NetQuery: A Knowledge Plane for Reasoning about Network Properties

Alan Shieh Emin Gün Sirer Fred B. Schneider
Department of Computer Science, Cornell University
Ithaca, NY, USA
{ashieh, egs, fbs}@cs.cornell.edu

ABSTRACT

This paper presents the design and implementation of NetQuery, a knowledge plane for federated networks such as the Internet. In such networks, not all administrative domains will generate information that an application can trust and many administrative domains may have restrictive policies on disclosing network information. Thus, both the trustworthiness and accessibility of network information pose obstacles to effective reasoning. NetQuery employs trustworthy computing techniques to facilitate reasoning about the trustworthiness of information contained in the knowledge plane while preserving confidentiality guarantees for operator data. By characterizing information disclosure between operators, NetQuery enables remote verification of advertised claims and contractual stipulations; this enables new applications because network guarantees can span administrative boundaries. We have implemented NetQuery, built several NetQuery-enabled devices, and deployed applications for cloud datacenters, enterprise networks, and the Internet. Simulations, testbed experiments, and a deployment on a departmental network indicate NetQuery can support hundreds of thousands of operations per second and can thus scale to large ISPs.

Categories and Subject Descriptors

C.2.3 [**Computer-Communication Networks**]: Network Operations—*network management, network monitoring*

General Terms

Design, Management

1. INTRODUCTION

Depending on their configuration, administration, and provisioning, networks provide drastically different features. For instance, some networks provide little failure resilience, while others provision failover capacity and deploy middleboxes to protect against denial of service attacks [13, 6]. Agreements between network operators often include requirements that are governed by such network features. Peering and service agreements, for example, can mandate topology, reliability, and forwarding policies, while terms-of-use agreements can mandate end host deployment of up-to-date security mechanisms. Yet the standard IP interface masks the differences between networks; each network appears to provide the same, undifferentiated "dial-tone" service. Consequently, clients and networks must resort to ad hoc techniques for advertising the quality of a network and for reasoning about meta-level properties of the network.

Knowledge planes [17] have been proposed to disseminate these underlying network features, thereby giving providers a channel for advertising network capabilities and enabling applications to use reasoning to find suitable networks for their requirements. This paper describes *NetQuery*, a knowledge plane for multi-operator networks, like the Internet. To our knowledge, NetQuery is the first instantiation of a knowledge plane for federated networks comprised of mutually untrusting administrative domains. NetQuery disseminates information (e.g., routing tables, neighbor lists, and configurations) about network entities (e.g., routers, switches, and end hosts) to support application-level reasoning. NetQuery enables reasoning while simultaneously respecting the information disclosure policies of participants, which may be hesitant to share information in a federated environment. Under NetQuery, reasoning can occur between mutually untrusting entities without leaking confidential information.

NetQuery applications act only on information from sources that they trust; NetQuery binds each property to a responsible principal. In environments where networked components are equipped with secure coprocessors, such as the Trusted Platform Module (TPM) [21], NetQuery can use this hardware as a root of trust for reasoning about information sources.

This paper outlines the design and implementation of NetQuery. First, it proposes mechanisms for disseminating and discovering facts about the state of network elements. These mechanisms also enable applications to track changes to such state. Second, it outlines a logic-based framework for using such facts, supported by attribution information, to assure that the network possesses an application-specific characteristic. NetQuery leverages trustworthy computing to achieve expressive reasoning while respecting the confidentiality requirements of each network operator. Finally, it describes our implementation of NetQuery devices and applications, as well as our experiences running NetQuery in simulated ISP networks and on an operational departmental network of 73 L2 and L3 Ethernet switches and over 700 end hosts. We show that NetQuery applications can derive guarantees about network performance that cannot be achieved with existing monitoring-based approaches. We also show that NetQuery imposes little overhead and supports topology sizes and event rates encountered in typical service provider networks. To wit, a single NetQuery server can handle 500,000

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGCOMM'11, August 15–19, 2011, Toronto, Ontario, Canada.
Copyright 2011 ACM 978-1-4503-0797-0/11/08 ...$10.00.

requests per second and store the full forwarding table state of an entire Internet POP.

The remainder of this paper describes the design and implementation of NetQuery. Section 2 describes the motivation for NetQuery and outlines the design. Section 3 discusses the NetQuery data model. Section 4 describes how the data model is coupled with a logical framework and with TPMs to support inference. Section 5 describes the security model, its architectural implications, and how analyses can determine that information sources are trustworthy. Section 6 describes the incremental deployment benefits of NetQuery. Section 7 describes and evaluates our implementation and applications. Section 8 presents related work and Section 9 concludes.

2. MOTIVATION AND OVERVIEW

Knowledge planes enable new applications that depend on reasoning about properties of the network. By providing mechanisms for determining the characteristics of a network, such as the expected level of performance, redundancy, or confidentiality, NetQuery enables network participants (e.g., peers, providers, or customers) to make better informed decisions when establishing sessions, entering into contracts with one another, or verifying compliance.

Sound network reasoning improves network transparency and accountability, facilitating many types of commercial transactions. On the Internet, the price that an operator can charge depends on the paths and performance that they advertise; since routing traffic on a different path may result in lower cost, operators are incentivized to deviate from their advertised behavior to maximize profit [26]. Existing reputation-based mechanisms have not proven sufficient to constrain selfish operators. Indeed, consumer advocates often accuse last mile ISPs of misrepresenting the quality and capacity of their networks [10]. Community forums often advise users to verify independently whether their datacenter provider is truly multihomed [18]. Differences between networks are advertised through manual, ad hoc channels such as interstitial web pages. Absent automatic mechanisms for discovering and disseminating claims about network capacity and redundancy, agreements are difficult and costly to enact and competitors can engage in unscrupulous practices. With NetQuery, ISPs with good networks can advertise the quality of their networks in an automatic, remotely-verifiable fashion.

Transparency and accountability for claims improve market efficiency by reducing the economic transaction costs associated with establishing agreements. In particular, NetQuery allows applications to discover network properties that are otherwise difficult or impossible to determine by external data plane probing.

NetQuery reasoning is highly flexible and able to perform reasoning spanning multiple ASes. Since NetQuery's logic-based credential system supports many mechanisms for establishing trust besides a priori relationships, such reasoning can readily incorporate information from any principal. For instance, TPM-based credentials leverage trusted hardware to incorporate device-generated information and audit-based credentials incorporate network information added by trusted third parties.

2.1 Scenarios and applications

NetQuery enables a wide range of applications based on reasoning about the properties of a remote network.

Enforcing interconnection policies. Although the level of direct interconnection between ASes on the Internet has grown substantially [57], lack of trust limits the potential benefit of this dense graph. For instance, engaging in mutual backup, wherein each AS allows the other to use its transit paths in emergencies, increases overall fault tolerance. Yet unscrupulous ASes might misuse these paths for non-emergency traffic. By checking a neighbor's BGP policies and forwarding table entries, a network can verify that backup paths are only used at appropriate times. This information is not currently available to external parties.

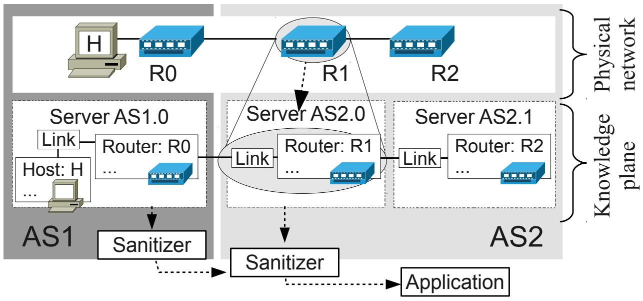


Figure 1: **NetQuery architecture.** A physical network and its knowledge plane representation, stored on knowledge plane servers operated by each AS.

Verifying AS path length. Since performance typically degrades as packets traverse multiple ASes, ISPs are motivated to establish peering or transit relationships to shorten AS path lengths. A small content provider or access ISP that seeks to reduce AS path length but lacks the resources to establish many direct interconnections might instead purchase service from a provider that has low AS path length connections to the desired destinations [22]. The purchaser benefits from outsourcing the overhead of managing many peering relationships and can use NetQuery to verify that traffic will be forwarded with minimal stretch. By comparison, establishing this arrangement today by using BGP-reported path information and traceroute would necessitate a large trusted computing base as well as incur the cost of active probing.

Advertising network redundancy. Some networks are constructed with redundant devices and network links to increase availability. A provider with a highly redundant network can use NetQuery to advertise this fact by using a reasoning process that inspects the network topology. By comparison, it is difficult or impossible to detect redundancy with probing, since the extra resources are only visible during failure.

Avoiding rogue Wi-Fi hotspots. In urban areas, mobile users are typically within range of many wireless networks [44]. Users can employ NetQuery to differentiate between these, using analysis to select networks with better security and performance. By checking for a network built from trustworthy devices and link-level encryption, a user can avoid connecting to rogue Wi-Fi hotspots [37]; by checking the capacity of the backhaul path to the upstream provider, a user can choose the best performing network in an area.

Future opportunities. Many proposals for improving service discovery, such as those for bandwidth markets [56] and virtualized routing infrastructure [36], have the potential to greatly expand the set of service providers and peers available to a given ISP. NetQuery can maximize the benefits of such proposals by providing new ways to check whether a newly discovered service provider or peer is suitable.

2.2 System overview

Network information is disseminated in NetQuery using a *knowledge plane* that maintains a representation of the network topology and configuration (Figure 1). The knowledge plane makes this information available to applications for determining whether the network exhibits desired properties. The process of inferring some high-level *characteristic* of the network (such as the loss rate on a route between hosts) from low-level properties (such as routing tables) is called *analysis*. Status information about network entities is typically self-reported by these entities (e.g., routers export their forwarding tables), though a transition mode is supported for proxies to transfer management information from legacy entities to NetQuery.

Logically, a single, global knowledge plane incorporates all properties across multiple administrative domains on the Internet. Physically, this knowledge plane is federated — each administrative domain runs a cluster of servers that locally stores all information describing its network. Federation facilitates incremental deployment and protects confidentiality, since an operator can independently run NetQuery servers without making information accessible to operators of other administrative domains.

Applications can query the knowledge plane for information about any participating network. Networks will typically restrict direct access from external parties, but instead allow *sanitizers* to execute operator-authorized sets of analyses on behalf of external applications. These analyses export only the network characteristics that meet an operator's information disclosure policies. The exported sanitizer results are accompanied by credentials certifying that the correct analysis code was executed.

Each NetQuery application independently defines the set of principals it trusts. The knowledge plane can include conflicting information from different sources; applications can filter such information based on the principals that they trust. This trust can be predicated on any credential associated with a principal. Often, such credentials are issued by a TPM, which binds statements issued by a principal to a hardware/software platform. TPMs enable NetQuery to collect a broad pool of attributed properties at low cost since they are inexpensive and enable trust establishment costs to be amortized. In particular, rather than establish trust with every potential counterparty, analyses need only establish trust with platforms, with this cost amortized across all participants deploying those platforms. This enables TPM-bearing devices to automatically issue unforgeable, fine-grained certificates for information inserted into the knowledge plane.

Often, there are multiple ways to satisfied a desired network characteristic. For instance, the fault resilience of a network may be derived through an analysis of its topology or through independent vetting by an auditor. NetQuery employs a logic for sound, flexible derivation of characteristics. Logical reasoning yields a proof that is self-documenting in that it describes all the assumptions and inference steps used to conclude that a characteristic holds. Such proofs are useful in logging and auditing.

3. DATA MODEL

A key challenge in building a knowledge plane for the Internet is to contend with its many trust domains. A knowledge plane that spans multiple organizations must support access control policies to protect confidential properties. Because the knowledge plane might contain inaccurate information, applications should be able to express policies that specify what information is safe to use for analysis. Since the system includes a diverse range of devices and implementations, standardization of nomenclature is necessary for interoperability.

The knowledge plane in our NetQuery prototype is based on a *tuplespace* representation. Every *tuple* is named by a globally unique *tuple ID* (TID) and stores properties as typed attribute/value pairs; an attribute/value pair and its associated metadata is called a *factoid*. NetQuery supports string, integer, references to tuples, dictionary, and vector values for factoids.[1]

NetQuery *principals* are the basis for policy decisions. Every producer and consumer of information from NetQuery is represented by a principal, which has a unique public/private key pair. The key pair is generated independently. NetQuery records two pieces of policy-associated metadata for every factoid it stores: *attribution*, the principal responsible for generating the factoid; and *export policy*, which defines what principals can read the factoid.

In a federated environment it is impractical to expect global consistency or uniform interpretation of properties contributed by diverse sources. But by retrieving attributions, applications have some basis to reason about whether a property is suitable for use based on whether the provider of that property is trustworthy.

To facilitate interoperability between devices and analyses, NetQuery information conforms to voluntary *schemas*. Each schema defines the set of properties that a given kind of network element must provide. NetQuery schemas prescribe a data format but do not prescribe associated code or operations. NetQuery provides standard schemas for representing devices (e.g., hosts, routers, and switches) and network abstractions (e.g., TCP connections, TCP/UDP endpoints, and IPsec security associations). NetQuery schemas are similar to those of network management systems (e.g., SNMP) that are supported by many devices. By adding to such devices a shim for outputting properties or by interfacing through an SNMP proxy, we enable them to participate in NetQuery.

Initializing factoids. Tuples and factoids for a network device can be initialized and maintained by different network participants, including the device itself, its administrator, or a third party. Since routers and switches have limited processing capacity, they offload their tuples to *tuplespace servers*, thereby insulating against application-induced query load.

On start up, a device discovers its local tuplespace server from a DHCP option and then transfers local configuration and initial state to that server by issuing **create**() operations to the tuplespace server to instantiate tuples and **update**() operations to write factoids. A newly activated router, for example, creates tuples for all of its interfaces and exports its initial forwarding table state. Hosts, routers, and switches also export local topology information to the tuplespace, using a neighbor-discovery protocol such as LLDP to generate ground facts. The device pushes any changes to its configuration and state to the tuplespace server.

Tuplespace servers and lookup protocol. Each AS or third-party information service operates tuplespace servers. The TID for a tuple embeds the IP of the tuplespace server storing that tuple, along with an opaque identifier. Device references are stored as TIDs, and therefore analysis can efficiently access the relevant tuplespace servers. To prevent changes to facts and metadata while in flight through man-in-the-middle attacks, tuplespace servers communicate over secure channels.

To prevent DoS attacks by applications that issue costly tuplespace server operations, all remote operations always terminate in bounded time. The tuplespace server only supports simple wildcard queries for attributes, which in the worst case loops over all attributes for a given tuple. NetQuery provides no mechanisms for invoking either

[1] A production version of NetQuery might well leverage the ongoing development of semantic web technologies such as RDF and OWL [43] for building the knowledge plane. Research into a semantic web has produced considerable infrastructure and theory for the federated knowledge stores, reasoning, and query processing that underpin any knowledge plane. NetQuery can also help ongoing efforts to extend the semantic web to cover network management information.

recursive queries or stored procedures. Clients, however, are free to aggregate data and locally perform expensive analyses.

Tuplespace servers make extensive use of soft-state to improve performance. Since the tuplespace contents derive from device state, a tuplespace server can always ask devices to re-export their state. This obviates the need for tuplespace servers to support a costly transactional recovery mechanism. The tuplespace uses lease-based storage management for factoids. Thus, once a device fails, the stale factoids will be garbage collected automatically.

3.1 Dynamics: changes and triggers

Changes in the network can invalidate properties in the knowledge plane. Time of check/time of use bugs can occur because network changes take place concurrently with analysis. Similar behavior can arise from probe-based measurements such as `traceroute`.

NetQuery provides a *trigger* interface to facilitate detection of changes to underlying properties during analysis. In addition to this, the interface allows applications that depend on long-running characteristics to be notified when some conclusion no longer holds. Once a trigger is installed, a callback packet is sent to an application-specified IP-port pair (*trigger port*) whenever a specified factoid has been modified. Triggers are stored by tuplespace servers as soft-state, so applications must periodically send keep-alives to refresh their triggers.

Our NetQuery applications check for relatively stable characteristics; here, triggers suffice to filter spurious analysis results that arise from observing transient states. Network tools built using probing interfaces typically would be fooled by such transients. To implement this filtering, applications install triggers for factoids being used. For instance, consider an application that enumerates all hosts in some given L2 Ethernet domain by issuing queries that traverse the network graph. Were the topology to change during these queries, the application might miss newly connected hosts. To guard against this, the application issues a **retrieve_and_ install_ trigger**() operation to atomically retrieve link information from a switch and thereafter to monitor it for changes. This atomic operation eliminates the window of vulnerability between the time a factoid is read to when monitoring for changes to that factoid starts.

Network delays in message delivery can cause updates from different devices to be received at a tuplespace server interleaved in unpredictable ways. A consistent cut guarantee would eliminate such inconsistent views, but the underlying network protocols and devices typically support neither consistency nor causality. We could augment devices with logical clocks, but that would be challenging to deploy incrementally, since means would be needed to approximate causal dependencies when exchanging messages with legacy devices. Fortunately, operator reasoning is typically focused on steady-state network characteristics, and causal consistency is less critical there. Future work will return to this problem.

4. ANALYSIS

NetQuery provides mechanisms that parse factoids acquired from tuplespace servers, determine whether a factoid is trustworthy, and check if a desired network characteristic is supported by trusted factoids.

Nexus Authorization Logic (NAL) provides the logical foundation for making inferences from factoids. A full discussion of NAL is beyond the scope of this paper (see [49]). Below, we simply outline the main features of NAL, describe how it is used in NetQuery, and discuss the implications of our choice. NAL admits reasoning about factoids and attribution information, enabling applications to reconcile conflicting statements uttered by different principals. The says and speaksfor operators, along with a set of two dozen inference rules, permit inferences based on an application's trust assumptions.

NAL associates a *worldview* with each principal. This worldview contains beliefs the principal holds about the network. Reasoning in NAL is local to every worldview: by default, inference takes into account only local beliefs, rather than statements believed by other principals. This local reasoning restriction prevents reasoning by one principal from being corrupted by contradictory facts attributed to another principal. Using speaksfor, a principal can incorporate into its worldview beliefs from other principals that it trusts. An optional scoping parameter restricts speaksfor to import only a subset of statements that concern a specific matter of interest.[2]

Applications use NAL to derive theorems about the network from information provided by the NetQuery knowledge plane. Specifically, factoids are converted to logical statements, which are then used to prove some given *goal statement*. A goal statement is a NAL formula that characterizes what a client application wants to establish. An application initially populates its worldview with an *import policy*, specifying what factoids the client deems to be trustworthy. Import policies often include speaksfor statements that specify which hardware and software platforms are trusted sources for factoids.

Ground statements. Factoids from tuplespace servers translate into NAL *ground statements*. Tuplespace API operations, such as fetching factoids, translate their return values to NAL formulas. For instance, a **retrieve**() operation issued on router *R0*'s tuple returns factoids as NAL axioms of the form:

$$\begin{aligned}&\mathit{TuplespaceServer}\ \mathsf{says}\\ &\qquad \mathit{R0}\ \mathsf{says}\ (\mathit{R0}.\texttt{Type} = \text{“Router”} \wedge\\ &\qquad\qquad \mathit{R0}.\texttt{fwd\_table} = \{\mathit{ip}_A => \mathit{nexthop}, \ldots\})\end{aligned}$$

where "$\mathit{ip}_A => \mathit{nexthop}$" denotes a forwarding table entry specifying the next hop for a given destination. The nesting of says formulas captures the full chain of custody during the factoid's traversal through the knowledge plane. Here, the factoid was exported from *R0* to a particular tuplespace server, *TuplespaceServer*.

The NAL *proof* for a goal statement is a derivation tree with the goal statement as the root, NAL inference rule applications as internal nodes, and ground statements and axioms as leaves. Analyses typically provide a proof generator that embodies a programmer's understanding of how to check whether a given characteristic holds into a proof generation strategy.

Proofs can be consumed entirely within a single application or exported to other parties as a self-documenting certificate. The certificate can be logged to create an audit trail for accounting, documentation, and debugging. Such audit trails are also useful in application domains governed by external compliance requirements, such as Sarbanes-Oxley and HIPAA.

4.1 Example: Checking network paths

This example shows how an application might use NAL to verify that a network complies with a performance requirement. Suppose site *A* wants to establish that the path to site *B* provides low loss rate (Figure 2).

[2] Since NAL does not encode a notion of degrees of trust, speaksfor is monolithic in that it incorporates all in-scope statements. NetQuery can switch to a logic that supports such reasoning [15, 41] should the need arise.

The goal statement for a path P to satisfy a bound r on the loss rate is

$$\begin{aligned}&\exists P \in \mathit{Routers}^{2+}:\\ &\quad (P[1]=A)\ \wedge\ (P[|P|]=B)\ \wedge\\ &\quad (\forall i\ 1 \le i < |P| : P[i] \rightarrow \texttt{fwd\_table}\langle B\rangle = P[i+1])\ \wedge\\ &\quad (\exists r' \in \mathit{Reals} : (r' < r)\ \wedge\ \text{TrustedLossRate}(P, r'))\end{aligned} \tag{Goal}$$

We write "$\mathit{Routers}^{k+}$" for the set of all finite router-tuple sequences of length k or greater, "$\texttt{TID.name1} \rightarrow \texttt{name2} = v$" as shorthand for dereferencing a reference value factoid to access a second factoid, "$|P|$" for the length of a sequence, "$P[k]$" for the kth element in a sequence, and "$T\langle key\rangle$" for a lookup from a factoid of dictionary type. The first three lines constrain P to be a valid path given the source, destination, and forwarding table state. The last line asserts an upper bound on the expected one-way loss rate on P given information from trusted principals, and that predicate is defined as

$$\begin{aligned}&\text{TrustedLossRate}(P, r) \triangleq\\ &\quad (\exists P', P'' \in \mathit{Routers}^{1+}\ \exists p \in \mathit{Routers}\ \exists r', r'' \in \mathit{Reals}:\\ &\qquad (P = \text{Concat}(P', p, P''))\ \wedge\ (r = r' + r'')\ \wedge\\ &\qquad \text{TrustedLossRate}(\text{Concat}(P', p), r')\ \wedge\\ &\qquad \text{TrustedLossRate}(\text{Concat}(p, P''), r''))\ \ \vee\end{aligned} \tag{Cjoin}$$

$$\begin{aligned}&(\exists R, R_{next} \in \mathit{Routers}:\\ &\quad (P = \text{Concat}(R, R_{next}))\ \wedge\\ &\quad (R\ \textsf{says}\ R.\texttt{curr\_loss\_rate\_to}\langle R_{next}\rangle = r)\ \wedge\\ &\quad \text{IsTrustedRouter}(R))\ \ \vee\end{aligned} \tag{C0}$$

$$\begin{aligned}&(\exists I \in \mathit{ISPs}:\\ &\quad (\forall i\ 1 \le i \le |P| : P[i].\texttt{ISP} = I)\ \wedge\\ &\quad (I\ \textsf{says}\ I.\texttt{sla\_loss\_rate}\langle P[1], P[|P|]\rangle = r)\ \wedge\\ &\quad \text{IsTrustedISP}(I))\ \ \vee\end{aligned} \tag{C1}$$

$$\begin{aligned}&(\exists A \in \mathit{Auditors}:\\ &\quad (A\ \textsf{says}\ A.\texttt{measured\_loss\_rate}\langle P[1], P[|P|]\rangle = r)\ \wedge\\ &\quad \text{IsTrustedAuditor}(A))\end{aligned} \tag{C2}$$

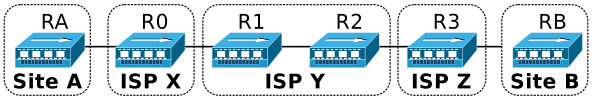
where "Concat()" denotes sequence concatenation. IsTrustedRouter(), IsTrustedISP(), and IsTrustedAuditor() are predicates derived from import policies, where the policy for IsTrustedRouter() checks the attestation and platform while the latter two check whether I and A are on a whitelist of trusted principals.

This analysis incorporates facts from multiple trustworthy sources, each having different ways to determine the loss rate of a path: (C0) expresses trust in certain routers to report their instantaneous link statistics, (C1) expresses trust in certain ISPs to claim SLAs, and (C2) expresses trust in certain auditors and measurement tools [40] to report measured performance. Each rule infers loss rate using different data schemas.

4.2 Using TPM attestations

Every TPM is uniquely identified by a set of keys that is certified by a PKI being operated by the device manufacturer. Attestations are remotely-verifiable, unforgeable, and tamper-proof certificates that use these device keys to bind a software-generated bit string (typically, a message) to the particular hardware and software platform that generated it [25]. By linking attestations to attribution metadata, NetQuery allows applications to unequivocally link each factoid back to some particular device.

Attestation is not a panacea against all misbehavior. Attestation merely establishes accountability; a NetQuery client using a factoid has to decide whether to trust the platform that is attesting to that factoid. For instance, suppose a routing platform is designed to honestly reports its observations of the control- and data-plane as factoids. It is tempting to assume that attestation of this platform implies that the factoids agree with the real-world. But malicious operators can manipulate these observations and trick poorly-constructed devices into issuing factoids that disagree. Section 5.2.2 discusses such attacks.



< RA.*fwd_table* = { RB => R0, ... } >
< R0.*curr_loss_rate_to<R1>* = 0.1%,
R0.*fwd_table* = { RB => R1, ... } >
< R1.*ISP* = ISP-Y,
R1.*fwd_table* = { RB => R2, ... } >
< ISP-Y.*sla_loss_rate<R1,R3>* = 0.1% >

< R2.*ISP* = ISP-Y,
R2.*fwd_table* = { RB => R3, ... } >
< R3.*ISP* = ISP-Z,
R3.*fwd_table* = { RB => B, ... } >
< A.*measured_loss_rate<R3,RB>* = 0.1%>

Figure 2: **Topology and tuplespace contents for network analyzer example.** Attribution metadata ("says" information) has been omitted for brevity.

TPM optimizations

NetQuery applications make their own determination about how factoids are interpreted based on attribution information. In the baseline system, every factoid and credential is signed, allowing clients to independently check that factoids are attributed to the right principals and that credentials are issued by the right parties. However, this checking is costly if clients use factoids and credentials from many different principals. Hence, NetQuery incorporates two optimizations.

Avoid TPM signatures. Whenever possible, NetQuery avoids obtaining signatures from the TPM, which is much slower than the CPU. Instead, NetQuery constructs software principal keys using a TPM-rooted certificate chain that the CPU uses for signing every factoid update. Thus, TPM attestations are used only once per boot, to bind the software principal to the platform.

Attest to tuplespace servers. Signatures provide end-to-end integrity and authentication for factoids. This helps when tuplespace servers might manipulate factoids, but it is unnecessary for tuplespace servers that are trusted to relay factoids correctly. By distinguishing such servers with attestation, NetQuery can replace per-factoid signatures with secure channels built from symmetric keys. Clients using this optimization specify an import policy that accepts tuples without signature-verification from attested tuplespace servers. These policies leverage the says information within ground statements, which encodes the tuplespace server's position as a repository of utterances from other principals.

4.3 Confidentiality and sanitizers

Agreements between network participants often stipulate the presence of certain network features. For instance, peering agreements mandate up-time and fault tolerance guarantees [3], SLAs mandate desired latency and loss rate characteristics, and service agreements for cloud datacenters reference the network bisection bandwidth and oversubscription levels. Verifying the accuracy of such advertised claims is at best difficult and often impossible. Trust establishment is typically performed manually, pairwise for each agreement, using ad hoc means.

In contrast, knowledge plane analysis can verify such claims in a principled fashion. However, most ASes have strict disclosure policies about internal network information. A naïve use of NetQuery, where external parties run analysis to verify properties of interest, can reveal detailed internal information. To be practical, a

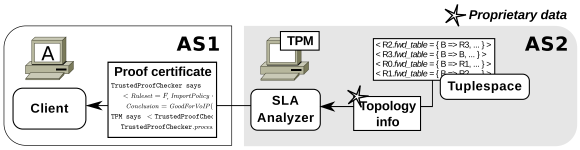

Proprietary data
AS1
A
Proof certificate
TrustedProofChecker says
< Ruleset = F, ImportPolicy
Conclusion = GoodForVoIP(
TPM says < TrustedProofChec
TrustedProofChecker.proces
Client
AS2
TPM
< R2.fwd_table = { B => R3, ... } >
< R3.fwd_table = { B => B, ... } >
< R0.fwd_table = { B => R1, ... } >
< R1.fwd_table = { B => R2, ... } >
Tuplespace
SLA Analyzer
Topology info

Figure 3: **Preserving confidentiality.** Sanitizers support external clients without leaking information, since the exported certificates serve as trusted proxies for the full proof.

knowledge plane needs some way to provide assurances to external parties without revealing confidential data.

NetQuery provides *sanitizers* for this purpose. A sanitizer is a service that converts secret factoids into unforgeable summary factoids suitable for release to other parties (Figure 3). Sanitizers execute analysis code on behalf of the remote party. To provide assurance that a remote analysis is done correctly, a sanitizer executes in a trusted execution environment and provides an attestation certificate that ties the output factoid to the sanitizer binary and the execution time. This approach does require both parties to agree on the sanitizer at the time of contract establishment, but once that agreement is in place, a sanitizer that checks the contract stipulations reveals no more information than what is revealed in the contract itself.

NetQuery provides confidentiality guarantees through its careful use of the trusted execution platform. A sanitizer executes on an AS's own computers, so factoids that it processes never leave the AS's custody. The trusted execution environment is used solely to provide an execution-integrity guarantee to a remote party. The data confidentiality guarantees then come from the sanitization embedded in the analysis itself—not from the underlying operating system. Using sanitizers means that NetQuery does *not* require an execution environment that can provide confidentiality guarantees against attackers with physical access to the execution hardware—such systems are difficult to build, even with TPMs. Further, since ASes have complete control over when, where and how often sanitizers execute and how much data they reveal to external parties, an AS can prevent outside applications from crafting query streams that consume excessive system resources or that induce a sanitizer to leak information.

4.4 Confidentiality-preserving applications

The following examples show how NetQuery supports different applications while preserving the confidentiality of factoids. These applications, along with the ones from Section 2.1, can all be implemented with sanitizers.

Verifying performance and reliability guarantees. Configuration generators that use global network optimization to achieve performance and reliability goals are increasingly prevalent [50, 4]. These tools automatically configure a network based on workload, topology, and performance constraints. Operators that rely on configuration generators can use NetQuery to advertise the achieved goals.

Configuration generators are typically complex and proprietary to an operator; so they are not disclosable to network clients as a means of certifying performance goals. But a sanitizer could run an industry standard configuration checker (such as [16]) to verify that the output configuration from a configuration generator meets the performance claim. Moreover, a network operator can upgrade to a new configuration generator without updating the contract or disclosing the new code.

As an example of this construction, suppose an operator offering MPLS-based VPNs advertises guaranteed bandwidth in the presence of a single node or link failure [34]. The operator could run a global optimizer to find an assignment of MPLS primary and backup paths that satisfies all reservations and link capacity constraints. A configuration checker can then validate this assignment by walking through the MPLS-related factoids: for each failure scenario, the checker would verify that backup paths do not overload any links.

Dynamic verification of contractual obligations. Some contractual obligations are easier to verify dynamically than statically. For instance, precomputed backup paths in MPLS VPNs provide bandwidth guarantees only for the first failure. To establish resilience against additional failures, the operator needs to compute a new set of primary and backup paths after each failure. A configuration checker can detect this by using triggers and updating its output factoids accordingly.

But updating the output factoid every time the provider-topology changes leaks information about outage and maintenance intervals. To prevent this disclosure, the operator can interpose another sanitizer certifying that the network successfully recovers from failures within a reasonable time. As long as this assertion is met, the sanitizer leaks no information beyond that found in the customer contract.

5. USING TRUSTWORTHY COMPUTING ABSTRACTIONS

Operators are not only well-positioned to launch attacks on NetQuery but they have incentives to do so. In this section, we outline the NetQuery security model and discuss the applicable results from trustworthy computing research in building routers that can provide assurances. We also discuss the vulnerabilities that can exist in analyses that improperly interpret knowledge plane information, along with defenses to protect against such concerns.

5.1 Security model assumptions

NetQuery depends on the following security assumptions:

- **Hardware and software security.** Attackers cannot tamper with the execution semantics of a device. The only way to affect running code is to use the explicit interfaces provided by the device (e.g., I/O ports, RPCs, configuration console) rather than side channels (e.g., installing hardware probes on its memory bus). Moreover, attackers cannot extract secrets stored in secure coprocessors.
- **Cryptographic algorithm security.** This assumption, shared by most work on security, implies that digital signatures used for attestation cannot be forged. It also implies the confidentiality and integrity of messages conveyed by secure channels.

Together, these assumptions imply that TPM attestations are unforgeable, since execution, encryption, and credentials cannot be compromised or spoofed. Consequently, messages used to implement the knowledge plane can be bound to the hardware/software platform responsible for generating them.

TPM-equipped commodity PC hardware approximates our hardware security assumptions. Network devices are substantially similar to PC hardware — the primary difference is an additional high-performance switching backplane not found on most PCs, but this backplane is logically equivalent to sophisticated I/O devices, which can be attested to [31].

Technology trends suggest that future trustworthy computing platforms will be even better approximations of our hardware resilience assumptions. This suggests that device manufacturers have incentives to further improve commodity platforms. It is already possible to build highly tamper-resistant platforms, ranging from high-performance encryption of buses to protect against probing attacks [53, 32] to highly secure processors for protecting major financial and PKI transactions [55]. Thus, even today, designers of trustworthy routers can choose a level of hardware security that can support the needs of NetQuery. Even without such improvements, TPMs are becoming more deeply integrated into platforms, raising the bar for physical attacks. Even if a TPM is compromised, the damage is localized, since the extracted keys can only be used to generate information attributed to that TPM.

5.2 Incomplete and inaccurate world views

The knowledge plane encodes an incomplete view of the data plane, control plane, and broader operating environment. Analyses must cope with the potential incompleteness to avoid deriving unsound analysis results. Inaccurate information may also arise due to poor device implementations.

Below, we present examples of incompleteness and inaccuracy, along with ways to ensure that analysis and devices are implemented correctly.

5.2.1 Exogenous information

Knowledge plane incompleteness can arise because relevant network information is outside the purview of NetQuery devices. For instance, consider a pair of routers connected with multiple links. Though they appear independent to the devices, these links could be physically separated or they could reside in the same undersea cable bundle and subject to correlated failures. Similarly, distinct devices might exhibit common mode failure due to physical breaches or financial insolvency of the operator.

A fault-tolerance analysis that equates multiple links, as reported by the device, with independence would derive unsound conclusions. Only by including more information in the knowledge plane, such as the physical location of fibers as compiled in databases [52], can one hope to avoid such errors. The NetQuery knowledge plane can incorporate such data.

5.2.2 Inaccurate information

Consider a network that has routers where the forwarding and control layers behave as today, but run NetQuery and satisfy its security assumptions. This strawman router design provides weak assurances; we will show how to improve on this.

Router implementations are complex; a malicious operator could well gain control of the control plane, data plane, or NetQuery processes, then induce them to export inaccurate facts. Trustworthy computing platforms provide isolated monitoring mechanisms [51] that operate independently of application code; using these to monitor the control plane and data plane and to export the inferred properties to the knowledge plane results in improved robustness.

A malicious operator could introduce non-NetQuery nodes into the real network such that the nodes are not reported to the knowledge plane, yet substantially change the behavior of the network. Any actions taken based on analysis of this knowledge plane could be misguided and violate the intended policy. This attack is inexpensive to launch, since the introduced nodes can be implemented with commodity devices.

To illustrate these attacks, suppose a dishonest ISP installs additional nodes to trick NetQuery analyses into inferring a high quality network even while the actual physical network supports only a fraction of the claimed capacity, provides no redundancy, and sends customer traffic over indirect routes.

In existing networks, operators have access to the keys used to secure control protocols such as OSPF and BGP. Malicious operators can use these keys to spoof routing advertisements, in a *forged control message attack*. Further, a malicious operator can also use network virtualization to hide a slower physical network or tunneling to redirect traffic along alternate paths. Such attacks embody forms of *data plane/control plane dissociation*.

In each of these cases, the knowledge plane reports that the routers are directly connected at the control or data link layer, yet they are in fact connected to an operator-controlled node. The solution is to add countermeasures to ensure that the knowledge plane and real network match. To protect against the forged control message attack, NetQuery encrypts all control messages between NetQuery-equipped devices with per-session keys known only to the devices. To protect against the dissociation attack, NetQuery devices can adopt standard solutions for monitoring the data plane for anomalies, such as trajectory sampling [20] to probabilistically detect packet redirection, link-layer encryption [33] to ensure that no control or data packets at all are redirected, and performance monitoring [7] to establish capacity bounds on every link.

5.3 Confidentiality-preserving analysis architectures

One of the design principles of NetQuery was to leverage existing trust relationships rather than requiring new ones to be forged. This principle comes into play if we consider the problem of protecting the confidentiality of operator information, which is a central concern of operators.

5.3.1 Sanitizers

NetQuery sanitizers execute in a machine controlled by the network operator, with the assurance that the right analysis was executed, as discharged by attesting to the code that ran on the machine. This leverages a pre-existing trust relation: in the absence of NetQuery, the external party has to trust the network operator to issue a properly-derived result. Hence, a NetQuery sanitizer does not change the extant trust relationship. Rather, it simply puts the original trust assumption on a mechanically-backed basis.

The alternative would be to execute the sanitizer in a TPM-equipped machine controlled by the external party. Here, the network operator would have to ship confidential information to external machines and would have to trust those machines to not reveal this information. The trust relation required here is substantially different from the original; operators might resist deploying this type of sanitizer. Their concern is well-founded: this architecture provides less defense-in-depth against the compromise of confidential information and there are indeed low-cost attacks that can extract confidential information from the memory of TPM-equipped machines [30].

5.3.2 Analyses spanning multiple domains

In preceding applications, we showed how to support multi-domain analyses that are decomposable into independent, per-domain sanitizers. We offer here a design for another approach that admits analyses that span multiple domains while preserving confidentiality. Such analyses are useful for implementing traffic engineering across multiple ASes while preserving confidentiality [39].

Our solution is to combine NetQuery with secure multiparty computation (MPC). In fact, NetQuery's strengths complement the weaknesses of MPC. Adding NetQuery helps to satisfy the trust assumptions of MPC. Since MPC protocols do not constrain participants to be honest about their inputs, each MPC application typically requires an application-specific security analysis of the implications of such cheating. Some MPC protocols leak information to participants that do not follow the protocol, which poses a deployment risk in realistic scenarios, but NetQuery obviates such concerns by protecting the MPC inputs and protocol implementation. This approach

Libraries	
Server & client	18,286
NAL	2,254

Data sources	
Nexus host	543
Software switch	1,853
Quagga router	+777
SNMP proxy	1,578

Applications	
Network access control	787
L2/L3 traceroute	483
Oversubscription	356
Maximum capacity	316
Redundancy	333

Figure 4: **Source line count for NetQuery libraries, devices, and analyses.** Router figure is the code size increase relative to Quagga. SNMP proxy supports HP and Cisco devices and exports a superset of the data for switch and router.

improves assurance by attesting to the execution of a trustworthy MPC implementation and restricting the MPC inputs to factoids from trusted devices.

5.4 TPM deployment

NetQuery's flexible import policies enables applications to benefit whether or not devices are equipped with TPMs.

Deployments without TPMs. For example, TPMs are not necessary in scenarios where pre-existing relationships are considered sufficient for trusting the claims of another party. Operators currently invest significant effort in establishing trust before entering peering agreements and may trust each other enough to exchange information through network diagnostic tools. This trust relationship can be represented by import policies that accept the remote ISP as a root of trust in lieu of a TPM key. In NetQuery, we have the remote ISP sign X.509 certificates with a self-generated key, and specify a corresponding import policy. Although this TPM-less configuration does not help with trust establishment, it still streamlines coordination and can be used to generate an audit log documenting why a claim was accepted.

Legacy network elements will lack a TPM with which to generate factoids. Here, an ISP can use its own self-signed key to issue statements on behalf of such legacy devices. Modern ISPs run extensive management software that collects information on the state of the network and devices within, and exporting the data from such systems into NetQuery enables even ISPs with pure legacy devices to support NetQuery applications. This approach supports a transparent transition as new TPM-equipped network devices are introduced, and ISP-signed statements are subsumed by device-issued factoids.

Finally, NetQuery sanitizers can enable external applications to participate in NetQuery even when they are not equipped with TPMs. Because sensitive factoids never leave an administrative domain, lack of a remote TPM can never lead to information leakage.

External checkers. Some trustworthy properties can be obtained by using TPM-equipped devices to infer or monitor legacy devices. Past work has examined how to obtain guarantees about the behavior of legacy BGP speakers by monitoring their inputs and outputs [47, 29]. Since monitors only need to inspect control traffic to infer details of BGP behavior, low cost monitoring hardware suffices to provide assurance about the behavior of expensive legacy equipment, such as high-performance routers.

6. INCREMENTAL DEPLOYMENT

NetQuery analyses can verify common advertised claims even when only some devices are upgraded to support NetQuery.

Bilateral benefits. Many claims in bilateral contracts can be computed almost entirely from factoids exported by one of the counterparties, and thus require minimal support for NetQuery outside of the participating networks. Since most Internet agreements are bilateral [22], this is a common case.

	Completion time (seconds)	Network cost (sent/recv'd)
L2/L3 traceroute	0.16 s	247 KB
Oversubscription	(pre-processing) 7.9 s (per switch) 0.1 s	17 MB 0 KB
Maximum capacity	0.16 s	247 KB
Redundancy	12.65 s	24 MB

Figure 5: **Performance of analyses on department network**. The execution time and network cost of each analysis suffices to support network management and data center SLA queries. Oversubscription analysis pre-processes the tuplespace to reduce query costs.

Some claims are completely self-contained within an ISP's network; these include those providing VPN service between multiple customer sites or guaranteeing an intra-domain latency or redundancy SLA. Other analyses, such as the Wi-Fi hotspot and AS hop count analyzers, require a modicum of support from ASes adjacent to the Wi-Fi or transit provider. These analyses verify that the provider routes traffic to a specific destination domain as promised: for the former, to the public Internet (e.g., outside the hotspot's domain), for the latter, to the destination AS. The adjacent networks need only install NetQuery devices at the edge and assert their ownership of these device, say by signing a factoid using a AS-number certificate issued by a regional Internet registry [11]. Adjacent networks need only ensure that they deliver packets into their network as claimed by the knowledge plane. To do so, they can simply deploy a low-cost TPM-equipped host, rather than upgrade edge routers.

NetQuery islands. In other scenarios, there may be islands of NetQuery-enabled devices separated by multiple legacy devices. Islands might arise within a single provider that deploys starting at the edge of each POP or in a few POPs at a time. Islands may also be isolated by legacy devices controlled by a third party. By establishing tunnels between one another, backed by encryption and performance monitoring [7], NetQuery devices can export properties describing the intervening legacy network. Such tunnels are similar to the defenses against the dissociation attack and the preceding deployment optimization for adjacent ASes.

7. PROTOTYPE AND FEASIBILITY STUDY

We have implemented a prototype of the NetQuery system described above. The core functionality is supported by an embeddable tuplespace server for building NetQuery devices and sanitizers; a C++ client library for writing NetQuery applications; NAL proof generators and checkers; and a stand-alone tuplespace server. Using these components, we have built a NetQuery switch for Linux, a NetQuery router adapted from the open source Quagga router [1], a NetQuery host that runs the Nexus trusted operating system, and an SNMP to NetQuery proxy (Figure 4). We also built a network access control (NAC) system and several network performance analyzers.

We describe, through our experience with building applications, the benefits of NetQuery-enabled analysis. We also demonstrate, through microbenchmarks of tuplespace operations and experiments and devices, that NetQuery achieves high throughput and low latency and that extending network devices to support NetQuery involves little code modification, low overhead, and low deployment cost.

All experiments used a testbed built from Linux 2.6.23 hosts equipped with 8-core 2.5GHz Intel Xeon processors and connected over a Gigabit Ethernet switch. Unless otherwise stated, all TPM and NAL optimizations from Section 4.2 were enabled.

7.1 Applications and production deployment

Here, we outline the implementation of each NetQuery application and describe the achieved performance and operational benefits.

7.1.1 Network access control

The NAC system restricts network access only to machines that are unlikely to harm the network, such as those running a firewall and virus checkers. Such policies are widely embraced today, yet often rely solely on user cooperation. This system installs triggers on local NetQuery switches to detect new hosts, which are initially allowed only limited network access. In response to a trigger notification, the application analyzes the new host. For each policy-compliant new hosts, the application sends a configuration command to the switch to grant network access. Such policy decisions are implemented in the switch enforcer process, consisting of 787 lines of code (LOC). The NetQuery switch consists of 1,853 LOC, including a full control plane and software data plane with link-level encryption to prevent dissociation attacks. The host runs the Nexus operating system, which exports its full process list to a locally running tuplespace server.

To prevent leakage of sensitive user information beyond the user's computer, NAC uses a sanitizer that releases the sanitized fact CompliantHost(H) if H's process list indicates that it is running the required software. NAC uses Nexus's process level attestation to verify execution of the sanitizer and tuplespace server, which run in separate processes. The proof tree consisted of eight ground statements: five tuplespace values, authenticated by tuplespace server MACs, and three attestations, authenticated by digital signatures.

Together, the proof generation and proof checking processes took less than 67 seconds of wall clock time, which is low compared to the long duration of typical Ethernet sessions. Digital signature verification at the enforcer dominated the cost.

7.1.2 Analysis of a production network

We deployed NetQuery on our department's production network consisting of 73 HP and Cisco L2 and L3 switches and over 700 end hosts. Using standard network management data exported by these switches, we built several analyses for detecting properties of interest to customers of cloud providers and ISPs; Figures 4 and 5 summarize the source code size and performance of each analysis. NetQuery enables these analyses to discover information that is otherwise difficult or impossible to obtain through the data plane. Our deployment relies on an SNMP-to-NetQuery proxy that periodically exports the neighbor, forwarding, routing, and ARP tables of every switch.

We implemented the following analyses, which can be used to generate remotely verifiable advertisements of datacenter network quality. These advertisements enable customer applications to pick the most appropriate network for a given workload.

L2/L3 traceroute analysis. Traceroute is widely used for diagnostics. Since standard IP traceroute returns only L3 information, it provides little information in a network composed primarily of L2 switches. We have built a NetQuery traceroute that iteratively traverses the topology graph contained in the knowledge plane, instead of using probe packets. At each switch, the analyzer performs forwarding table and ARP table lookups as appropriate to determine the next hop. To support traceroute on our network, the analysis understands many commonly used features of L2/L3 switched Ethernet networks, including link aggregation groups and VLANs. This analysis is often used as the basis for other analyses.

Over-subscription analysis computes internal network capacity and determines the ratio between the capacity of a given network link and the maximum amount of traffic that hosts downstream from the link can generate or consume. To compute the aggregate capacity across all hosts, the analysis traces through the L3 core and L2 tree, down to the leaf switches, recording every host access link. Customers with network-intensive workloads such as MapReduce can benefit from choosing datacenters that are less oversubscribed.

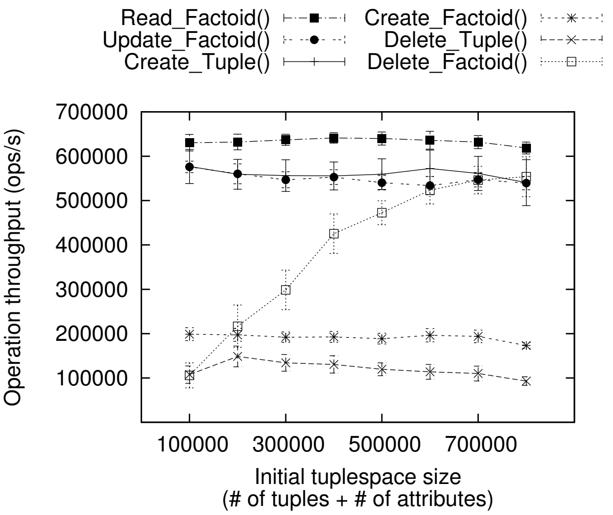


Figure 6: **Throughput of operations issued in bulk.** The throughput of all operations is independent of the tuplespace size.

Maximum capacity analysis determines the available bandwidth through the Internet gateway. This analysis determines the best-case throughput between a given host and network egress point by running NetQuery traceroute to find the host to egress path, then computing the minimum capacity across the links on that path. Customers deploying public services benefit from choosing datacenters with high available capacity through the gateway.

Redundancy analysis verifies that the network is robust against network failures. We implemented an analysis that decomposes the network graph into biconnected components, which are by definition robust against the failure of a single switch. Customers that require high availability should place their nodes in the same component as critical services and the Internet gateway.

In addition to using these analyses to support external customers, the datacenter operator can use them to debug network problems. For instance, we have used these tools to help us inventory and locate network equipment and to determine the network failure modes for our research group's externally-accessible servers.

7.2 Scalability

We evaluate the performance of a tuplespace server with throughput and latency microbenchmarks that correspond to different usages and scenarios. High throughput enables devices to initialize the knowledge plane quickly when they boot or are reconfigured. Reduced latency enables analysis to complete more quickly and limits exposure to concurrent changes to the network.

Throughput. In this experiment, a traffic generator issues a sequence of requests, without waiting for responses from a tuplespace server running on a separate machine. To fully utilize processor cores available on the server, the tuplespace is distributed across eight processes.

The results show that NetQuery can support large tuplestores and high tuplespace access and modification rates (Figure 6); a single tuplespace server can support more than 500,000 read and update operations per second. The throughput of all tuplespace operations

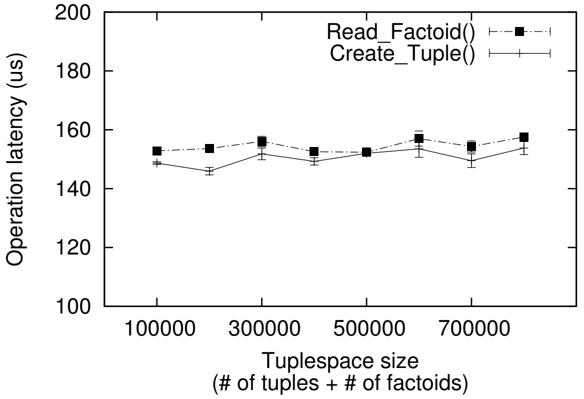


Figure 7: **Latency of operations issued sequentially.** The latency of all operations remains the same at all tuplespace sizes.

SNMPv3	NetQuery (Unoptimized)		NetQuery (Optimized)	
Bulkwalk	Update	Read	Update	Read
46,572 ±300	29 ±0.06	2,723 ±30	97,422 ±2,000	94,844 ±8,000
Throughput in SNMP variables/s or NetQuery factoids/s (± 95% conf. interval)				

Figure 8: **Comparison of SNMP and NetQuery.** TPM- and NAL-based optimizations enable NetQuery to achieve better throughput than SNMP while providing stronger guarantees.

is decoupled from the size of the tuplespace, with the exception of **Delete_Factoid**(). For smaller tuplespace sizes, **Delete_Factoid**() experiments complete so quickly that initialization overheads dominate overall execution time. The tuplespace server achieves this high performance because it only holds soft-state, for which a simple in-memory implementation suffices.

Latency. The latency of accessing the tuplespace (Figure 7) affects the completion time of NetQuery applications. In this experiment, a single client issues sequential requests for the TID from **Create_Tuple**(); and for the factoid value from **Read_Factoid**(). Latency remains constant as tuplespace size increases.

Cryptographic optimizations and SNMP. To evaluate the overhead of accountability, we compared the performance of NetQuery to that of SNMPv3. To evaluate the benefits of the optimizations from Section 4.2, we measured the throughput of NetQuery with and without trusting the tuplespace server. Since the Linux SNMP server is not multithreaded, we ran both NetQuery and SNMP on a single core. We used multiple concurrent instances of `snmpbulkwalk` to retrieve SNMP server MIBs. The NetQuery experiments retrieved comparable amounts of data. SNMP and NetQuery were both configured to provide confidentiality and integrity.

When the tuplespace server is not trusted, devices sign all factoids on export and applications check signatures on all factoids on import, significantly increasing CPU overhead. When the tuplespace server is trusted, NetQuery provides better performance than SNMP (Figure 8). Thus, we expect that SNMP analyses that are ported to NetQuery will perform comparably well, yet provide accountability.

Since TPMs and trustworthy computing can be used to establish the trustworthiness of tuplespace servers, these results show that high performance knowledge planes that support accountability can be deployed at little additional cost.

7.3 Building NetQuery devices

We built several NetQuery devices to determine the implementation and runtime costs that NetQuery adds to devices. We also show that the knowledge plane and devices can efficiently support realistic workloads.

Quagga router. We modified a Quagga router to evaluate the cost of extending an existing device to support NetQuery. Only localized changes to the router's control plane were necessary to export all interface and routing table changes to the knowledge plane. NetQuery-Quagga interposes on Quagga's calls to `rtnetlink`, the low-level interface to the in-kernel dataplane, and translates all relevant requests, such as changes to the forwarding table and NIC state, into tuple updates. In total, only 777 lines of localized changes were needed, out of a total code base of 190,538 LOC.

Initialization. Quagga is a demanding macrobenchmark that exports significant amounts of state during operation. To demonstrate that routers can efficiently shed this data to a tuplespace server, we measured the initialization and steady state performance of the router. We used a workload derived from a RouteViews trace. The Quagga router, tuplespace server, and workload generator ran on separate machines. To demonstrate that NetQuery can efficiently import bulk data, we measured the completion time to load a full BGP routing table (268K prefixes) and the resulting tuplespace memory footprint.

Upon receiving routing updates, the BGP router downloads a full forwarding table to the IP forwarding layer. Added latency could affect network availability. Without NetQuery, an update of the full table took 5.70 s; with NetQuery, it took 13.5 s. Our prototype blocks all updates while waiting for NetQuery; a production implementation can eliminate this dependency by updating the knowledge plane in a background process.

Exporting the full forwarding table to the tuplespace server required 62.8 MB of network transfer and 10.7 MB of server memory to store the table. Though full updates are the most intense knowledge plane update workload, only a modest amount of hardware is needed to support them.

Steady state. To demonstrate that NetQuery routers perform well in steady state, we evaluated the router against a workload derived from RouteViews update traces. The workload generator batched updates into one second buckets, and submitted them as bursts. The experiment recorded the time needed to commit the resulting changes to the IP forwarding tables and to the knowledge plane. NetQuery increased the median completion time to 63.4 ms, from 62.2 ms in the baseline server. Thus, the NetQuery router reacts almost as quickly as a standard router, minimizing the disruption to forwarding table updates. NetQuery required only 3 KB, 92 KB, and 480 KB to transmit the median, mean, and maximum update sizes; thus, any server configuration provisioned to support initialization load can also support steady state load.

Convergence time. NetQuery does not impact eBGP convergence time: in eBGP, route propagation is governed by a thirty second Minimum Route Advertisement Interval, which exceeds the latency of exporting a full forwarding table update to NetQuery.

To measure the impact on IGP route convergence, we simulated the update traffic from large correlated link failures on the Sprint RocketFuel topology. This topology consists of 17,163 Sprint and customer edge routers. We converted the simulation trace into a POP-level NetQuery workload, which we fed to a single-core tuplespace server.

We measured for each run the convergence time after failure. For the five largest POPs, consisting of 51 to 66 routers, and link failure rates of up to 0.05, the mean and median increase in update completion times were less than 0.24 s and 0.14 s, respectively.

Thus, networks can deploy NetQuery while achieving sub-second IGP convergence time, which is the desired level of performance for operators [24].

8. RELATED WORK

Perhaps the work closest to NetQuery are network exception handlers (NEH) [35], ident++ [42], Maestro [12], NOX [27], and DECOR [14]. All are logically centralized systems for enterprise networks that disseminate network information to administrative applications. NetQuery supports such network management applications for enterprise networks, while also supporting applications that issue queries spanning multiple ASes. Moreover, the the NetQuery knowledge plane goes one step further and supports heterogeneous information sources by tracking the source of every statement and by leveraging trusted hardware.

Declarative programming [58] has been proposed as an alternative for building applications and for managing networks. In such systems, application logic is written as high-level rules that manipulate a database representation of the network. DECOR [14] uses declarative programming to autoconfigure network devices to meet high level operational goals. DECOR uses logical specifications for device semantics and state that can be helpful in writing NetQuery analyses and sanitizers. Both NetQuery and DECOR provide frameworks for extending existing devices to support policy analysis and management applications. NetQuery applications can span mutually distrusting administrative domains, for which we provide trust establishment and sanitization techniques not needed in the problem domain of DECOR.

Trusted Network Connect [54] and [48] are network access control systems with similar client authorization to NetQuery. Before an end host is allowed to join the network, these systems use attestation to verify that the end host's software and hardware configuration satisfies the access policy. Unlike NetQuery, these systems do not provide a channel that end hosts can use to discover properties of a network before deciding to connect.

The ubiquity of TPMs has inspired many systems that rely on trusted end host functionality to improve network security and performance. For instance, [19, 23] move middlebox, filtering, and monitoring functionality to end hosts, while [8, 46, 28] rely on end hosts to perform packet classification. NetQuery provides a standard interface for describing these local guarantees, enabling applications in other administrative domains to rely upon their presence.

Several extensions to IP have been proposed to provide guarantees about the sender of each packet. Accountable IP (AIP) [2], [9], and packet passports [38] provide accountability for network packets. These systems use optimizations whose safety rely on global network configuration invariants spanning multiple ASes. They can use NetQuery to verify these trust assumptions. Assayer attaches trustworthy sender information to packets as unforgeable annotations, obviating the need to reconstruct this information at middleboxes [45]. NetQuery and Assayer make similar use of the TPM.

NetReview [29] enforces fault detection for BGP behavior using a tamper-evident log. Since NetQuery supports analysis over router RIBs, it can check similar BGP policies as NetReview. NetReview relies on an AS's neighbors to achieve tamper-evidence and trustworthy detection on publicly-disclosable information. In contrast, NetQuery can bootstrap trust from TPMs where available, and it can also use sanitizers to perform trustworthy analysis on confidential network information.

Keller et al. [36] applies trustworthy computing techniques to a new operating model where service providers build wide area services using virtual router slices leased from infrastructure providers. NetQuery uses similar techniques, but targets the existing operating model and uses a logical framework for analysis.

Network Confessional [5] provides verifiable performance measurements at the granularity of network paths and peering points. Such approaches are complementary to a knowledge plane; factoids extracted through Network Confessional can be extracted and disseminated through NetQuery.

9. CONCLUSIONS

This paper makes the case for leveraging trustworthy computing abstractions in building knowledge planes that are well-suited for heterogeneous, federated networks. We described NetQuery, a federated, lightweight, and scalable knowledge plane that uses these abstractions to assure the soundness of knowledge plane reasoning in such networks. Through granular access control, accountability, and sanitization, NetQuery supports the confidentiality policies and trust relationships found in real networks. It provides a logical analysis framework that enables applications to combine local information about network entities into global guarantees. Experiments show that NetQuery performs well on real routers and can support the volume of network events found in large enterprise and ISP networks. Overall, NetQuery's extensible data model and flexible logic supports a diverse range of applications and can help ISPs differentiate their services. We believe that NetQuery's design principles and abstractions address significant obstacles to building a practical federated knowledge plane and enable novel applications based on global network reasoning.

10. REFERENCES

[1] The Quagga routing suite. Available at http://www.quagga.net/.

[2] D. G. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker. Accountable Internet Protocol. In *ACM SIGCOMM*, Aug. 2008.

[3] AOL. AOL Transit Data Network: Settlement-Free Interconnection Policy, 2006. http://www.atdn.net/settlement\%5Ffree\%5Fint.shtml.

[4] D. Applegate, A. Archer, V. Gopalakrishnan, S. Lee, and K. K. Ramakrishnan. Optimal Content Placement for a Large-scale VoD System. In *ACM CoNEXT*, 2010.

[5] K. Argyraki, P. Maniatis, and A. Singla. Verifiable Network-Performance Measurements. In *ACM CoNEXT*, 2010.

[6] AT&T. AT&T Internet Protection Service, Aug. 2009. Available at http://www.corp.att.com/abs/serviceguide/docs/ip_sg.doc.

[7] I. Avramopoulos and J. Rexford. Stealth Probing: Efficient Data-Plane Security for IP Routing. In *USENIX Annual Technical Conference*, May 2006.

[8] K.-H. Baek and S. W. Smith. Preventing Theft of Quality of Service on Open Platforms. In *Securecomm*, Sept. 2005.

[9] A. Bender, N. Spring, D. Levin, and B. Bhattacharjee. Accountability as a Service. In *SRUTI*, June 2007.

[10] K. Bode. Why Are ISPs Still Advertising Limited Services As Unlimited?, Dec. 2008. http://www.dslreports.com/shownews/Why-Are-ISPs-Still-Advertising-Limited-Services-As-Unlimited-99769.

[11] R. Bush. An Operational ISP and RIR PKI, Apr. 2006. https://www.arin.net/participate/meetings/reports/ARIN_XVII/PDF/sunday/pki-bush.pdf.

[12] Z. Cai, F. Dinu, J. Zheng, A. L. Cox, and T. S. E. Ng. The Preliminary Design and Implementation of the Maestro Network Control Platform. Tech. Report TR08-13, Rice University, Oct. 2008.

[13] M. Casado, P. Cao, A. Akella, and N. Provos. Flow-Cookies: Using Bandwidth Amplification to Defend Against DDoS Flooding Attacks. In *IEEE IWQoS*, June 2006.

[14] X. Chen, Y. Mao, Z. M. Mao, and J. V. der Merwe. DECOR:

DEClarative network management and OpeRation. *ACM SIGCOMM CCR*, 40(1):61–66, 2010.

[15] P. Cheng, R. Pankaj, C. Keser, P. A. Karger, G. M. Wagner, and A. Reninger. Fuzzy Multi-Level Security: An Experiment on Quantified Risk-Adaptive Access Control. In *IEEE Symposium on Security and Privacy*, May 2007.

[16] Cisco Systems. IP SLAs–LSP Health Monitor, June 2006. http://www.cisco.com/en/US/docs/ios/12_4t/12_4t11/ht_hmon.html.

[17] D. D. Clark, C. Partridge, J. C. Ramming, and J. T. Wroclawski. A Knowledge Plane for the Internet. In *ACM SIGCOMM*, Aug. 2003.

[18] D. Collins. Is Your Provider Truly Multi-Homed? http://www.webmasterjoint.com/webmaster-articles/web-hosting/10-multi-homed-is-your-provider-truly-multi-homed.html.

[19] C. Dixon, A. Krishnamurthy, and T. Anderson. An End to the Middle. In *HotOS*, May 2009.

[20] N. Duffield and M. Grossglauser. Trajectory sampling for direct traffic observation. *IEEE/ACM Transactions on Networking*, 9(3):280–292, June 2001.

[21] P. England, B. Lampson, J. Manferdelli, M. Peinado, and B. Willman. A Trusted Open Platform. *Computer*, 36(7):55–62, 2003.

[22] P. Faratin, D. Clark, P. Gilmore, S. Bauer, A. Berger, and W. Lehr. Complexity of Internet Interconnections: Technology, Incentives and Implications for Policy. In *Annual Telecommunications Policy Research Conference*, Sept. 2007.

[23] W. Feng and T. Schluessler. The Case for Network Witnesses. In *NPSec*, Oct. 2008.

[24] P. Francois, C. Filsfils, J. Evans, and O. Bonaventure. Achieving sub-second IGP convergence in large IP networks. *ACM SIGCOMM CCR*, 35(3):35–44, July 2005.

[25] M. Gasser, A. Goldstein, C. Kaufman, and B. Lampson. The Digital Distributed System Security Architecture. In *NIST-NCSC*, pages 305–319, 1989.

[26] S. Goldberg, A. D. Jaggard, and R. N. Wright. Rationality and Traffic Attraction: Incentives for Honest Path Announcements in BGP. In *ACM SIGCOMM*, Aug. 2008.

[27] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. NOX: Towards an Operating System for Networks. *ACM SIGCOMM CCR*, 38(3):105–110, 2008.

[28] R. Gummadi, H. Balakrishnan, P. Maniatis, and S. Ratnasamy. Not-a-Bot (NAB): Improving Service Availability in the Face of Botnet Attacks. In *ACM/USENIX NSDI*, Apr. 2009.

[29] A. Haeberlen, I. Avramopoulos, J. Rexford, and P. Druschel. NetReview: Detecting when interdomain routing goes wrong. In *USENIX NSDI*, Apr. 2009.

[30] J. Halderman, S. Schoen, N. Heninger, W. Clarkson, W. Paul, J. Calandrino, A. Feldman, J. Appelbaum, and E. Felten. Lest we remember: cold-boot attacks on encryption keys. In *USENIX Security Symposium*, 2008.

[31] J. Hendricks and L. van Doorn. Secure Bootstrap Is Not Enough: Shoring up the Trusted Computing Base. In *SIGOPS European Workshop*, Aug. 2004.

[32] IBM Corporation. IBM Extends Enhanced Data Security to Consumer Electronics Products. Press Release, Apr. 2006. http://www-03.ibm.com/press/us/en/pressrelease/19527.wss.

[33] IEEE Computer Society. IEEE Std 802.1AE-2006. Aug. 2006.

[34] G. F. Italiano, R. Rastogi, and B. Yener. Restoration Algorithms for Virtual Private Networks in the Hose Model. In *IEEE INFOCOM*, 2002.

[35] T. Karagiannis, R. Mortier, and A. Rowstron. Network Exception Handlers: Host-network Control in Enterprise Networks. In *ACM SIGCOMM*, Aug. 2008.

[36] E. Keller, R. Lee, and J. Rexford. Accountability in hosted virtual networks. In *ACM VISA*, pages 29–36, 2009.

[37] J. Kirk. ’Evil twin’ hotspots proliferate, Apr. 2007. http://www.pcworld.com/businesscenter/article/131199/evil_twin_hotspots_proliferate.html.

[38] X. Liu, X. Yang, D. Wetherall, and T. Anderson. Efficient and Secure Source Authentication with Packet Passports. In *SRUTI*, July 2006.

[39] S. Machiraju and R. H. Katz. Reconciling Cooperation with Confidentiality in Multi-Provider Distributed Systems. Tech. Report UCB/CSD-4-1345, Computer Science Division (EECS), University of California, Berkeley, CA, 2004.

[40] H. V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani. iPlane: An Information Plane for Distributed Services. In *ACM/USENIX OSDI*, Nov. 2006.

[41] D. W. Manchala. E-Commerce Trust Metrics and Models. *IEEE Internet Computing*, (March-April):36–44, 2000.

[42] J. Naous, R. Stutsman, D. Mazieres, N. McKeown, and N. Zeldovich. Delegating Network Security Through More Information. In *WREN*, Aug. 2009.

[43] J. Z. Pan. Resource Description Framework. In *Handbook on Ontologies*, pages 71–90. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[44] J. Pang, B. Greenstein, M. Kaminsky, and D. Mccoy. Improving Wireless Network Selection with Collaboration. In *MobiSys*, June 2009.

[45] B. Parno, Z. Zhou, and A. Perrig. Help Me Help You: Using Trustworthy Host-Based Information in the Network. Tech. Report CMU-CyLab-09-016, Carnegie Mellon CyLab, 2009.

[46] A. Ramachandran, K. Bhandankar, M. B. Tariq, and N. Feamster. Packets with Provenance. Tech. Report GT-CS-08-02, Georgia Institute of Technology, 2008.

[47] P. Reynolds, O. Kennedy, E. Sirer, and F. Schneider. Securing BGP using external security monitors. Tech. Report TR2006-2065, Computer Science Department, Cornell University, Ithaca, NY, USA, 2006.

[48] R. Sailer, T. Jaeger, X. Zhang, and L. van Doorn. Attestation-based policy enforcement for remote access. In *ACM CCS*, pages 308–317, 2004.

[49] F. B. Schneider, K. Walsh, and E. G. Sirer. Nexus Authorization Logic (NAL): Design Rationale and Applications. In *TOSSEC*, Sept. 2010.

[50] V. Sekar, M. K. Reiter, W. Willinger, H. Zhang, R. R. Kompella, and D. G. Andersen. CSAMP: A System for Network-Wide Flow Monitoring. In *ACM/USENIX NSDI*, pages 233–246, 2008.

[51] A. Shieh, D. Williams, E. G. Sirer, and F. B. Schneider. Nexus: A New Operating System for Trustworthy Computing. In *ACM SOSP Work-in-Progress Session*, Oct. 2005.

[52] N. So and H.-H. Huang. Building a Highly Adaptive, Resilient, and Scalable MPLS Backbone. *MPLS World Congress*, 2007.

[53] G. Suh, D. Clarke, B. Gasend, M. van Dijk, and S. Devadas. Efficient memory integrity verification and encryption for secure processors. In *IEEE MICRO*, Dec. 2003.

[54] Trusted Computing Group. *TCG Trusted Network Connect: TNC Architecture for Interoperability, Specification Version 1.3*. Trusted Computing Group, Apr. 2008.

[55] T.W. Arnold and L. P. Van Doorn. The IBM PCIXCC: A new cryptographic coprocessor for the IBM eServer. *IBM Journal of Research and Development*, 48(3/4):491–503, 2004.

[56] V. Valancius, N. Feamster, R. Johari, and V. Vazirani. MINT: A Market for INternet Transit. In *ReArch*, Dec. 2008.

[57] B. Woodcock and V. Adhikari. Survey of Characteristics of Internet Carrier Interconnection Agreements. Tech. report, Packet Clearing House, May 2011.

[58] W. Zhou, Y. Mao, B. T. Loo, and M. Abadi. Unified Declarative Platform for Secure Networked Information Systems. In *IEEE ICDE*, Apr. 2009.