Secure Network Provenance

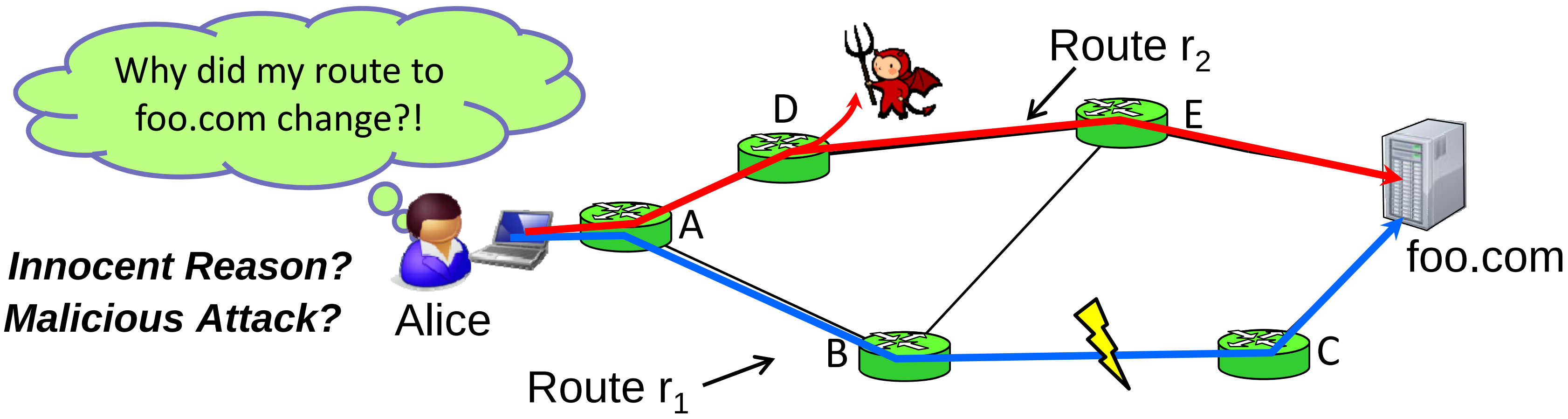
*Wenchao Zhou**, *Qiong Fei**, *Arjun Narayan**,

*Andreas Haeberlen**, *Boon Thau Loo**, *Micah Sherr*+

*University of Pennsylvania +Georgetown University

http://snp.cis.upenn.edu/

Motivation



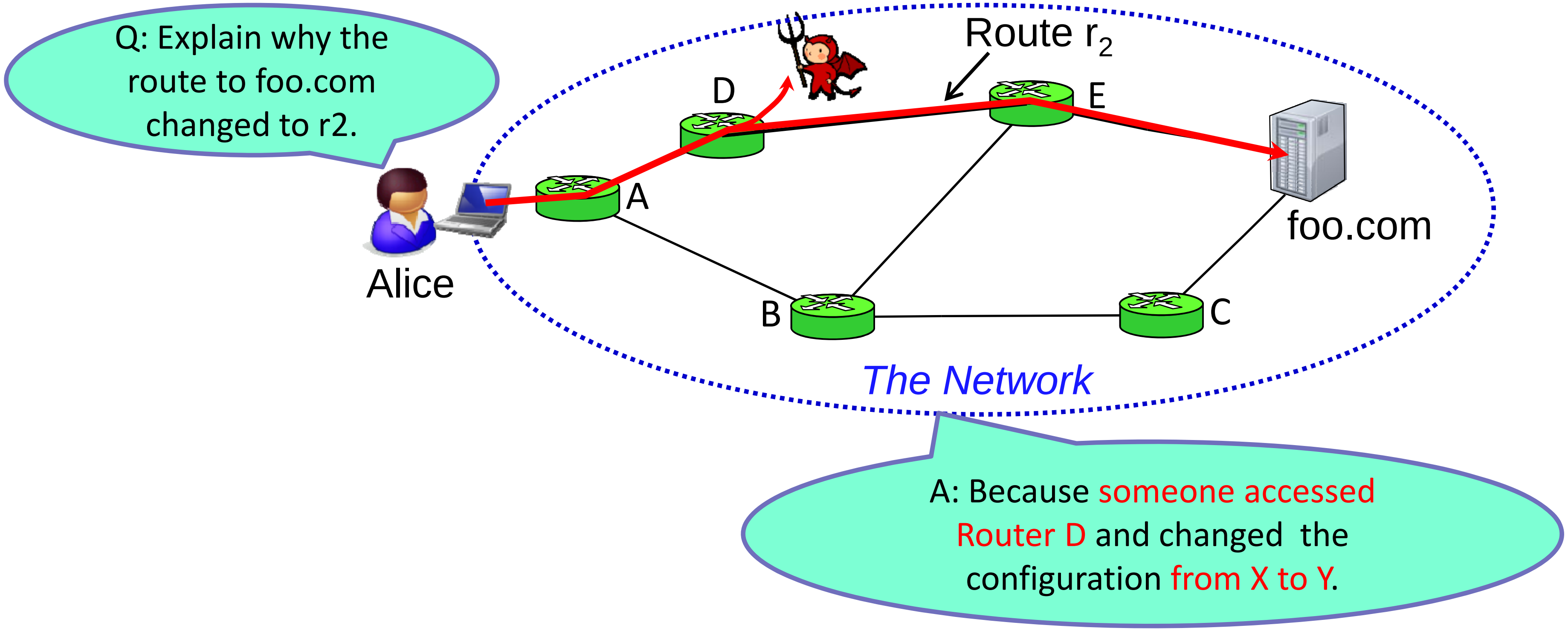
- **An example scenario: network routing**
  - System administrator observes strange behavior
  - Example: the route to foo.com has suddenly changed
  - ***What exactly happened (innocent reason or malicious attack)?***

We Need Secure Forensics

- **For network routing …**
  - Example: incident in March 2010
    - Traffic from Capitol Hill got redirected
- **… but also for other application scenarios**
  - Distributed hash table: Eclipse attack
  - Cloud computing: misbehaving machines
  - Online multi-player gaming: cheating

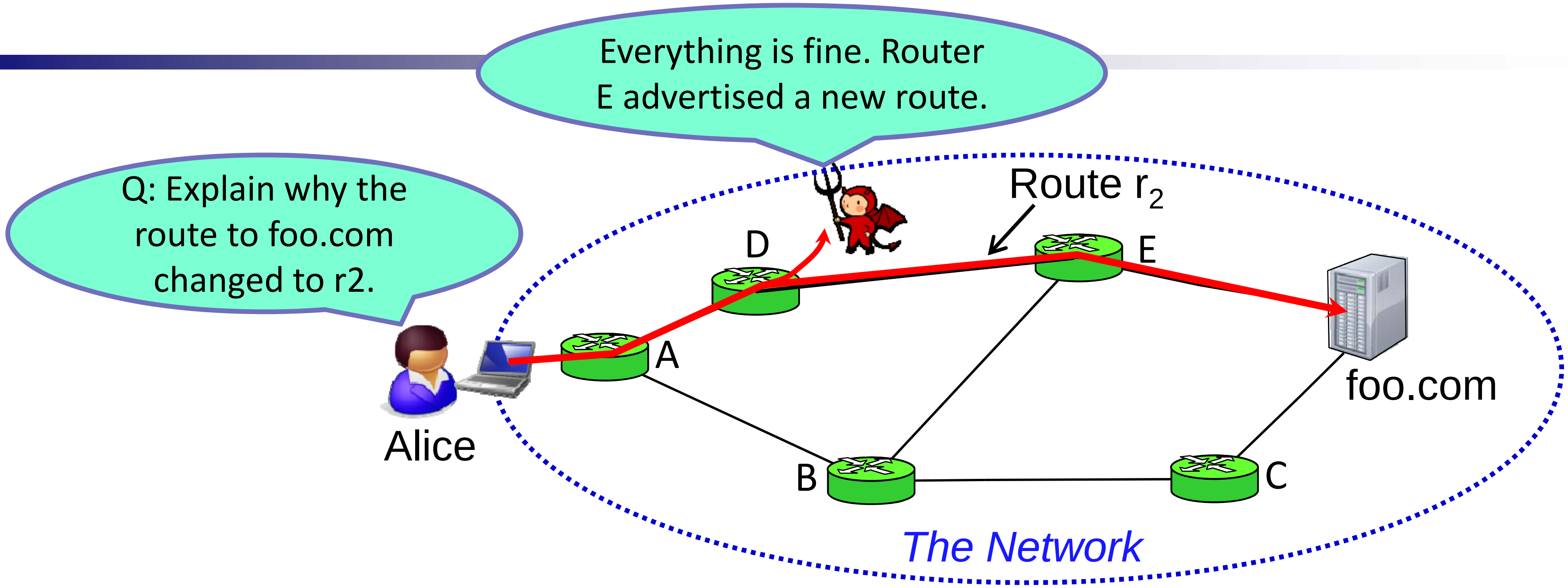
- **Goal: secure forensics in adversarial scenarios**

Ideal Solution



- **Not realistic: adversary can tell lies**

Challenge: Adversaries Can Lie



- **Problem: adversary can …**
  - ... fabricate plausible (yet incorrect) response
  - … point accusation towards innocent nodes

Existing Solutions

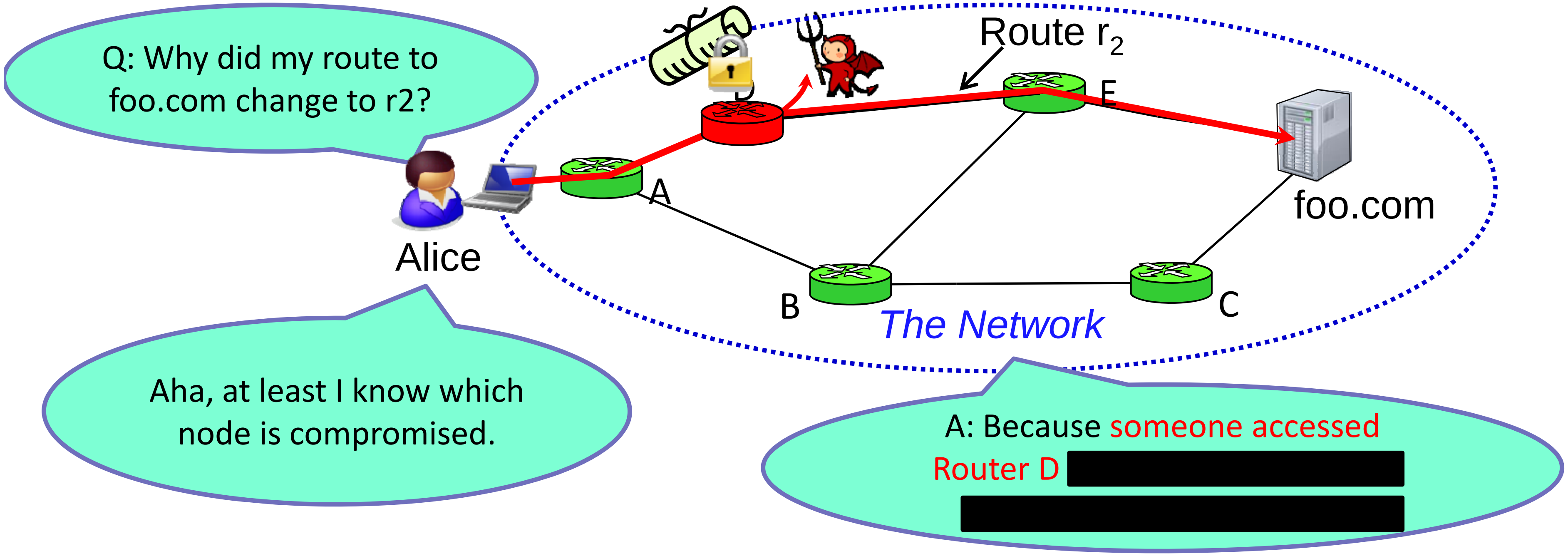
- **Existing systems assume trusted components**
  - Trusted OS kernel, monitor, or hardware
    - E.g. Backtracker [OSDI 06], PASS [USENIX ATC 06], ReVirt [OSDI 02], A2M [SOSP 07]
  - These components may have bugs or be compromised
  - ***Are there alternatives that do not require such trust?***
- **Our solution:**
  - We assume no trusted components;
  - Adversary has full control over an arbitrary subset of the network (Byzantine faults).

Ideal Guarantees

Fundamentally impossible

- **Ideally: explanation is always complete and accurate**
- **Fundamental limitations**
  - E.g. Faulty nodes secretly exchange messages
  - E.g. Faulty nodes communicate outside the system
- ***What guarantees can we provide?***

Realistic Guarantees



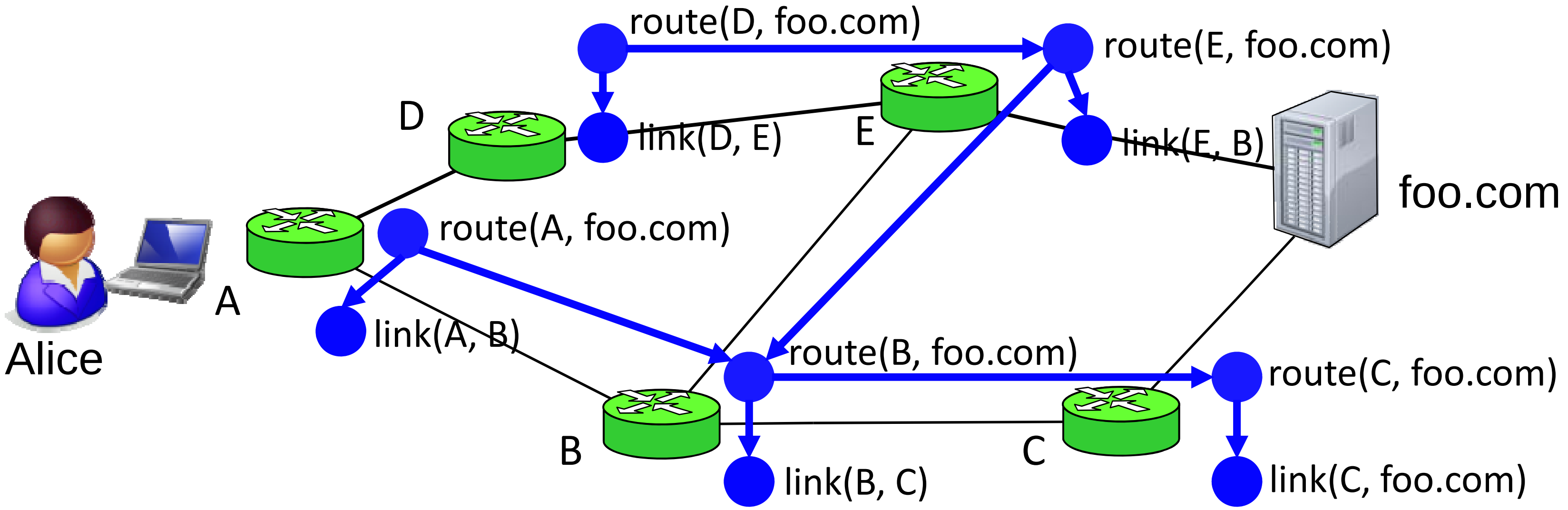
- **No faults: Explanation is complete and accurate**
- **Byzantine fault: Explanation identifies at least one faulty node**
- ***Formal definitions and proofs in the paper***

Outline

- **Goal: A secure forensics system that works in an adversarial environment**
  - Explains unexpected behavior
  - No faults: explanation is complete and accurate
  - Byzantine fault: exposes at least one faulty node with evidence

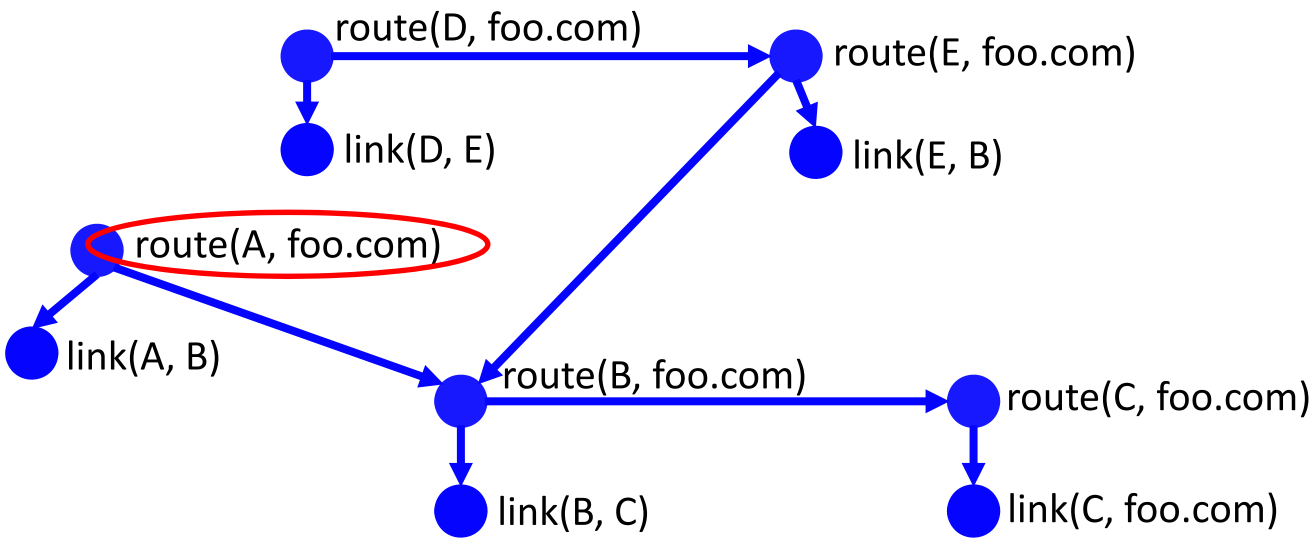
- ***Model: Secure Network Provenance***
- **Tamper-evident Maintenance and Processing**
- **Evaluation**
- **Conclusion**

Provenance as Explanations



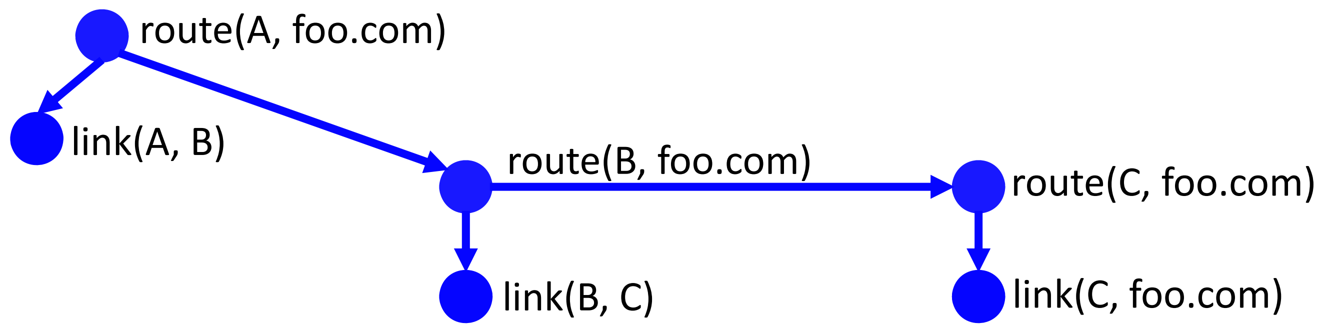
- **Origin: data provenance in databases**
  - Explains the derivation of tuples (ExSPAN [SIGMOD 10])
  - Captures the dependencies between tuples as a graph
  - "Explanation" of a tuple is a tree rooted at the tuple

Provenance as Explanations



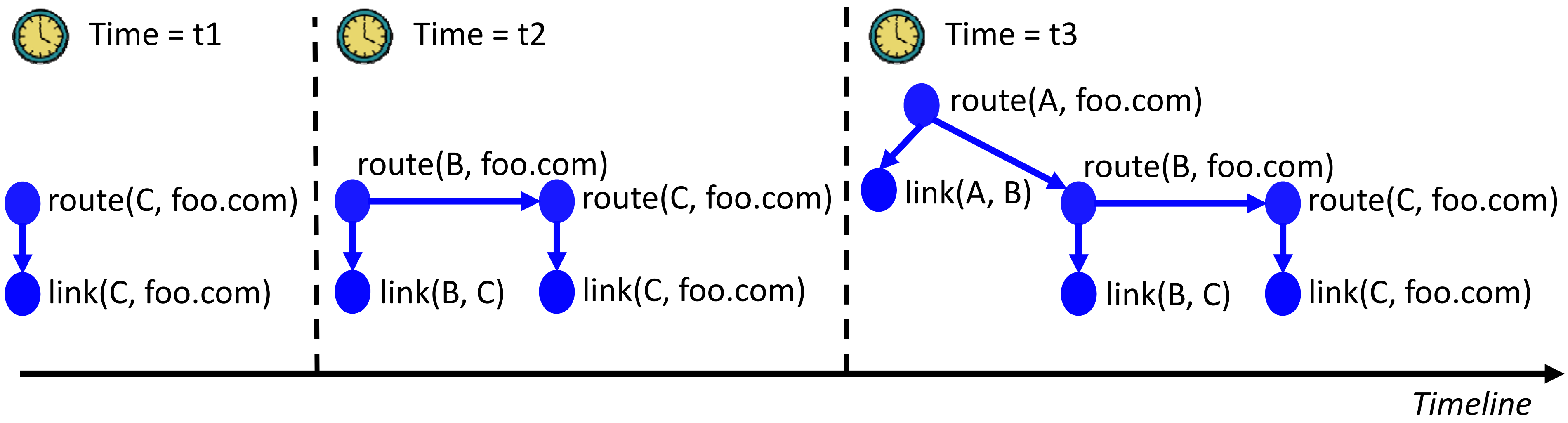
- **Origin: data provenance in databases**
  - Explains the derivation of tuples (ExSPAN [SIGMOD 10])
  - Captures the dependencies between tuples as a graph
  - "Explanation" of a tuple is a tree rooted at the tuple

Secure Network Provenance



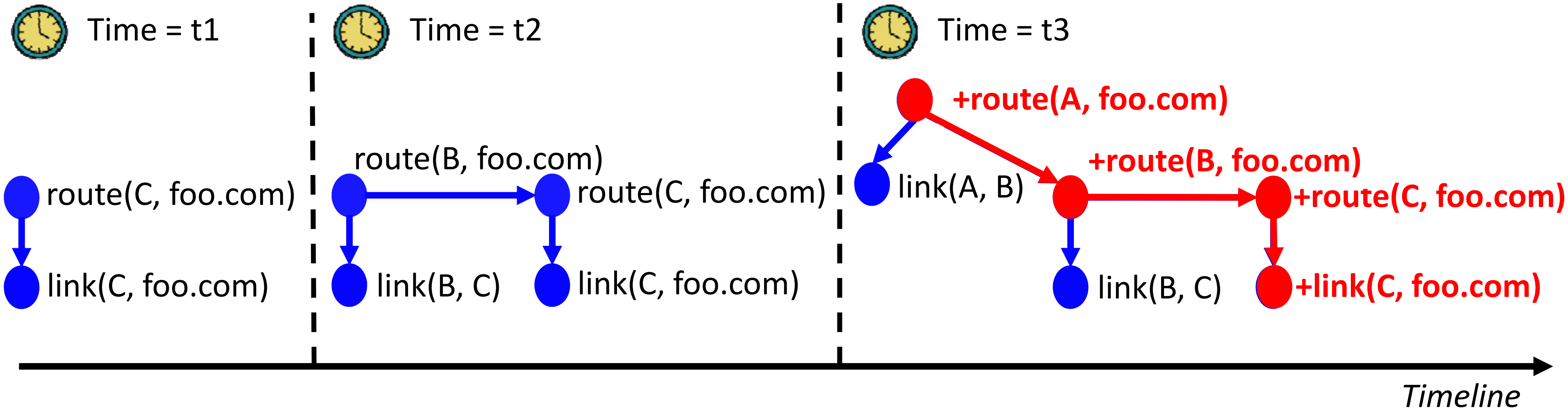
- **Challenge #1. Handle past and transient behavior**
  - Traditional data provenance targets current, stable state
  - What if the system never converges?
  - What if the state no longer exists?

Secure Network Provenance



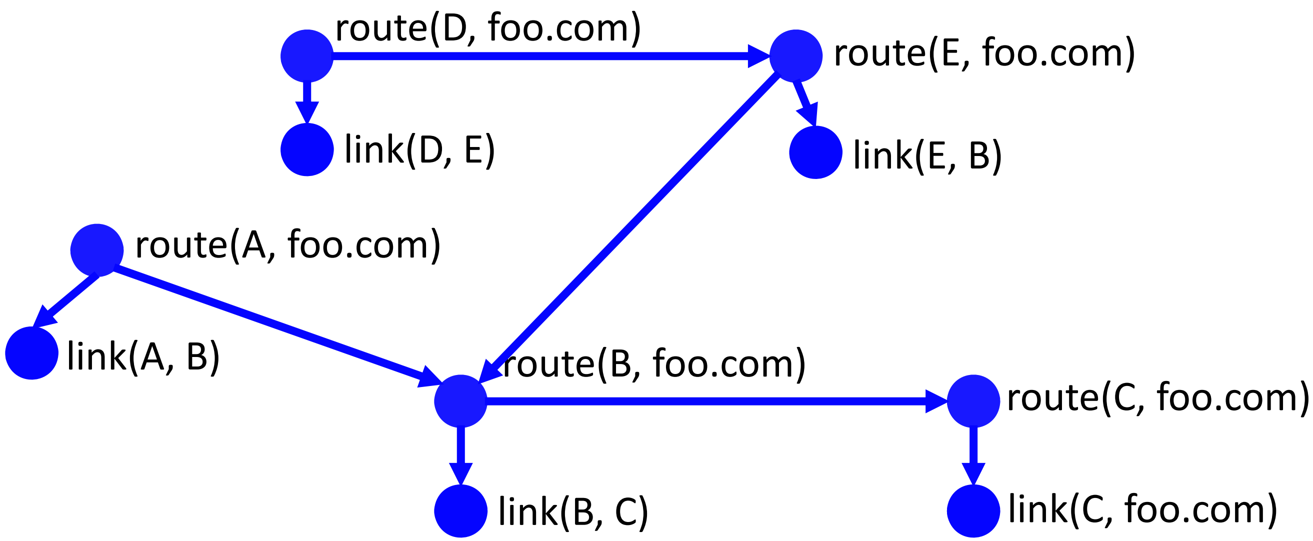
- **Challenge #1. Handle past and transient behavior**
  - Traditional data provenance targets current, stable state
  - What if the system never converges?
  - What if the state no longer exists?
  - *Solution: Add a temporal dimension*

Secure Network Provenance



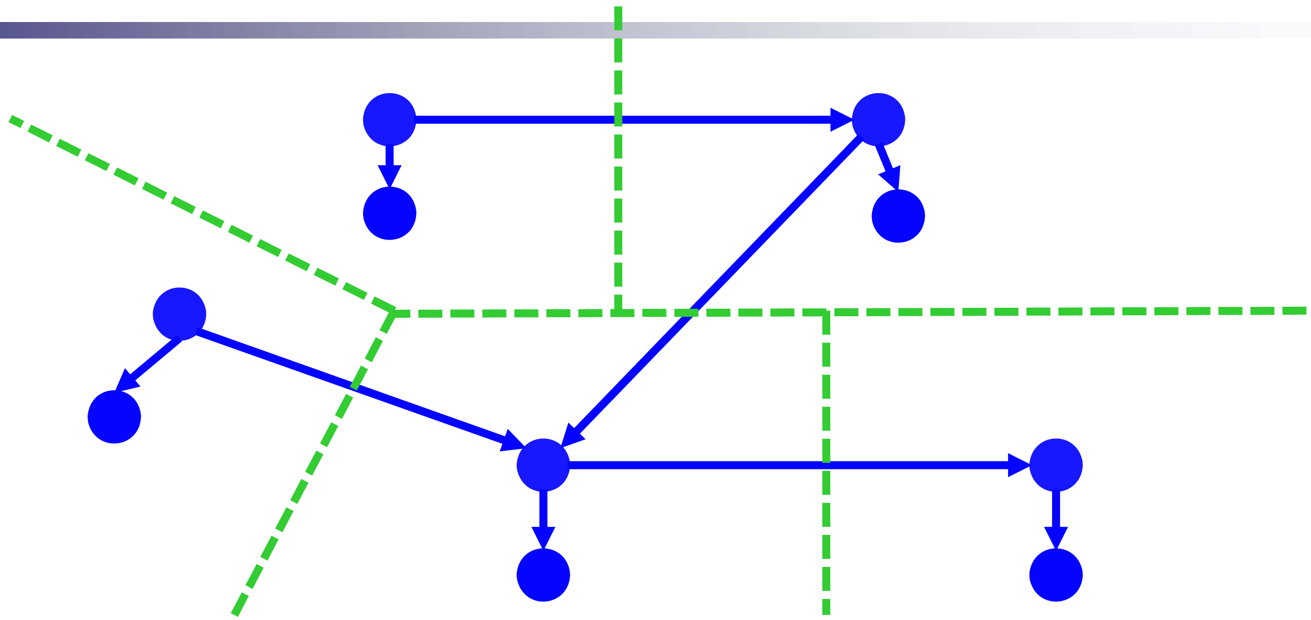
- **Challenge #2. Explain changes, not just state**
  - Traditional data provenance targets system state
  - Often more useful to ask why a tuple (dis)appeared
  - *Solution: Include "deltas" in provenance*

Secure Network Provenance



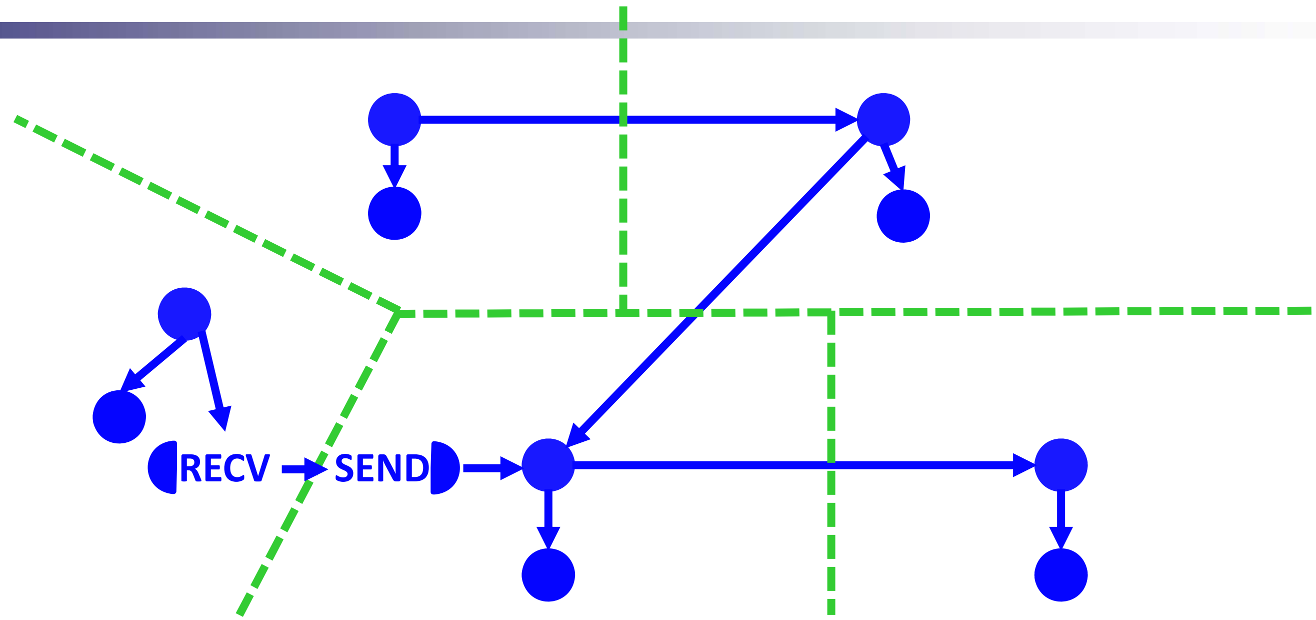
- **Challenge #3. Partition and secure provenance**
  - A trusted node would be ideal, but we don't have one
  - Need to partition the graph among the nodes themselves
  - Prevent nodes from altering the graph

Partitioning the Provenance Graph



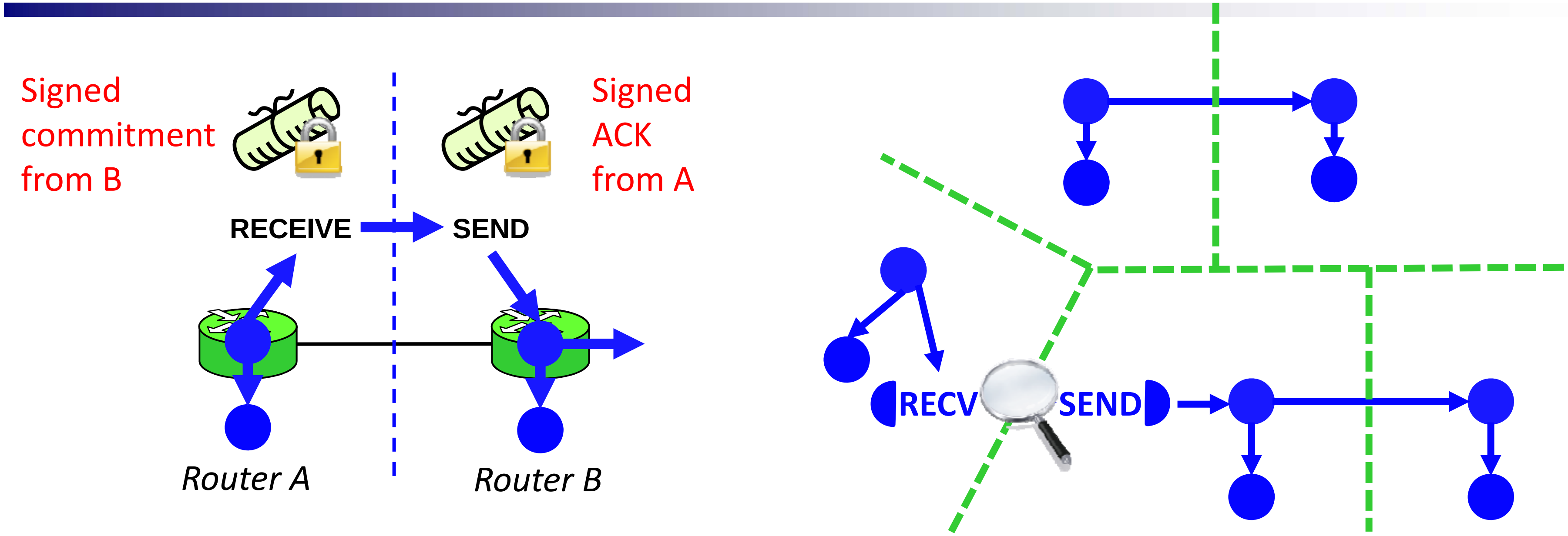
- **Step 1: Each node keeps vertices about local actions**
  - Split cross-node communications

Partitioning the Provenance Graph



- **Step 1: Each node keeps vertices about local actions**
  - ☐ Split cross-node communications
- **Step 2: Make the graph tamper-evident**

Securing Cross-Node Edges

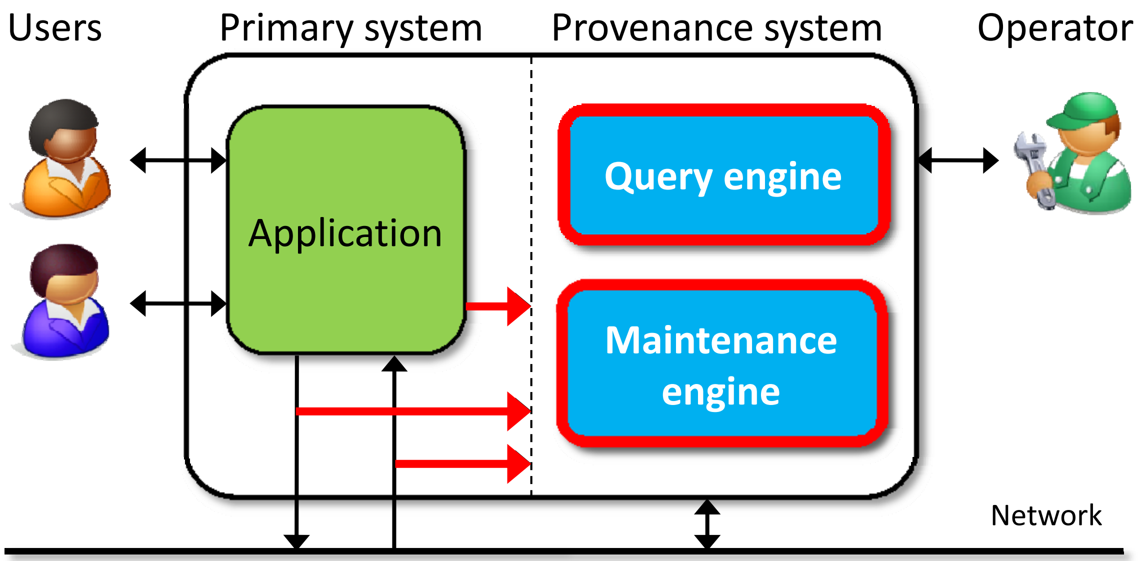


- **Step 1: Each node keeps vertices about local actions**
  - Split cross-node communications
- **Step 2: Make the graph tamper-evident**
  - Secure cross-node edges (evidence of omissions)

Outline

- **Goal: A secure forensics system that works in an adversarial environment**
  - Explains unexpected behavior
  - No faults: explanation is complete and accurate
  - Byzantine fault: exposes at least one faulty node with evidence
- **Model: Secure Network Provenance**
- ***Tamper-evident Maintenance and Processing***
- **Evaluation**
- **Conclusion**

System Overview



Extract provenance

Maintain provenance

Query provenance

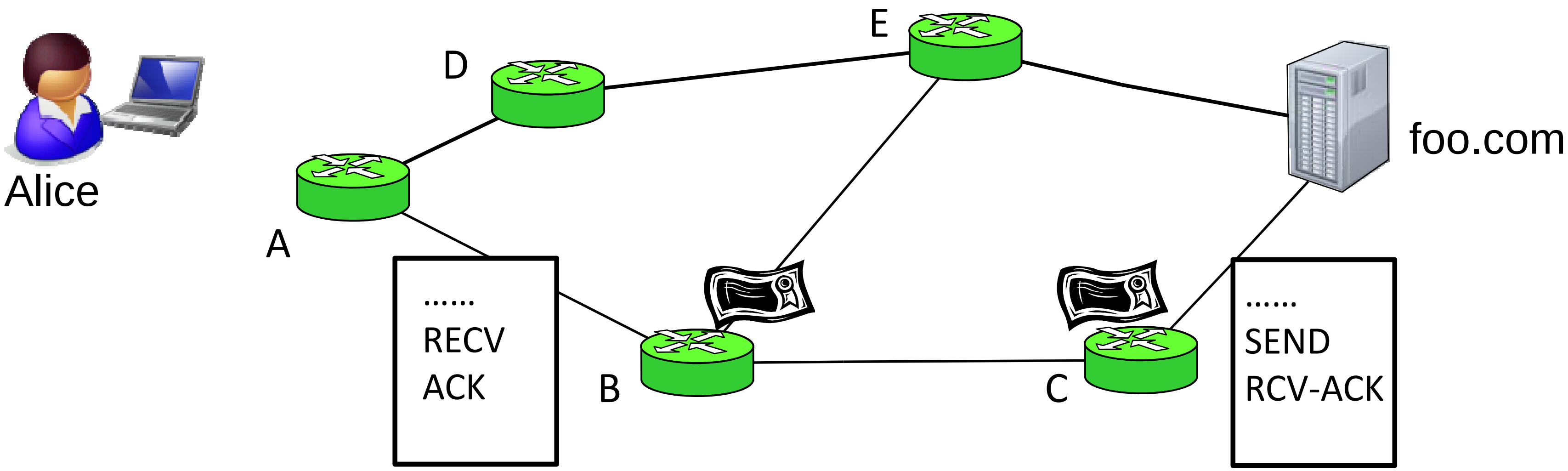
- **Stand-alone provenance system**
- **On-demand provenance reconstruction**
  - Provenance graph can be huge (with temporal dimension)
  - Rebuild only the parts needed to answer a query

Extracting Dependencies

- **Option 1: Inferred provenance**
  - Declarative specifications explicitly capture provenance
  - E.g. Declarative networking, SQL queries, etc.
- **Option 2: Reported provenance**
  - Modified source code reports provenance
- **Option 3: External specification**
  - Defined on observed I/Os of a black-box system

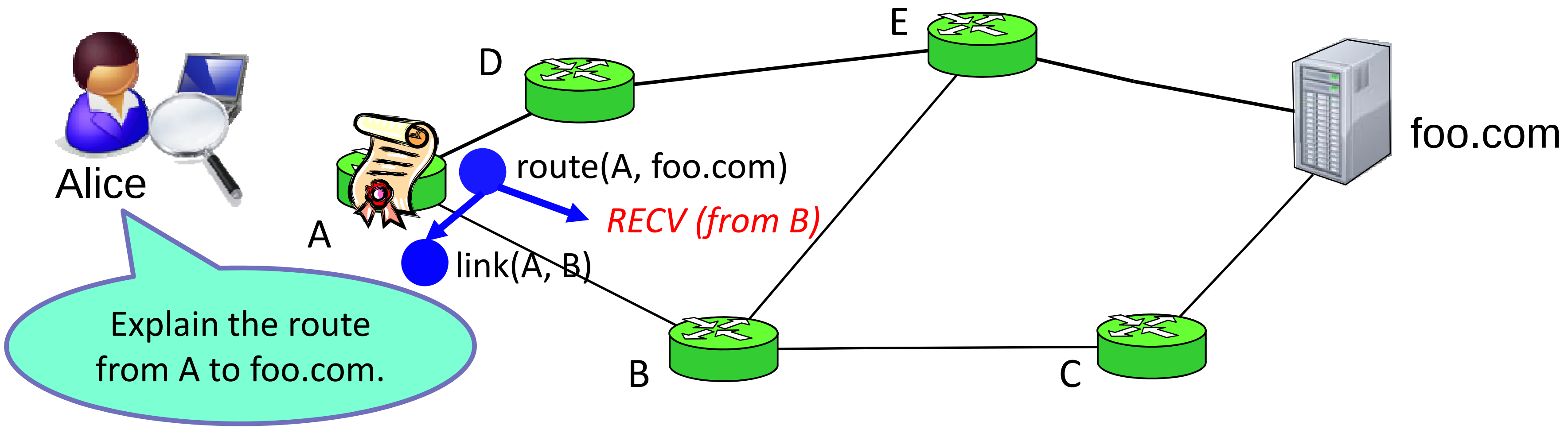
Secure Provenance Maintenance

- **Maintain sufficient information for reconstruction**
  - I/O and non-deterministic events are sufficient
  - Logs are maintained using tamper-evident logging
    - Based on ideas from PeerReview [SOSP 07]



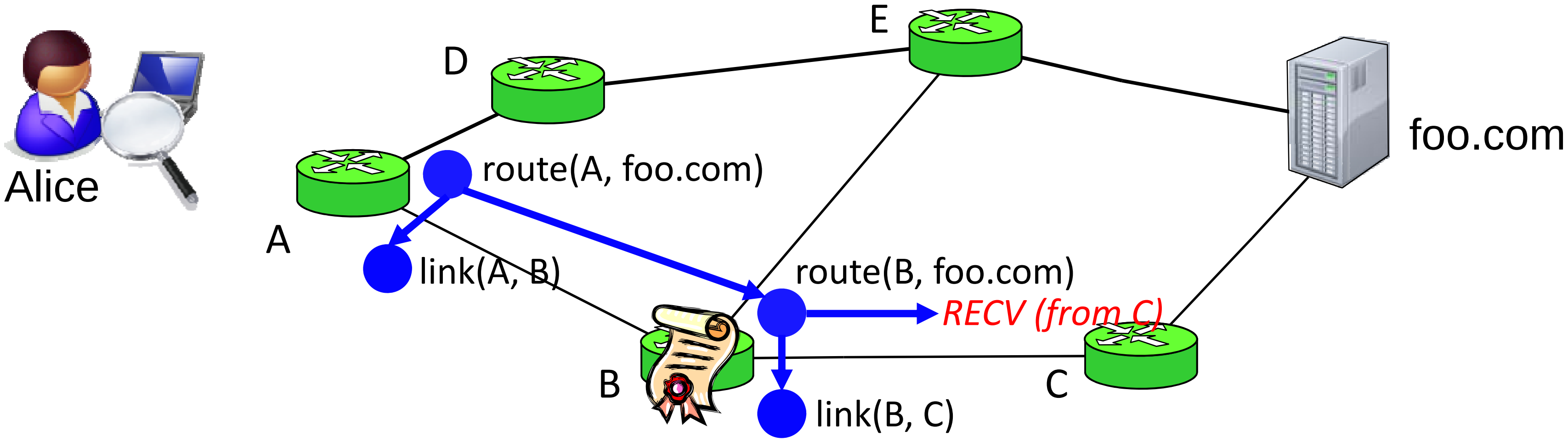
Secure Provenance Querying

- **Recursively construct the provenance graph**
  - Retrieve secure logs from remote nodes
  - Check for tampering, omission, and equivocation
  - Replay the log to regenerate the provenance graph



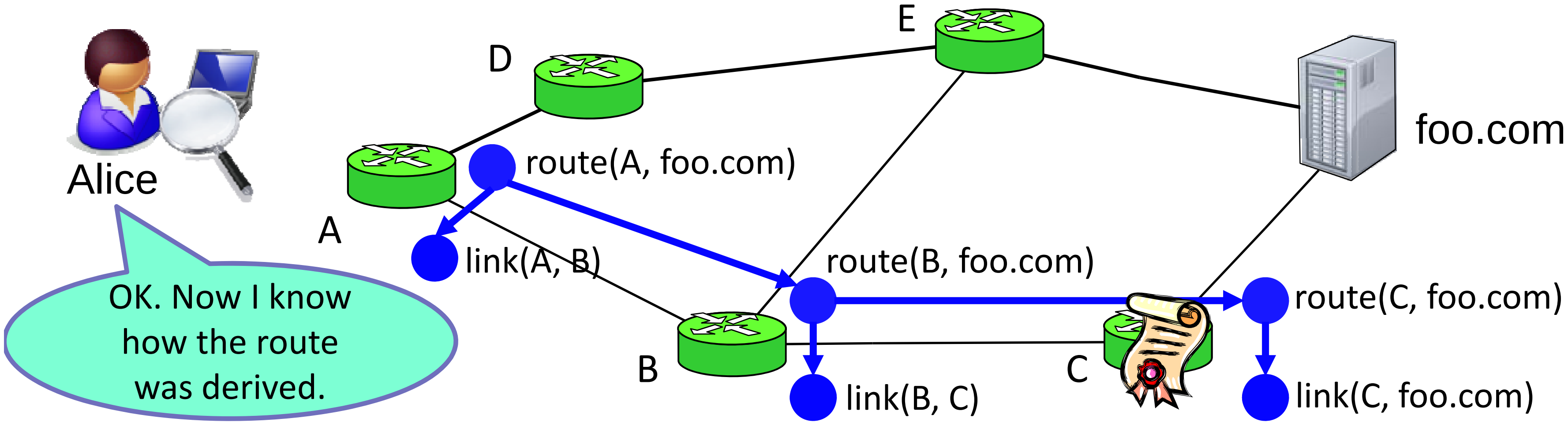
Secure Provenance Querying

- **Recursively construct the provenance graph**
  - Retrieve secure logs from remote nodes
  - Check for tampering, omission, and equivocation
  - Replay the log to regenerate the provenance graph



Secure Provenance Querying

- **Recursively construct the provenance graph**
  - Retrieve secure logs from remote nodes
  - Check for tampering, omission, and equivocation
  - Replay the log to regenerate the provenance graph



Outline

- **Goal: A secure forensics system that works in an adversarial environment**
  - Explains unexpected behavior
  - No faults: explanation is complete and accurate
  - Byzantine fault: exposes at least one faulty node with evidence
- **Model: Secure Network Provenance**
- **Tamper-evident Maintenance and Processing**
- ***Evaluation***
- **Conclusion**

Evaluation Results

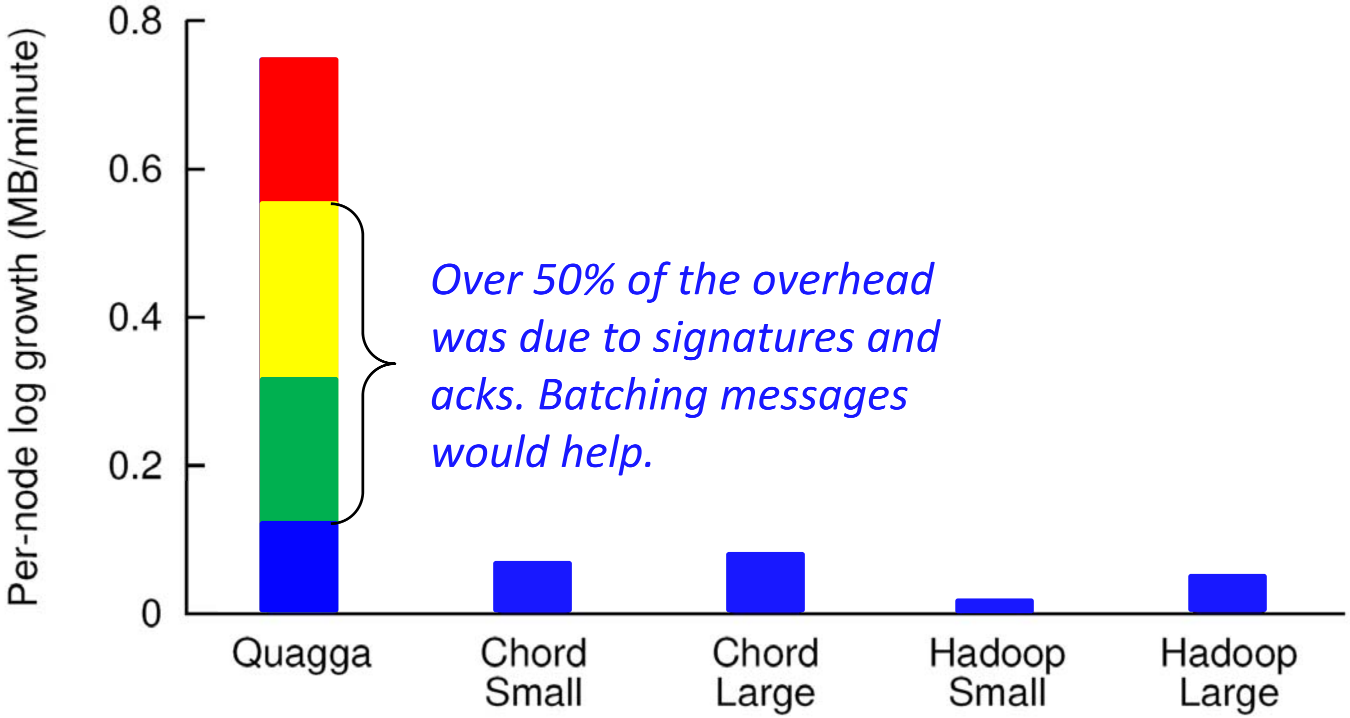
- **Prototype implementation (SNooPy)**
  - How useful is SNP? Is it applicable to different systems?
  - How expensive is SNP at runtime?
    - Traffic overhead, storage cost, additional CPU overhead?
    - Does SNP affect scalability?
  - What is the querying performance?
    - Per-query traffic overhead?
    - Turnaround time for each query?

Usability: Applications

- **We evaluated SNooPy with**
  - Quagga BGP: RouteView (external specification)
    - Explains oscillation caused by router misconfiguration
  - Hadoop MapReduce: (reported provenance)
    - Detects a tampered Mapper that returns inaccurate results
  - Declarative Chord DHT: (inferred provenance)
    - Detects an Eclipse attacker that always returns its own ID

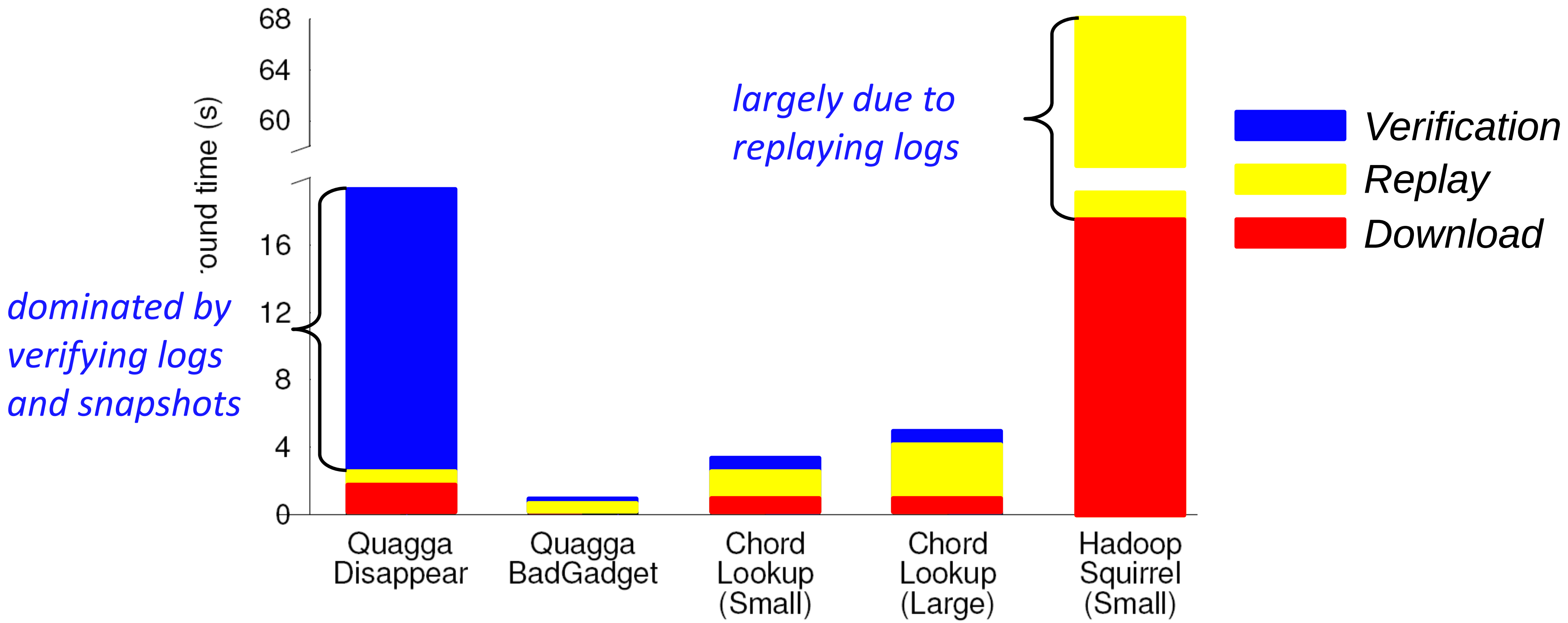
- **SNooPy solves problems reported in existing work**

Runtime Overhead: Storage



- **Manageable storage overhead**
  - One week of data: E.g. Quagga – 7.3GB; Chord – 665MB

Query Latency



- **Query latency varies from application to application**
- **Reasonable overhead**

Summary

- **Secure network provenance in untrusted environments**
  - Requires no trusted components
  - Strong guarantees even in the presence of Byzantine faults
    - Formal proof in a technical report
  - Significantly extends traditional provenance model
    - Past and transient state, provenance of change, …
  - Efficient storage: reconstructs provenance graph on demand
  - Application-independent (Quagga, Hadoop, and Chord)
- **Questions?**

Project website: http://snp.cis.upenn.edu/