islands. The [Controller](#_Controller) and all [verifier servers](#_Verifier_Server) must be Inside.

## **AuthedIP Protocol**

  
**Figure 1. Example network.**

The overall design of AuthedIP involves:

* Each user’s RSA public key is registered with the Controller.
* Each IP packet is signed by a user.
* Routers forward packets, but duplicate some of them to the nearest verifier server.
* Verifier servers checks the packets’ signature. If verification fails, alert the responsible router.
* When sufficiently alerted, routers enter the *verify-then-forward mode* to reject unverified packets.

The following subsections describe the complete AuthedIP protocol.

### **User registration**

One employee may have multiple users, each with different access privileges. Each user has an RSA key pair, the private key and the public key. The public key is 512 bits. If I was to put a public key in every packet, that will bring a 64-byte overhead per packet, which is already worse than all fields and options in the IP header combined. Therefore, each public key is associated with its 64-bit “public key ID”. The public key ID is simply the 64-bit suffix of the hash of the public key. The public key ID goes into every packet, which brings an acceptable 8-byte overhead per packet. The Controller stores a hash table that maps public key IDs to public keys. New users register with the Controller by adding an entry to the hash table. A 64-bit public key ID allows different users in the network. To avoid ID collision, use rejection sampling.

### **Packet**

In addition to an IP packet, an AuthedIP packet has three extra fields:

* timestamp (16 bytes),
* public key ID (8 bytes),
* signature (64 bytes).

Because the IP options are limited to 40 bytes, I put the three extra fields at the beginning of the IP payload, similar to related works (Argyraki et al. 2005, Naous et al. 2011). The packet’s “real” content is thus offset by 88 bytes.

The timestamp is an integer in seconds since Epoch. The public key ID identifies the public key for verifying the packet. The signature is an RSA signature of the SHA-256 hash of the packet’s *identity*. The packet’s identity is the 24-byte concatenation of its timestamp, its source address, and its destination address.

This way, a packet with a correct signature shows the private key owner (i.e., the registered user)’s endorsement for the source address, the destination address, and the timestamp. This is enough to deter DoS, as will be shown in section “How to DoS AuthedIP”.

## **Router**

Not all routers in the network need to be AuthedIP routers. However, every border router (separating the Inside and the Outside) has to be an AuthedIP router. An AuthedIP router knows which of its egress links point to the Inside and which links point to the Outside. This is pre-configured per-router. Also pre-configure the Controller’s IP address into each AuthedIP router. An AuthedIP router maintains a suspicion score, , for each of its ingress port. is the only persistent state, and is initialized to be zero for every port.

When a packet wants to go from the Outside to the Inside, the router first looks at associated with the ingress port. If the ingress port is in the *forward-then-verify mode*. The router first forwards the packet normally, then randomly decides whether to check the packet. The probability of checking the packet is which starts at a baseline as If the router decides to check the packet, the router makes a duplicate of the packet and wraps it in a new IP packet. The wrapper packet is sent from the router to the nearest [verifier server](#_Verifier_Server), and the wrapper packet contains two extra fields:

* a 5-byte field that identifies the ingress port;
* a 1-byte flag, marked as , meaning the verifier server should not forward this packet.

If the ingress port is in the *verify-then-forward mode*. Conceptually, the packet is redirected to go through a verifier as a middlebox. Concretely, the router wraps and sends the packet to the nearest verifier server (instead of its original destination). The wrapper packet has the two fields described above, but the 1-byte flag is set to , indicating that the verifier server should forward the packet to its original destination if its signature passes verification.

When the router is alerted by a verifier server that an unverified packet from ingress port has been detected, is incremented by . Each port’s decays over time, at a rate of . With these two rules, a DoS will increase quickly, while verification failure due to transmission distortion will decay spontaneously.

## **Verifier Server**

Verifier servers are simply some special end hosts attached to the network. Their sole purpose is to verify packet signatures. A verifier keeps a copy of the hash table of public keys via talking with the Controller. For every packet it receives, the verifier checks three things:

* The timestamp is fresh. Specifically, less than 10 seconds old.
* The public key ID is in the hash table, and resolves to public key .
* The RSA signature of the packet can be verified with

If any check fails, the packet is considered unverified. Because the wrapper packet contains router and ingress port information, the verifier can alert the responsible router to raise for the specific ingress port. At the same time, record the event in logs. Bursts of verification failure should notify the network operator.

If all checks passed, the packet is considered verified. The 1-byte flag in the wrapper packet tells the verifier whether to drop the packet or forward it to its original destination.

## **Controller**

Blah

## **Optionally: Active Subscription**

…

## **After an Attack**

After an attack with unverified packets, we know which border router the attack originated from. Either investigate that subnet on the Outside, or install more AuthedIP routers with physical security to push the frontier of the Inside. That will pinpoint the attacker’s host in the future.

After an attack with verified packets, we know which registered user signed the packets. Either investigate that user, or check their machine for malware.

## **How to DoS AuthedIP**

This section reasons from the adversarial perspective and tries to DoS an AuthedIP network.

Being unverified… there is a window, but short… this case is investiaged with simulation…

The levels decay over time. An attacker can send unverified packets sporadically, waiting for to reach zero before sending another one. This attack disguises as random link distortions. Therefore, it must not send unverified packets at a significantly higher rate than random link distortions. That means, this attack’s damage is at most as bad as random link distortions, up to a constant multiplier.

Being verified… brute force signature? Replay packets?

## **Simulation Results**

<https://github.com/Daniel-Chin/AuthedIP>. No route discovery, no subscription.

## **Miscellaneous**

* You want robust connectivity on the Inside, and many islands on the Outside.

## **References**

Argyraki, K. J., & Cheriton, D. R. (2005, April). Active Internet Traffic Filtering: Real-Time Response to Denial-of-Service Attacks. In USENIX annual technical conference, general track (Vol. 38).

Casado, M., Freedman, M. J., Pettit, J., Luo, J., McKeown, N., & Shenker, S. (2007). Ethane: Taking control of the enterprise. ACM SIGCOMM computer communication review, 37(4), 1-12.

Naous, J., Walfish, M., Nicolosi, A., Mazieres, D., Miller, M., & Seehra, A. (2011, December). Verifying and enforcing network paths with ICING. In Proceedings of the Seventh Conference on Emerging Networking Experiments and Technologies (pp. 1-12).