Verifying and enforcing network paths with ICING

Jad Naous, Michael Walfish,
Antonio Nicolosi, David Mazières,
Michael Miller, and Arun Seehra

Today: New protocol for every feature

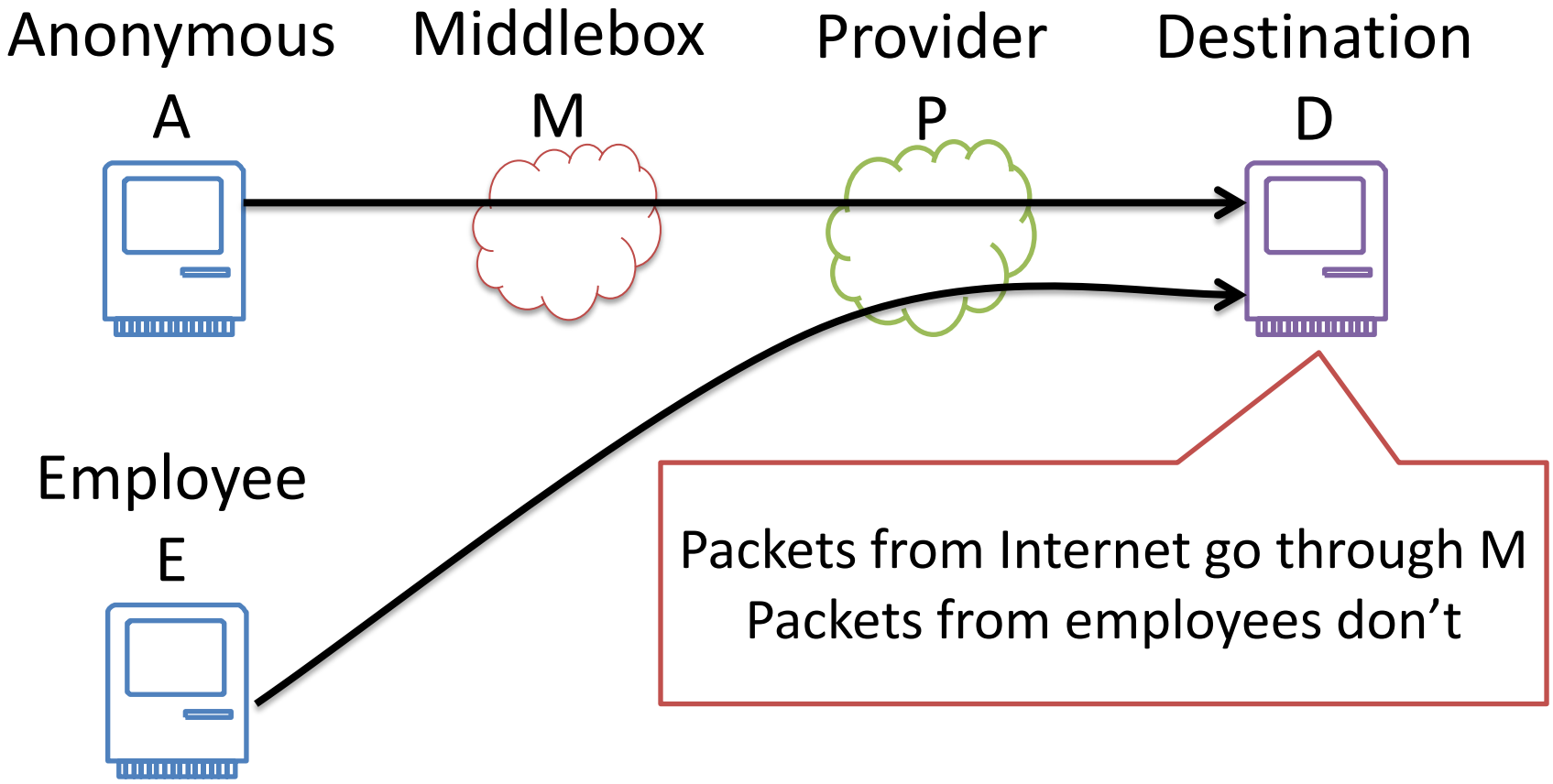
- Remote VPN, Private WANs, Specifying QoS, Firewalls, Filters, DoS protection, ACLs, Secure routing, …
- Tomorrow: security outsourcing, access delegation, better DoS protection, source routing?

Today: New protocol for every feature

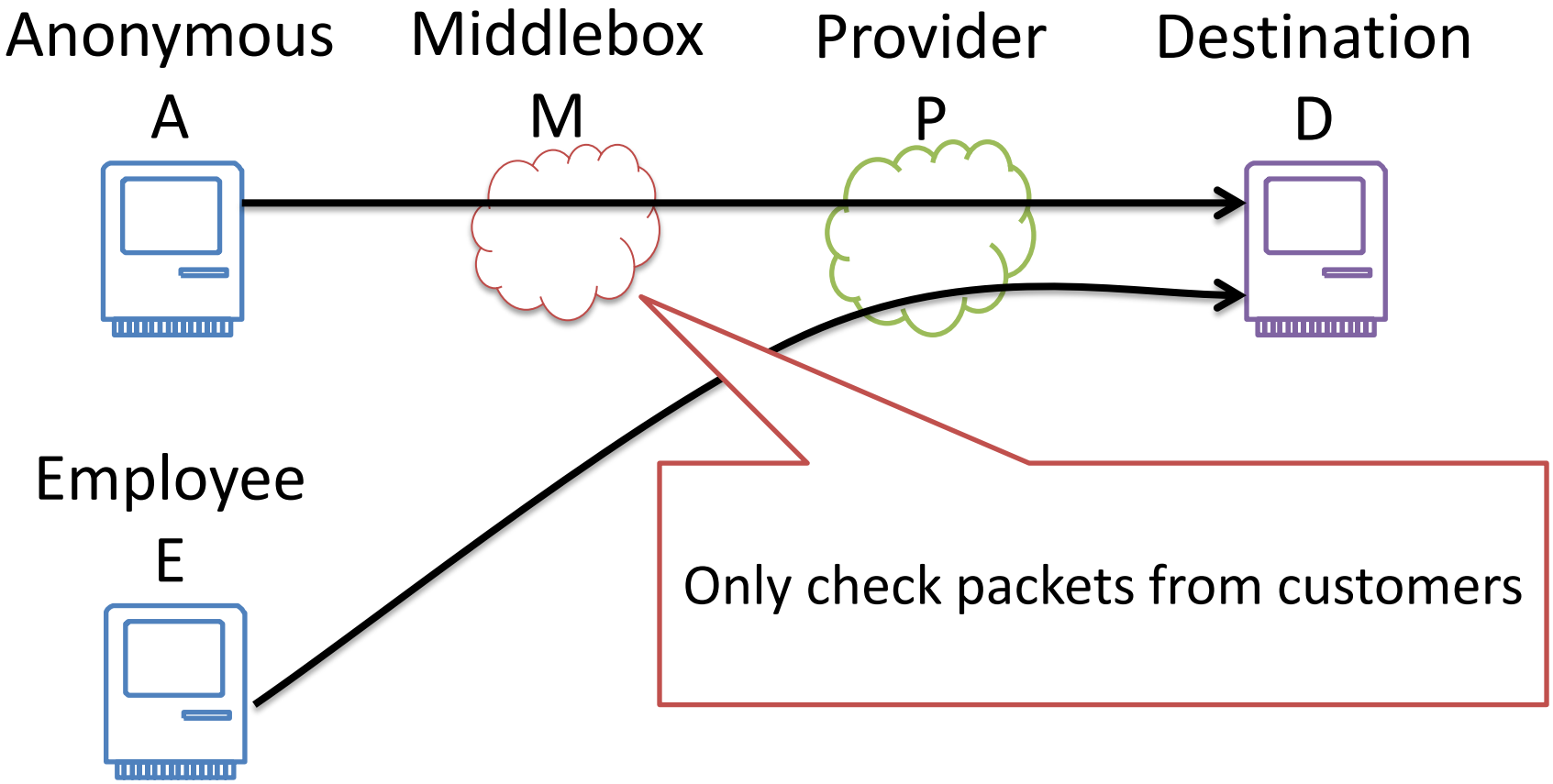
- Remote VPN, Private WANs, Specifying QoS, Firewalls, Filters, DoS protection, ACLs, Secure routing, …
- Tomorrow: security outsourcing, access delegation, better DoS protection, source routing?

Complexity, Incompatibility, Ossification

Example: enterprise outsourcing deep-packet inspection



Example: service provider verifies business relationship



One primitive to rule them all?

Path Consent:

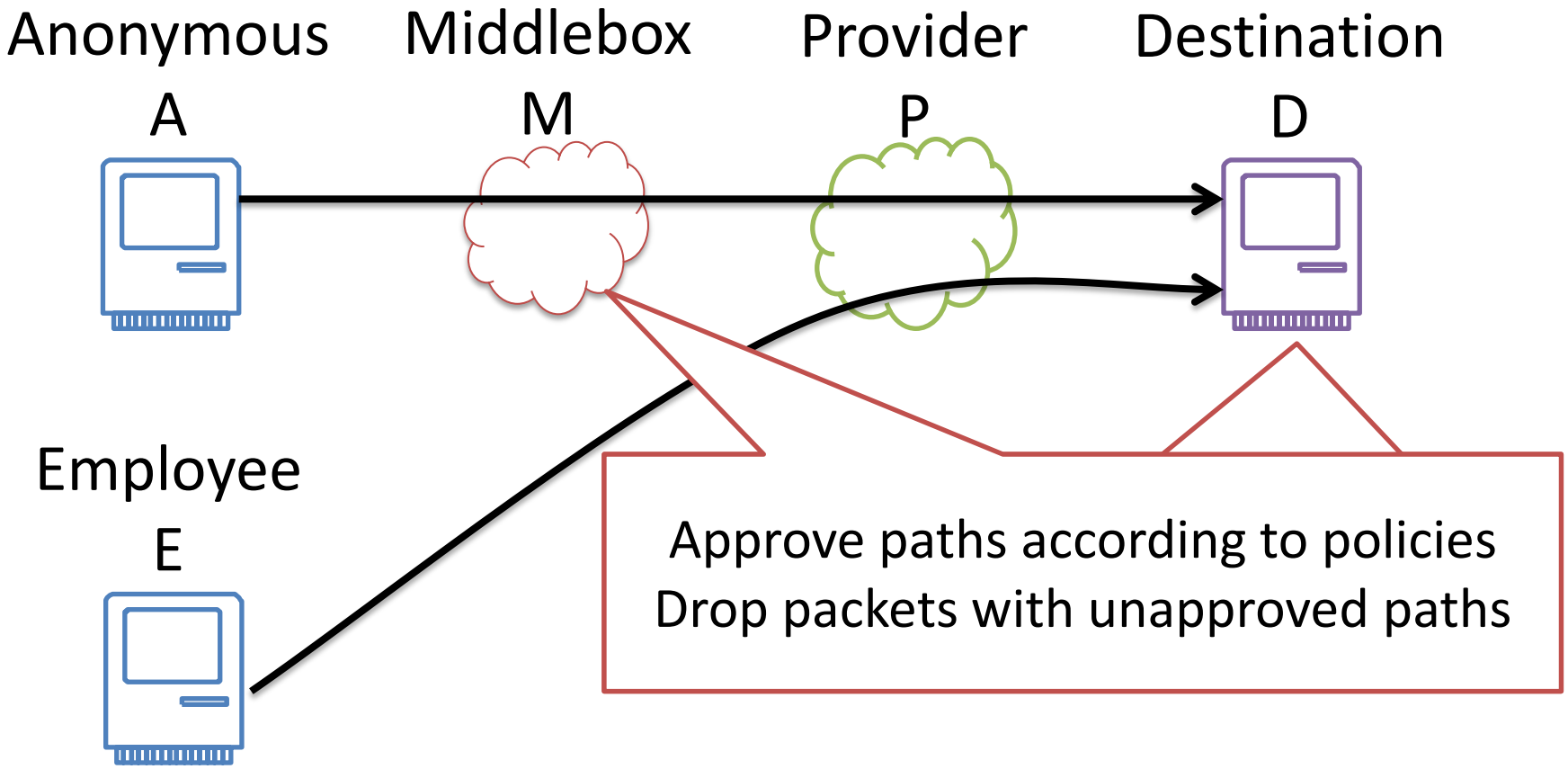
Every entity on the path (or a delegate) has to approve the whole path.

Path Verification:

Upon receiving a packet, every entity on the path can verify that the packet has followed an **approved** path

Difficult Challenge

Path Consent and Path Verification in action



Why are Path Consent and Path verification sufficient?

Other protocols give one entity more control over the other entities on the path.

What are the guarantees?

- **Granularity**: Domain level guarantees.
- **Role of honest nodes**: Honest nodes drop non-compliant packets.
- **No skipping**: Cannot skip an uncompromised honest hop, even with collusion.
- **No negative policies**: Cannot prove a packet did *not* pass through a certain entity.
- **Does not prove trustworthiness**: “Trusted” does not mean “Trustworthy”.

This talk will answer

- How can we provide Path Consent and Path verification?
- At what cost?

Outline

- Design in three iterations
- Prototype implementation and results
- Related work and conclusion

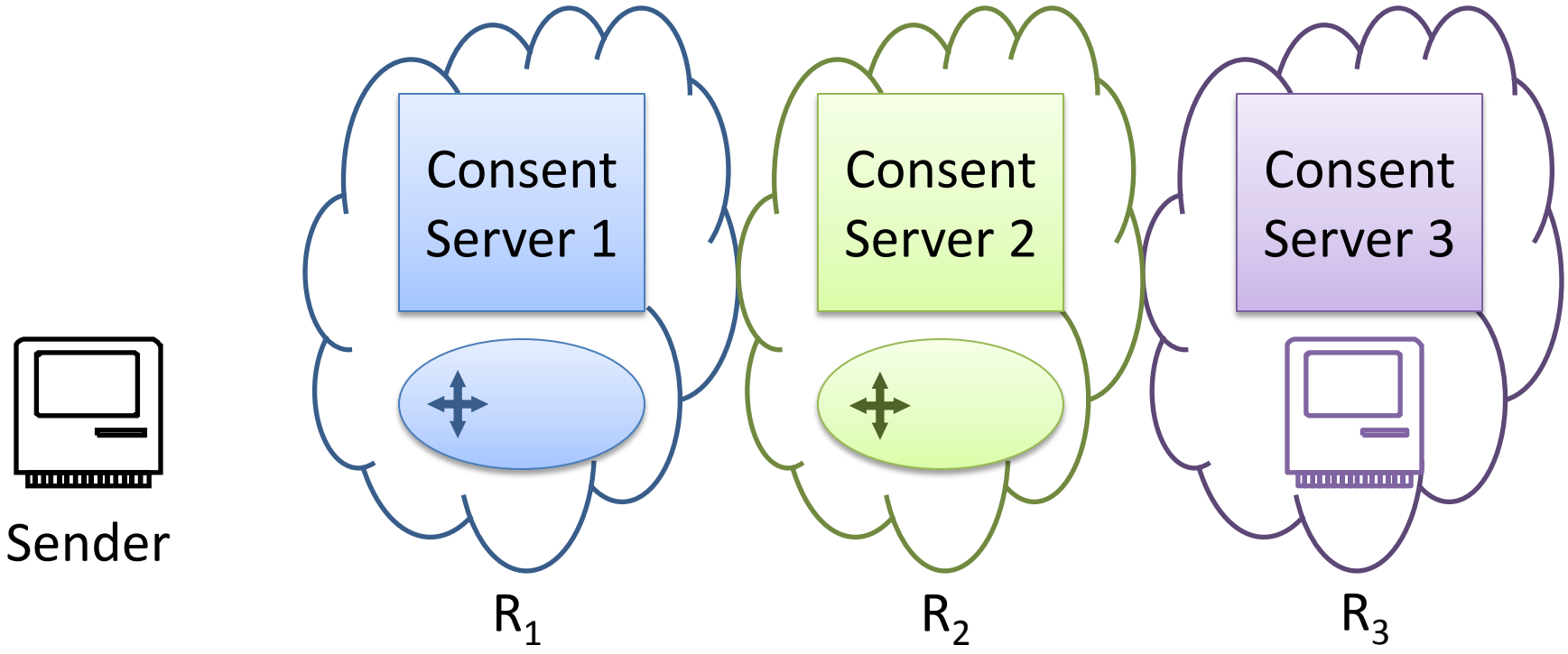
Operational constraints

Adversarial

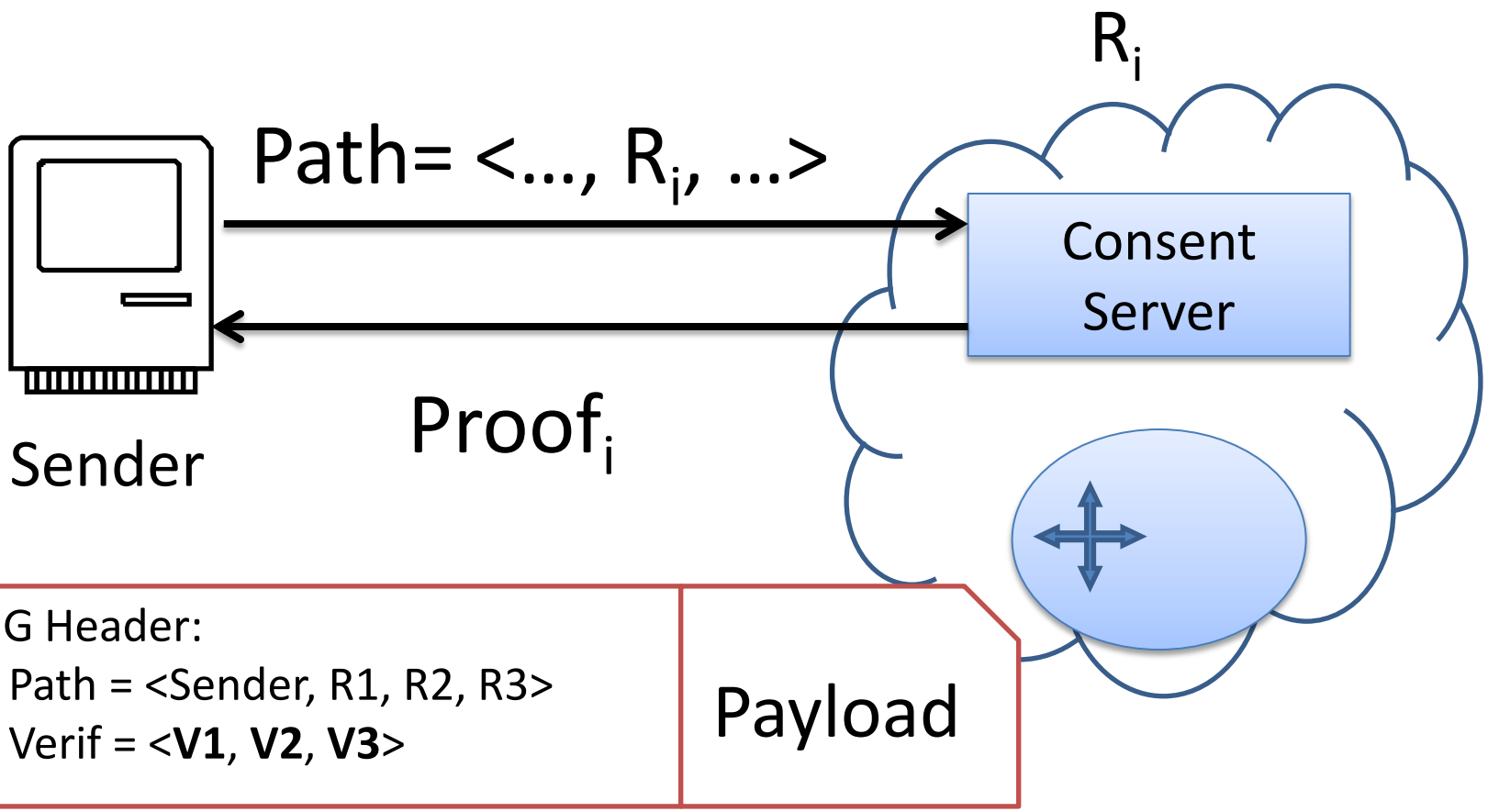
Decentralized

High performance

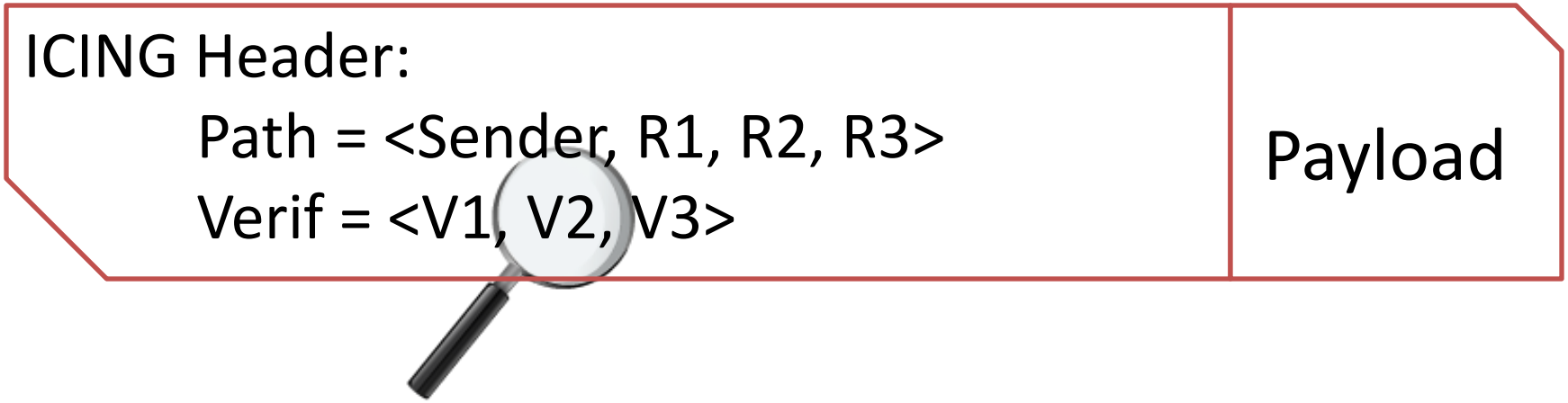
Architecture:
Control plane/Data plane split



Communication starts by contacting consent servers



Forwarder uses its verifier to implement Path Verification



Strawman 1: Public key crypto

Name entities by self-minted
public keys (PK/SK)

Use signatures for
Path Consent and Path Verification

Operational constraints

✓ Adversarial

✓ Decentralized

✗ High performance

Strawman 2: Symmetric Key Crypto

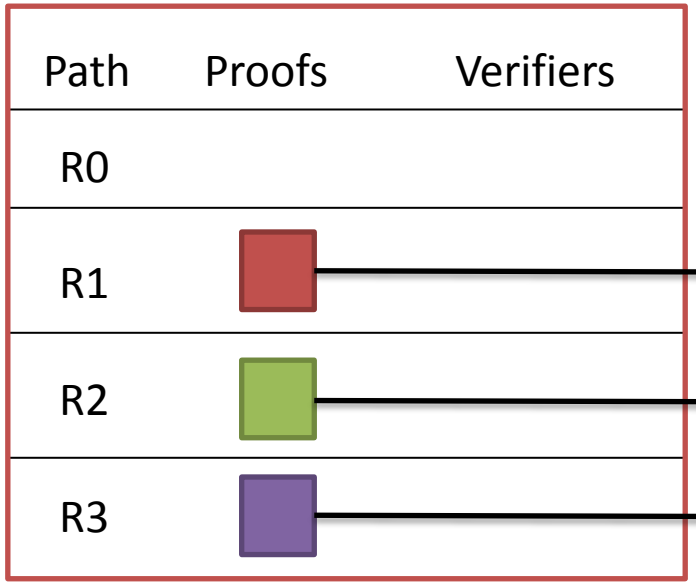
1 PK/SK ⟶ R pair-wise shared keys

1 Sig ⟶ n MACs

R = number of realms on Internet
n = number of realms on a path

O(R) symmetric keys for configuration
$O(n^2)$ overhead in the packet

Strawman 2: Sender inserts proofs of consent in the packet



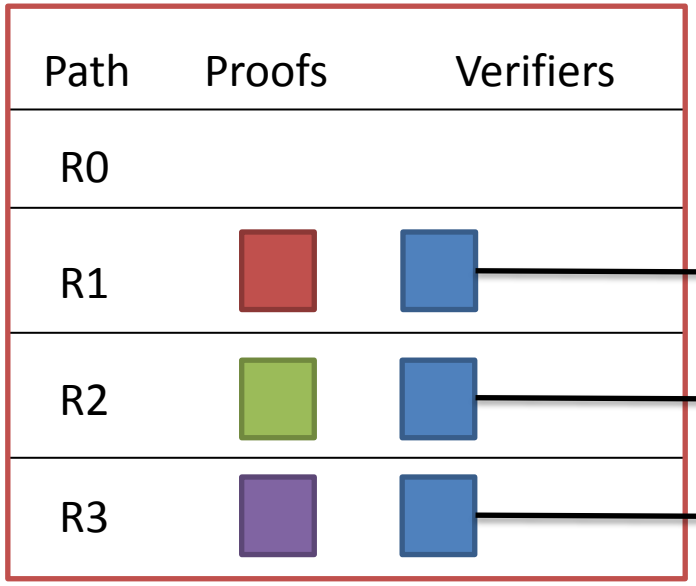
Created using symmetric shared keys between consent server forwarder.

Uses R1's consent key s_1

Uses R2's consent key s_2

Uses R3's consent key s_3

Strawman 2: Sender proves to later realms it has passed the packet using O(R) preconfigured keys

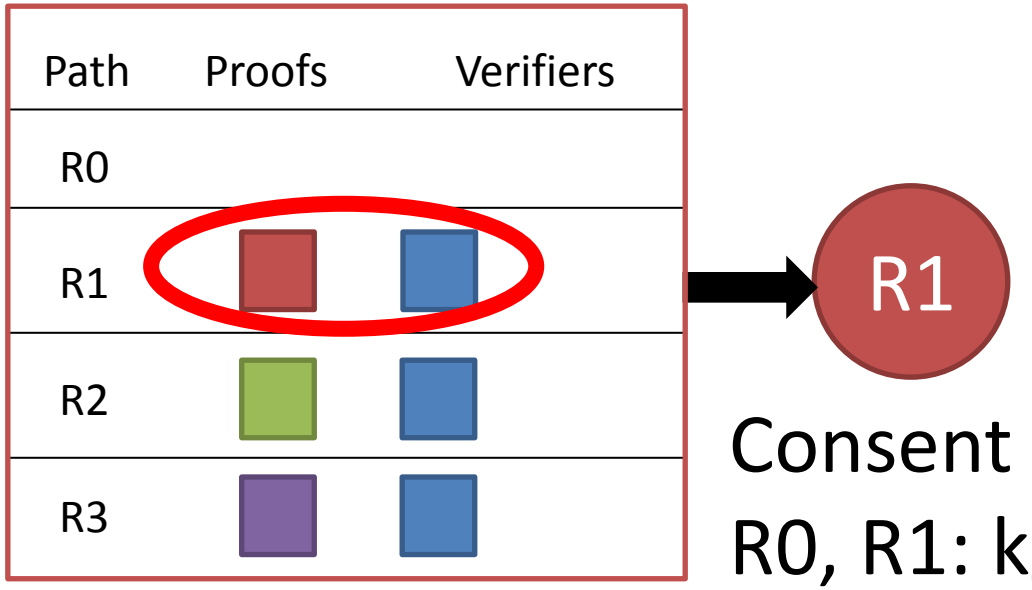


Uses k_{01} shared between R0, R1

Uses k_{02} shared between R0, R2

Uses k_{03} shared between R0, R3

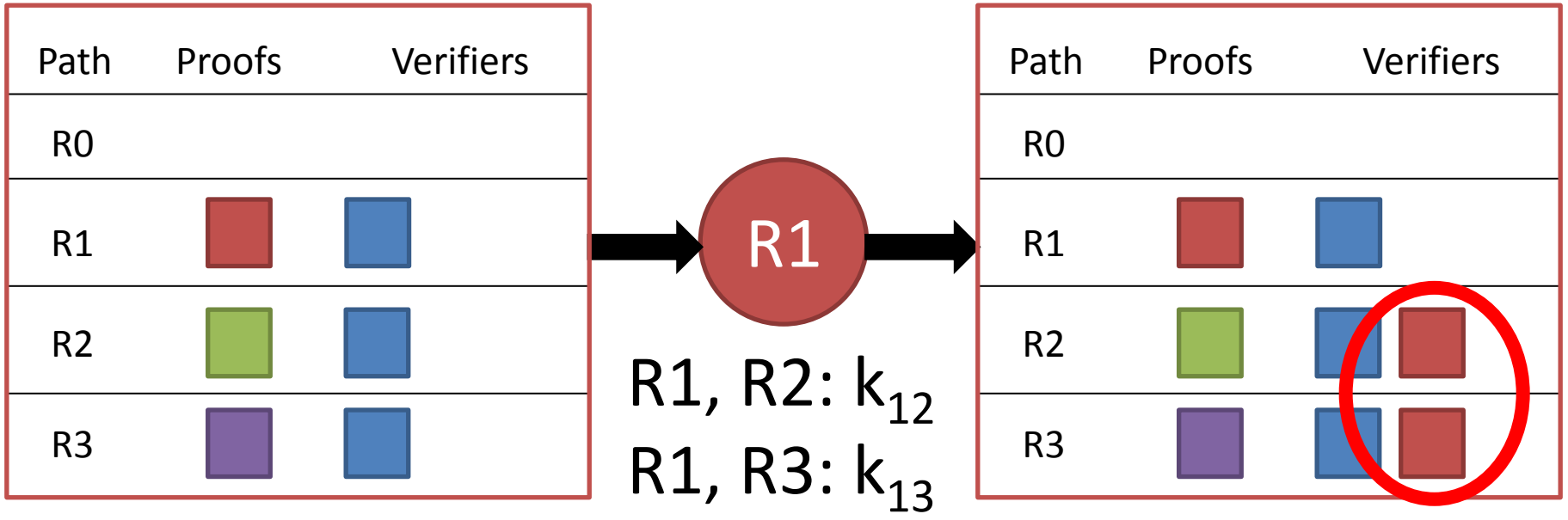
Strawman 2: Forwarders use O(R) symmetric keys for verification



Consent server, R1: s_1

R0, R1: k_{01}

Strawman 2:
Forwarder adds proof for later realms



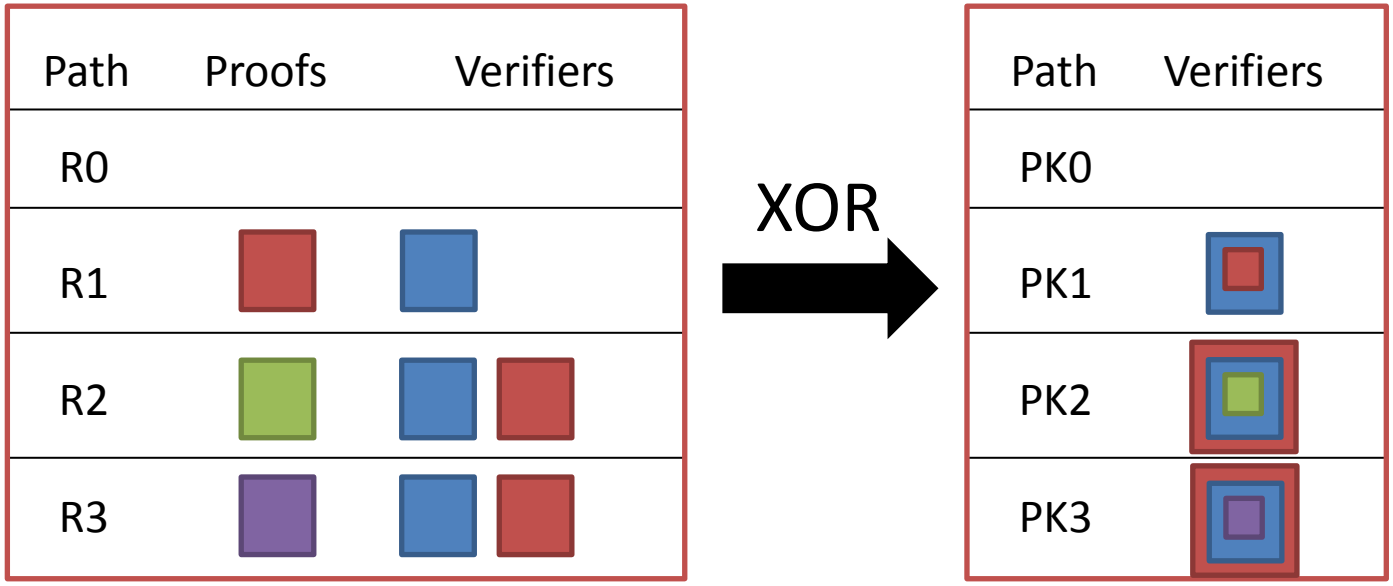
Operational constraints

✓ Adversarial

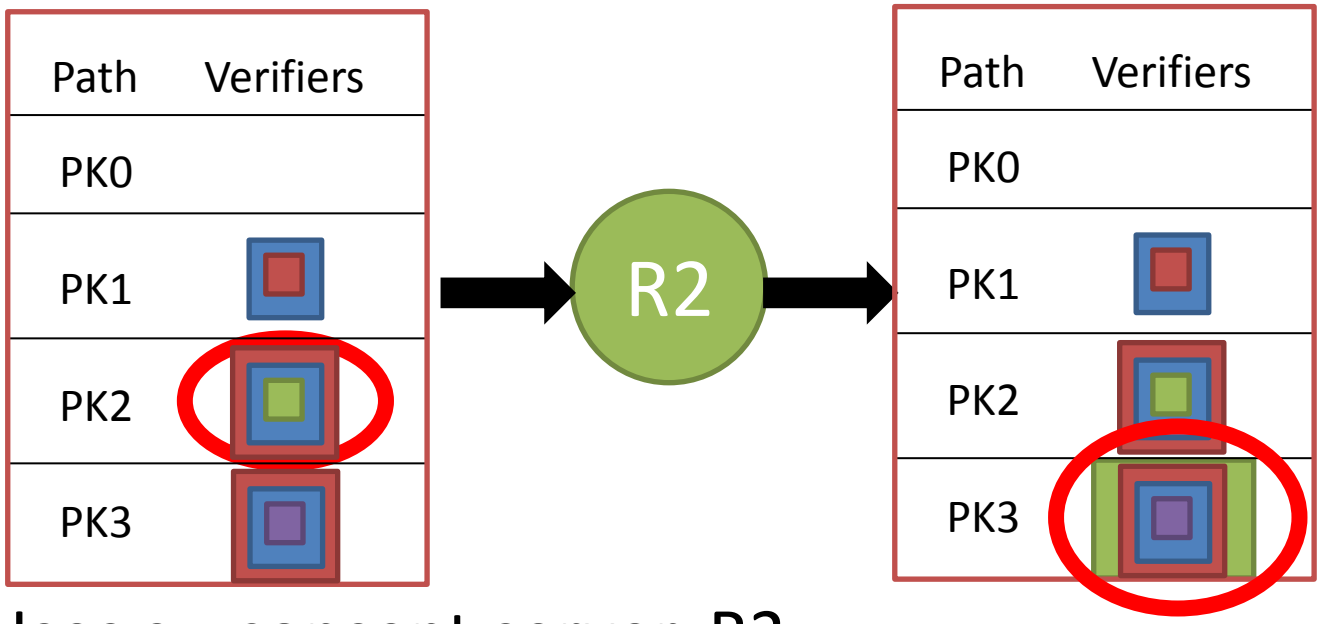
✓ Decentralized

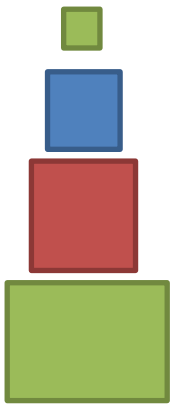
✗ High performance

ICING: Decrease overhead by XORing MACs and Proofs



ICING: public keys as names and pairwise keys non-interactive key exchange





Uses s_2: consent server, R2

Uses k_{02} = KEY-EXCH(SK0, PK2) = KEY-EXCH(SK2, PK0)

Uses k_{12} = KEY-EXCH(SK1, PK2) = KEY-EXCH(SK2, PK1)

Uses k_{23} = KEY-EXCH(SK2, PK3) = KEY-EXCH(SK3, PK2)

Missing functionality: Realm-specific services and delegation

- Indicate entity-specific meaning: QoS, billing, DPI, etc.
- Delegate ability to create proofs

Extend hop specification with tag

- Path

  $<(PK_1, tag_1), (PK_2, tag_2), \ldots, (PK_n, tag_n)>$

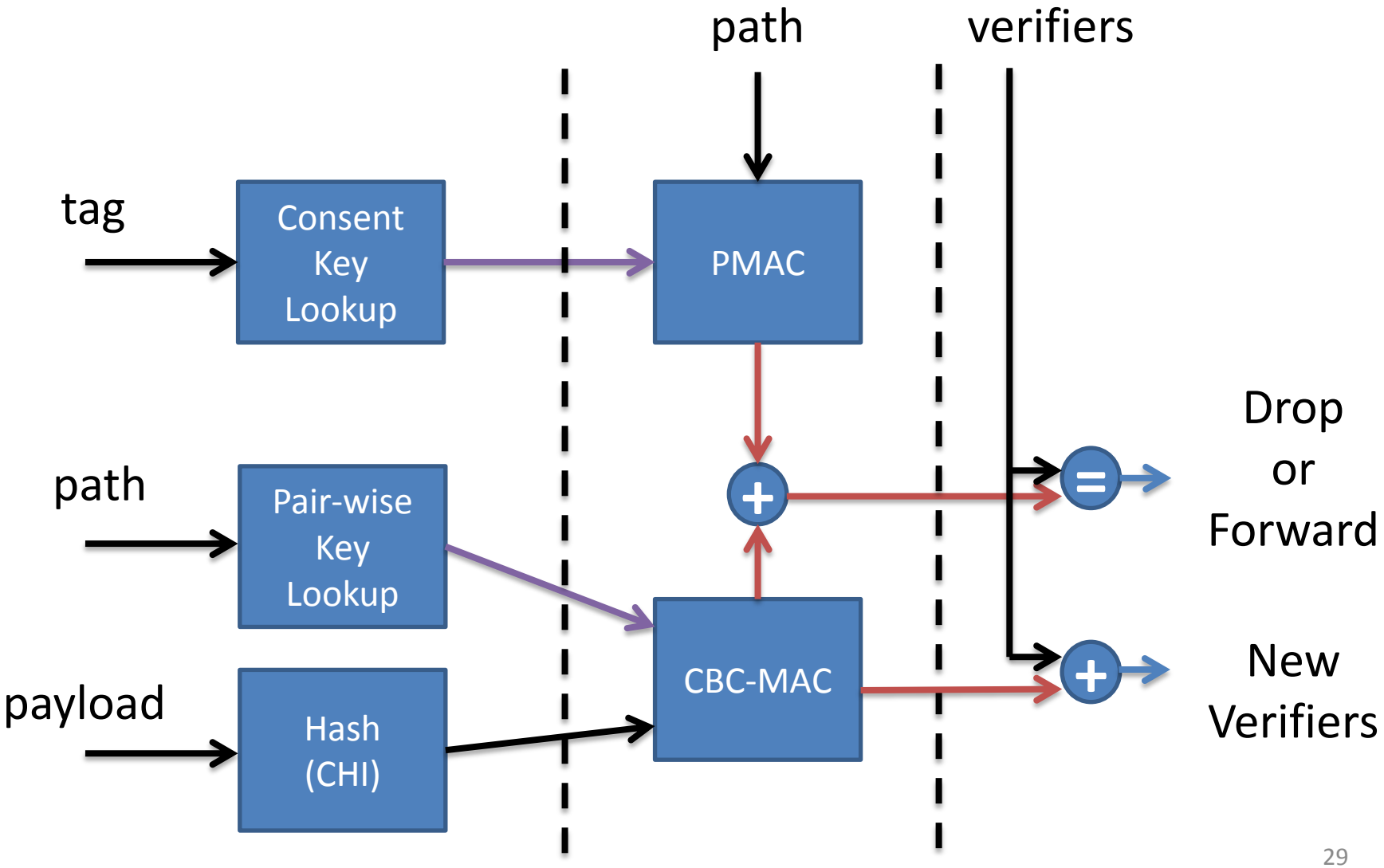
- Each (PK, tag) has a unique consent key (s_i)

- Keys generated from master keys, can be delegated by prefix.

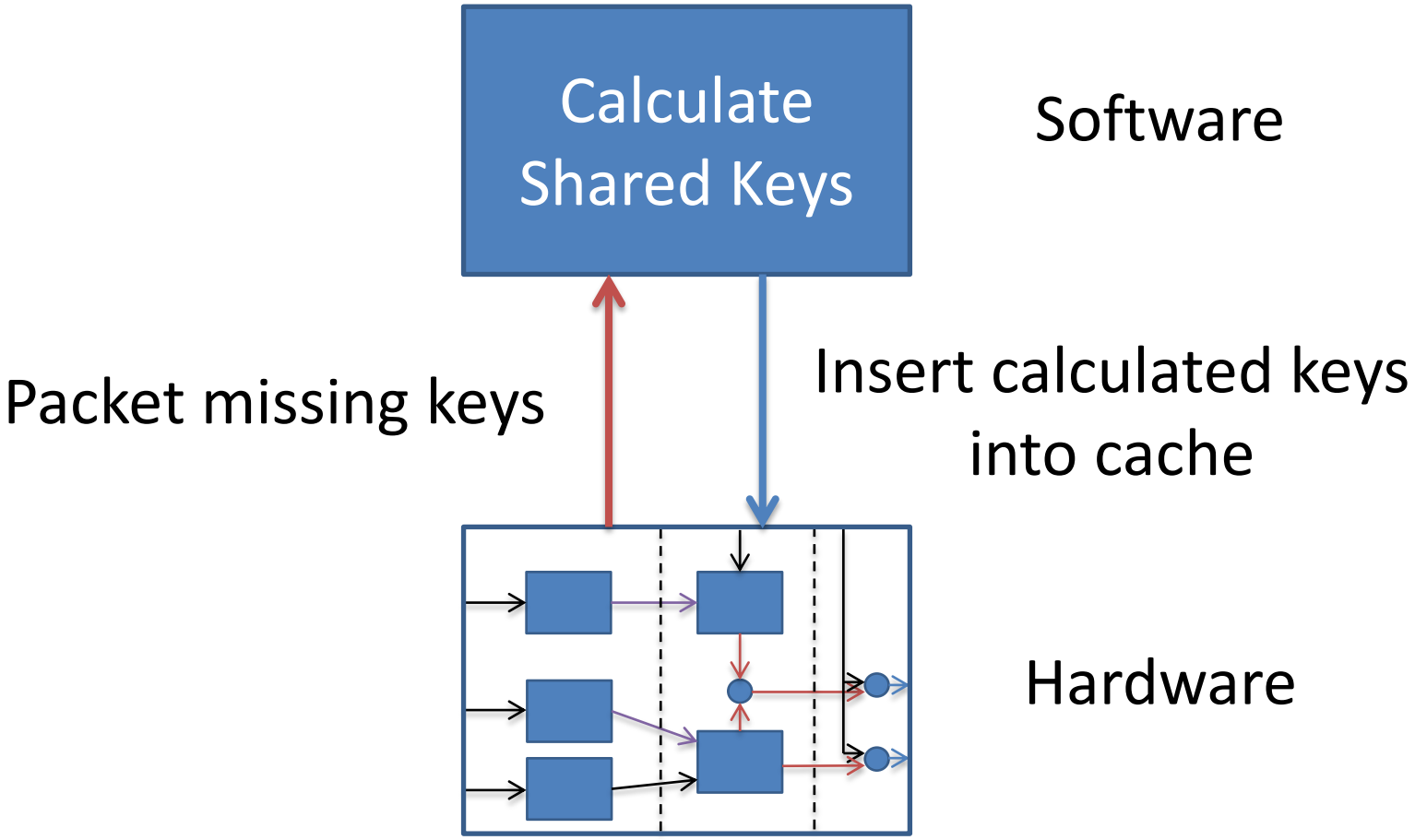
Outline

- Design in three iterations
- Prototype implementation and results
- Related work and conclusion

Hardware implementation uses three main pipeline stages



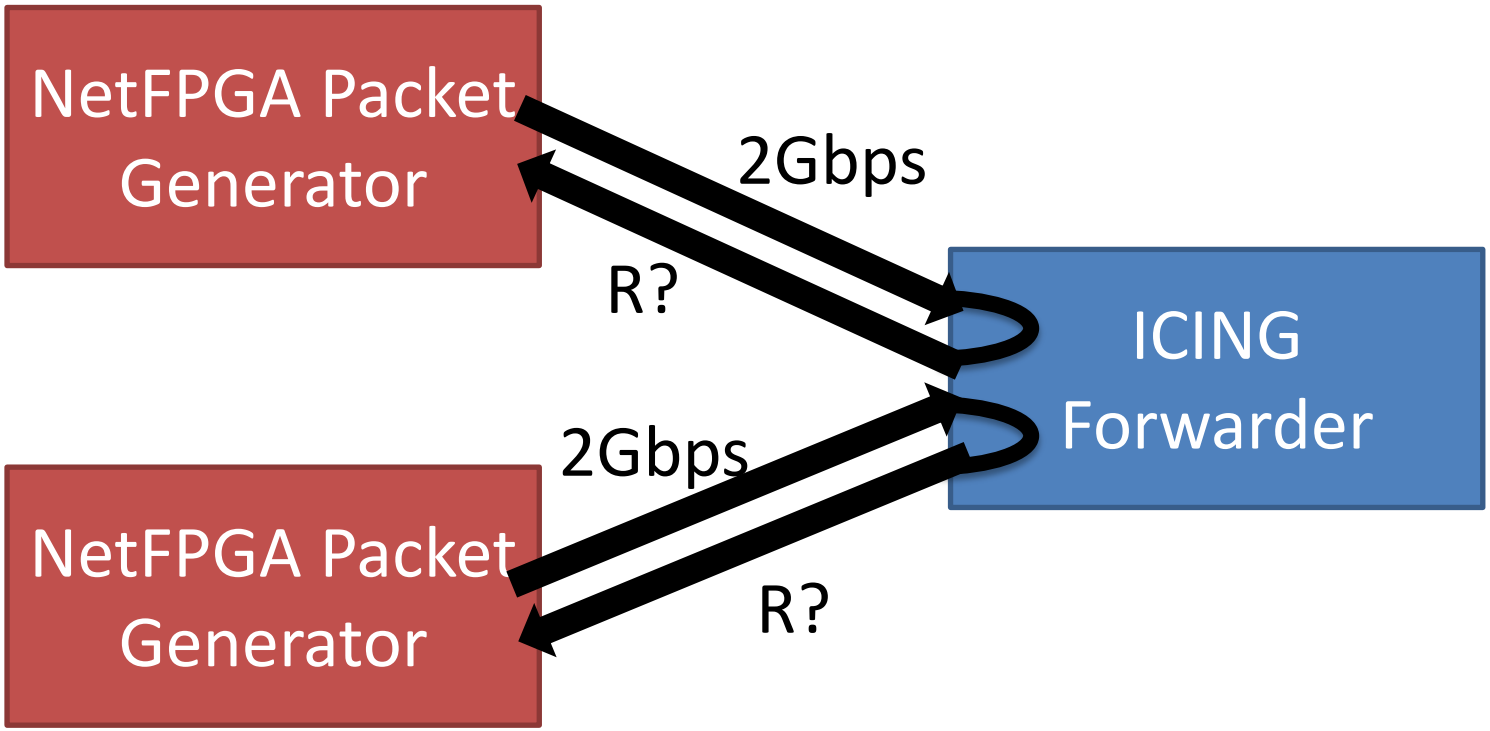
Slow path in software does key exchange



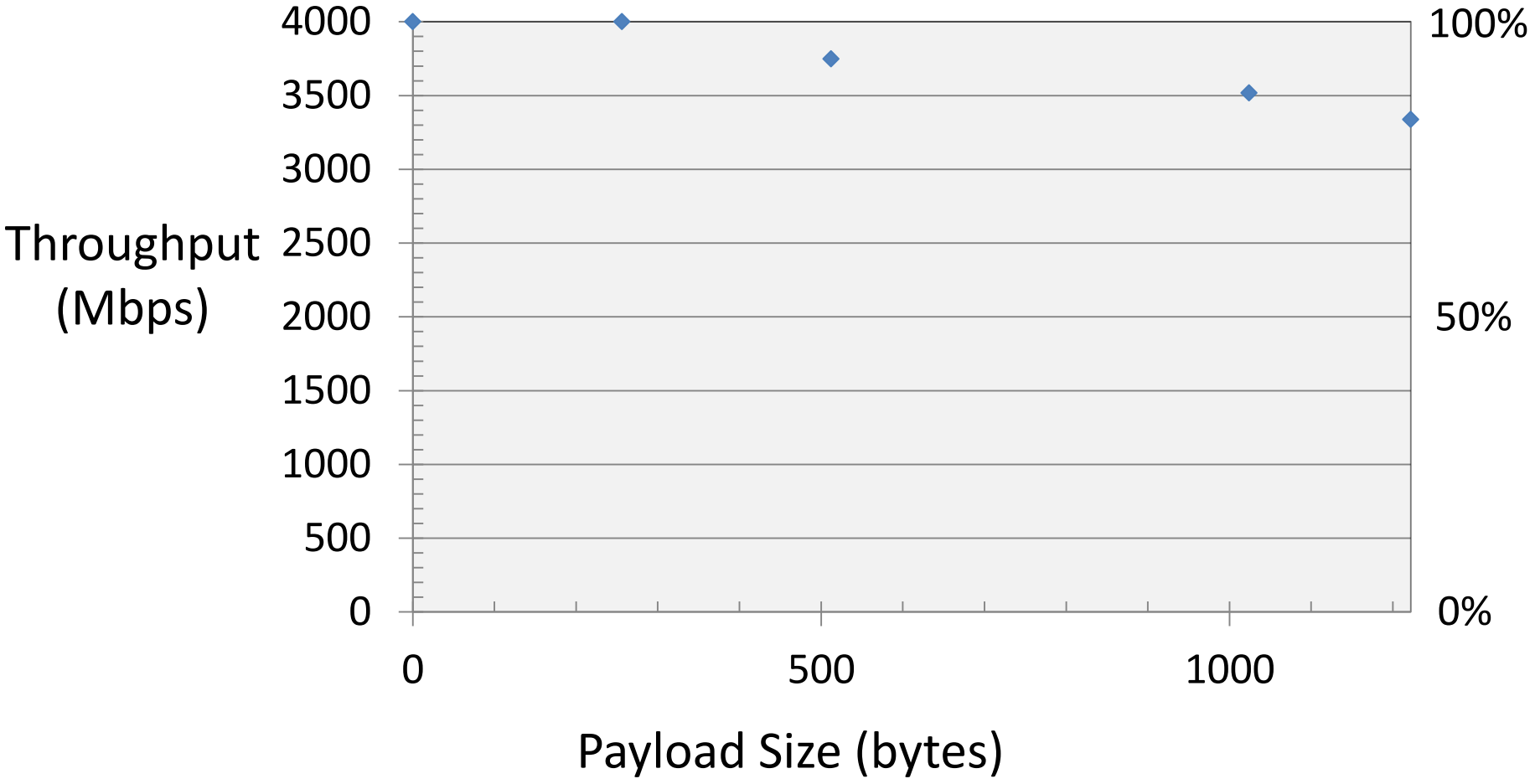
Evaluation questions

- What is throughput of the forwarder? Can it handle whatever packets are thrown at it?
  - Bottleneck at the hash function
- What is the hardware cost of an ICING forwarder?
- How much packet overhead does ICING add?

Throughput: Connect all forwarder ports to NetFPGA packet generators



Throughput vs. Payload Size (Path Length=7)



Hardware Cost

- Measure cost as equivalent gate count generated by Xilinx ISE 10.1i
- Our implementation costs 54% more than NetFPGA IP router and is 20% slower.
- Normalized cost (for the same throughput) is 93% more than NetFPGA IP router.

Packet overhead increase: Estimate from backbone trace

- 15-minute trace from Trans-Pacific 150Mbps line
- Assuming average path length of 5
- ICING would add < 25% more overhead

- 187.5Mbps ICING line = 150Mbps IP line

Outline

- Design in three iterations
- Prototype implementation and results
- Related work and conclusion

Selected related work

- Enriching control and policy:
  - [Calvert et al, *Broadnets* '07] PoMo
  - [Popa et al, *OSDI* '10] RBF
  - [Yang et al, *ACM/IEEE Trans. on networking* '04] NIRA
- Related mechanisms:
  - [Liu et al, *NSDI* '08] Passport
  - [Andersen et al, *SIGCOMM* '08] AIP
  - [Raghavan and Snoeren, *SIGCOMM* '04*]* Platypus

Conclusion

Single primitive with two simple properties can provide functions of many other protocols.

Solving hard problems using scalable per-packet cryptography

Line-rate enforcement and verification at an additional hardware cost of 93% and <25% average packet overhead