

Routing with Confidence: Supporting Discretionary Routing Requirements in Policy Based Networks

Apu Kapadia*, Prasad Naldurg, Roy H. Campbell

Dept. of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL, USA
{akapadia, naldurg, rhc}@uiuc.edu

Abstract

We propose a novel policy-based secure routing framework that extends the mandatory nature of network access-control policies and allows users to exercise discretionary control on what routes they choose in a given network. In contrast to existing research that focuses mainly on restricting network access based on user credentials, we present a model that allows users to specify discretionary constraints on path characteristics and discover routes based on situational trust attributes of routers in a network. In this context, we present three levels of trust-attribute certification based on inherent, consensus based, and inferred characteristics of routers. We also define a “confidence” measure that captures the “quality of protection” of a route with regard to various dynamic trust relationships that arise from this interaction between user preferences and network policy. Based on this measure, we show how to generate paths of highest confidence efficiently by using shortest path algorithms. We show how our model generalizes the notion of Quality of Protection (QoP) for secure routing and discuss how it can be applied to anonymous and privacy-aware routing, intrusion tolerant communication, and secure resource discovery for ubiquitous computing, high performance, and peer-to-peer environments.

1 Introduction

With the advent of Policy Based Networking (PBN), network administrators now have the ability to spec-

ify, administer, and enforce an organization’s network-access and utilization policies more effectively. PBN has traditionally focused on *which* users have access to *what* resources in a network [9]. A PBN framework uses bandwidth management, traffic-flow management, firewalling, caching, and other routing protocol and network security solutions such as IPSec, VPNs, etc., to provide differentiated services to groups of users in a dedicated network.

For most part, the policies in a PBN refer to *mandatory* access control (MAC) and utilization policies that the network, as a system, applies to its users. The PBN architecture [10] organizes different network objects such as resources and services into different *object roles*, and defines a policy as a relationship between these object roles and different *user groups*. For example, traffic from certain groups of users can be treated preferentially, or access to certain network resources can be restricted to users belonging to a specific group. In addition, policies can be defined based on the attributes of the traffic itself—e.g., music file transfers or other application specific packets can be bandwidth-limited. PBN Policies are stored in a (possibly distributed) policy repository and enforced at Policy Enforcement Points (PEPs) on firewalls, routers and switches, etc., using a wide variety of mechanisms such as access control, filtering, and queue management.

The PBN framework has greatly simplified the management and administration of organizational network security policies. In this paper, we propose a novel extension to this framework that incorporates a user’s expectations and preferences, with the existing mandatory network policies, to influence the path chosen by a user’s traffic within this setting. Our motivation stems from the observation that the *discretionary* demands of users have been largely ignored in any formulation of PBN policies.

In addition to a user’s identity and group membership

*Apu Kapadia is funded by the U.S. Dept. of Energy’s High-Performance Computer Science Fellowship through Los Alamos National Laboratory, Lawrence Livermore National Laboratory, and Sandia National Laboratory.

information, our extended framework explicitly models static and dynamic trust attributes of both users and network objects and effectively captures the changing trust relationships between them as the system evolves over time. To illustrate this, consider a user who may want to avoid certain routers based on the knowledge that the routers may be compromised because they are running outdated software with known security holes. The system administrator may not have installed the latest patch, or the patch may not be available. Note that the user’s demands in this situation do not violate the mandatory system policy in any way. While a user would be dependent on the administrator in a traditional PBN, in our proposed model, a user can encode this requirement and discover a path dynamically, consisting of routers that do not have this vulnerability, and use only these routers until the vulnerability is patched.

Other examples of a user’s discretionary policies in this setting include the ability to exclude routers that belong to an administrative domain that the user does not trust, or exclude routers that are dropping an unacceptable fraction of packets, and so on. A point to note here is that the trust attributes of both the user and the network object are *dynamic*, in the sense that they may change over time. We list different types of attributes of both user groups and network objects and classify them according to whether they are inherent, consensus based, or need to be inferred by the user in some way. This extends the traditional notion of “Quality of Service” to what we refer to broadly as the “Quality of Protection” (QoP) [3, 14] of a network route. We explore the issue of trust management and describe what entities are needed to enable certification and validation of dynamic trust attributes.

In order to capture the effect of dynamically changing trust values and relationships on the quality of routes our model can discover, we introduce a quantitative measure called *confidence*. Using this metric, we describe different functions to combine meaningfully the confidence values of individual links along a route, presenting what we believe is a novel quantitative model of *trust* relationships. This metric reflects the perceived threat quantitatively, and users change the confidence levels in response to exposed threats and vulnerabilities. We show how we can efficiently compute routes that maximize the confidence a user can expect given the current threat model and trust relationships. We explore these issues in the context of three representative environments—a military network, a ubiquitous computing scenario, and a peer-to-peer network.

Yi et al. [13] propose the notion of *secure routing* for ad-hoc military environments. To the best of our knowledge, this is the only work besides ours that attempts to accommodate users’ demands on secure routing. While

their work was limited to ad-hoc wireless routing environments and to certain credentials of the users, we present a generalized protocol based on different types of attributes of users and routers, as well as trust and threat assumptions between these entities.

In addition to security, a user may also be interested in setting up routes that preserve their privacy, in terms of location anonymity or identity anonymity. Our model enables users to set up routes through routers in a way that does not compromise their privacy, leveraging on our experience with Mist [1]. Users can specify trust and threat attributes to avoid certain nodes and prefer some routes over others, rather than relying on the system to make anonymous routing decisions.

We envision a network in which users operate under the overall network MAC policy, but have the flexibility to apply dynamic trust attributes and relationships for improved confidentiality, privacy, and availability guarantees of their communication. In the future we propose to study different computational models of confidence and extend our model with probabilistic analysis to capture stochastic behavior, and sensitivity analysis to study how changing trust relationships and threat models can impact the overall security of routes in a given network.

Overview—In Section 2 we present an overview of our system’s architecture. Section 3 discusses how users can specify path requirements based on desired path properties. Section 4 describes our trust model and how we quantify “good quality” routes. We present some applications of our approach in Section 5. After a discussion in Section 6, we present our conclusions in Section 7.

2 Architecture

In this section, we present a high-level architectural view of our proposed model consisting of different network elements. Similar to traditional PBNs, our network includes a policy database, PEPs, and PDPs. We focus our attention on corporate or private networks that are effectively isolated from the Internet at large and provide the adequate support to enforce cohesive administrative and management policies across this network. Within our network we also have the ability to certify different static and dynamic attributes of users and network objects, through a centralized or distributed trust authority.

As shown in Figure 1, users connect to our routing infrastructure through *access points*. Services can be connected to access points, or certain services may be available at the routing nodes itself (e.g., discovery services that are part of the routing infrastructure). Based on the certified attributes that the user chooses to disclose to the authenticating system (for privacy reasons the users may disclose only a subset of their current attributes),

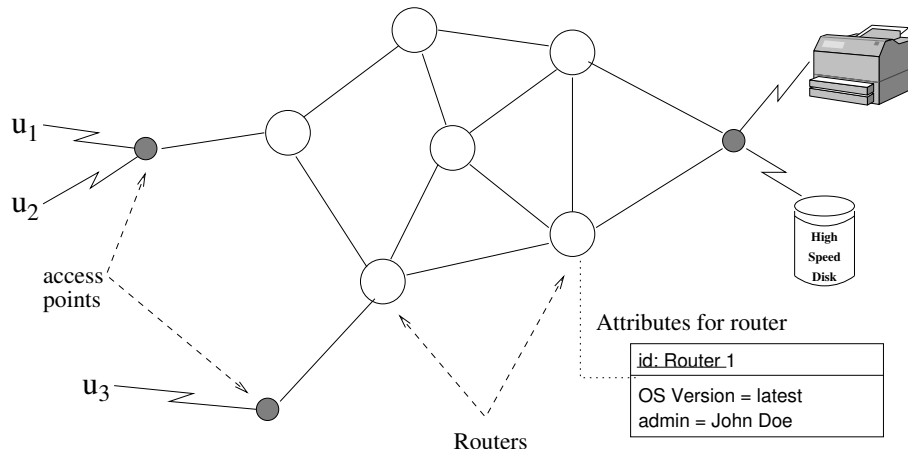


Figure 1. Architecture Overview

the user is presented with a snapshot of the system consisting of different network elements, including routers, links and servers. Note that this snapshot is a restricted view of the network, reflecting what resources a user is authorized to use based on the user’s disclosed credentials¹, according to the *mandatory* access policies of the organization.

The user hence possesses a logical view of the routers, their attributes, and their connectivity. When a user wishes to communicate with another entity on the network, he or she looks up the access point of the destination and computes a route to that access point. Within this view of the network, our framework allows users to restrict their preferences for routers, services and routes even further, in accordance with their discretionary demands. In the next few subsections we describe how each part of this process works, along with the trust negotiation and bootstrapping that occurs in the system. We begin with how attributes can be certified in our proposed system.

2.1 Attributes

We define three types of attributes to capture both the static and dynamic nature of evolving trust relationships in our system—*inherent attributes*, *consensus-based attributes* and *inferred attributes*. As we show later, these attributes help us quantify the trust relationships in the system.

Inherent attributes: These attributes are relatively static characteristics of an entity, which can be certified by a Certificate Authority (CA). Examples of inherent user attributes include identity, role, age, and gender. In-

¹We use *credentials* and *attributes* interchangeably since attributes are certified and are presented as credentials

herent router attributes include physical location, administrative authority, physical security, clearance level, and firewall security. A CA can create these certificates for inherent attributes and distribute them a priori to users and routers. For example, users can use these attributes to set up routes through routers that are physically secure and that belong to a certain trusted administrative entity.

Consensus-based attributes: These attributes relate to the behavior of an entity with respect to other entities in the system. For example, routers in the network can vouch for the integrity of neighboring routers if they appear to be routing packets correctly. A compromised router may stop forwarding packets, and neighboring routers would degrade their trust in that router with respect to packet delivery. Users can therefore use these dynamic attributes to set up routes through routers that have been routing packets reliably on a need-to-use basis. Routers may decide that a certain user is not honoring routing policies and exclude that user from future negotiations. For example, the user may be running a transfer protocol that does not have any congestion control mechanism (e.g., non TCP-friendly multimedia flows, or a denial of service attack). Hence routers may or may not vouch for a user’s behavior, which would hurt the user’s ability to set up future routes. This encourages good behavior of both users and routers within the network. Since these certificates are issued for the current behavior of a router or a user, it is impractical to have the CA issue such certificates.

Therefore, we need a robust and efficient protocol where routers and users can generate, agree, and distribute these relatively dynamic attributes. Since users and routers, especially compromised ones, can lie about these attributes, we plan to explore different intrusion

tolerant consensus protocols in the future, with different assumptions (including Byzantine failure) for this part of the framework. One option is to use COCA [16], an online certification authority that uses threshold cryptography to issue these certificates. The basic idea is that at least k out of n routers would need to agree on an attribute to issue a certificate for that attribute. COCA comes with built-in intrusion tolerance for Byzantine failures, and is reasonably efficient.

Inferred attributes: While entities in the network may have inherent or consensus-based attributes, users may have reasons not to trust certain routers, and likewise, certain routers may not trust certain users. For example a user might use probes to infer that certain routers are running outdated versions of software with a known vulnerability. This is an indicator that the router may be compromised and is not trustworthy. Hence a user may want to avoid such routers. Since these are attributes that the user assigns to routers (or vice versa), these attributes are local to the entity making the inference. No certification is required for such attributes. Other examples include latest patches, daemon processes, past behavior observed by the user, etc.

In the next subsection, we briefly explore how to accommodate a user’s privacy preferences.

2.2 Trust negotiation

We desire a system that honors the privacy of users. A user would like to reveal only those attribute certificates that are absolutely necessary to accomplish the user’s goals. For example, a user may want to use the network as a *Student*, without revealing the actual identity. Since the logical view of the network depends on the credentials of the user, this view is restricted based on the attributes the user reveals to the network. Moreover, when a user demands consensus-based attributes of routers, the router may first demand that user present credentials appropriate to that demand. For example the router may disclose routing statistics only to users with a high level of security clearance (high-priority users). This suggests the use of trust negotiation protocols such as those proposed by Yu et al. [15]. Such protocols can be effectively used to bootstrap trust between users and routers based on inherent and consensus based attributes.

2.3 Routing model

In this subsection, we formalize our routing model and describe how users can specify their discretionary policies based on attributes of routers in the organization. As explained before, users can obtain a map of the network that they are authorized to view according to the organizational mandatory policy at startup. This map

lists all the routers, and links, and labels each router with the set of static attributes that are valid on that router. Users are allowed to update this map with dynamic attributes at any point in time.

We model our network as a Kripke structure in order to take advantage of what is called model checking [4] and its accompanying formalism. Formally, a Kripke structure is the tuple $M = \langle S, S_0, R, L \rangle$, where S is a set of states, S_0 is the set of initial or start states, $R \subseteq S \times S$ is a total transition relation between states, and $L : S \rightarrow \mathcal{P}(AP)$ is a labeling function where $\mathcal{P}(AP)$ is the power set of atomic propositions AP . Given a state $s \in S$, $L(s)$ is the set of atomic propositions that are true in s .

Note how this formalism corresponds almost exactly with the attribute-based routing framework we propose. The set of routers corresponds to the set of states S in the model. If two routers s_1, s_2 are connected then $(s_1, s_2), (s_2, s_1) \in R$ since we assume symmetric links. Note relation R is total since our links are bidirectional. Each relation in R corresponds to the connectivity between routers. The set of attributes at each router can be viewed as atomic propositions (or truth valued statements) about attributes in that that state. Therefore the set AP is the set of all possible attribute-value pairs in our system. Note, this set is finite in our model. The set of start states S_0 are specified by the user.

Since the network is logically mapped to a Kripke structure, users can define their discretionary policies as path characteristics using temporal logic formulas that can be interpreted over what is called a computation tree of a Kripke structure. Formally, an infinite computation tree is obtained by unwinding the state-transition graph by starting with a fixed start state and applying all transitions from that state to other states in the model, and so on. Different types of temporal logic have been studied extensively in the past [4] to describe properties of these infinite computation trees. We believe that the most useful logic for our case is Linear Temporal Logic (LTL) which is used to specify characteristics of paths in this tree. We do not define the syntax and semantics of LTL as it is well known, but explain how we can use it to specify properties in the next section.

Our biggest motivation for using this formalism is the availability of automatic tools that can compute efficiently whether there exists a path in our model that satisfies the constraints imposed by the LTL formula. This process is called model checking. A model checker always provides a counter-example (if one exists) to a property specified by the user. Specifying the negation of a desired property, yields a path (counter-example) with the desired property. Model checkers can be modified to return more than one counter-example to yield all paths that satisfy a specific type of LTL formula [8].

We show how we can adapt this technique in the next section, and present a discussion on the suitability of model checking, its time complexity, and highlight specific characteristics of our model that make it particularly scalable with respect to the “state space explosion” problem.

3 Path specification

LTL formulas are a powerful way for users to express path requirements. As explained in the previous section, model checkers can be used to generate multiple paths, when they exist, that satisfy these constraints between a source access point and a destination access point in our model. Model checking algorithms for LTL formulas in general have time complexity $O(|M|2^{O(|f|)})$, where $|f|$ is the size of the LTL formula. However, note that the algorithm is tractable in general for LTL formulas of small size, and we expect that user specified LTL formulas will be reasonable in size. As we discuss below, there is more of a concern with the state space explosion problem (the size $|M|$), which is not a concern for us since $|M|$ is the size of the network, and there is no state space explosion in our model.

In this paper, we describe how we can optimize this algorithm for a subset of LTL formulas that reduces to the shortest path problem, which is polynomial in time (e.g., Dijkstra’s shortest path algorithm). For future work, we plan on exploring the utility of more complex LTL formulas. It is useful to note here that model checking is associated with what is called the “state space explosion problem,” which usually makes it unusable for a large system with dynamic behavior. Typically, a state transition occurs in a Kripke model when the truth values of the atomic propositions in that state change. As a result, computation trees that represent all possible behaviors of the system by enumerating states and transitions for all combinations of changes of these values, can become very large. In our case, the attribute certificates are fixed for a particular view of the network. We do not model the changing values of these attributes as different states for each router. The transitions can only occur between routers that have links between them in the real network we are modeling. Therefore, we can limit the size of our model by number of routers, links, and attributes in our network. In terms of model checking overhead, we do not suffer from the state-space explosion problem, and we are only limited by the complexity of algorithms for verifying LTL formulas. In general, if the length of the formula $|f|$ is small in comparison to size of the network (Kripke structure), then for all practical purposes the model checking takes place in asymptotic linear time with respect to the size of the network. Hence we believe that the use of model checking

for computing paths according to LTL specifications can be a powerful and efficient tool.

Manna and Pnueli [6] in their discussion on the expressive power of temporal logic discuss three useful classes of path properties—Invariance, Response, and Precedence. Invariance properties are true in every state in a path. These properties are useful to model user constraints such as “Only route through nodes that support IPSec.” Response properties are useful to model quantitative properties of bidirectional paths, e.g., in terms of round trip latency or available bandwidth. Precedence properties capture the causal relationships between properties along a path. We explore these properties in turn and show they can be specified in LTL.

3.1 Global or invariance properties

Consider LTL formulas of the form $\mathbf{G} p$. \mathbf{G} is the “globally” operator which means that in all states along the path, proposition p must evaluate to true. We restrict p to propositions on the attributes. The user requires that p must hold at all routers. The algorithm for computing paths that satisfy $\mathbf{G} p$ first eliminates all nodes from the graph (Kripke structure) where p does not hold. This solely depends on attributes at each router, and attributes at one router do not affect the satisfiability of p at another router. The graph that we are left with represents the routers that the user is willing to route through.

3.2 Response properties

These properties are of the form $\mathbf{G} (p \rightarrow \mathbf{F} q)$ where \mathbf{F} is the “finally” operator. The formula asserts that it is always true on our path that if proposition p is satisfied at any node, eventually proposition q will be satisfied. This property is useful to specify bounded-response and causal relationships between attributes. Quantitative versions of these properties (obtained by augmenting both the model and the temporal logic carefully with time variables as in [2]) can be used to specify path latencies and bandwidth constraints.

3.3 Precedence properties

We look at the case when certain attributes along the path must occur in a specific order. For example, the user may want to set up a path that goes through routers in a non-decreasing order of classification levels. Once a packet enters a router with high level of security, it must not pass through a node with lower security. Consider the case when routers append sensitive information to packets. If the packet is at a certain router, it can never contain previous data from a higher clearance router, and hence there is no information leakage. The

user can specify an attribute ordering p_1, \dots, p_n , where exactly one of these is true at every router. If p_i and p_j occur along a path, it must be the case that p_i occurred before p_j . This can be specified with the LTL formula: $\neg \bigvee_{i>j} \mathbf{F}(p_i \rightarrow \mathbf{F} p_j)$. Given this specification, we can remove all edges from the graph that violate the attribute ordering. Consider an edge (s_1, s_2) . If $i > j$, and $p_i \in L(s_1), p_j \in L(s_2)$, then we remove the edge (s_1, s_2) from the graph. Hence no path in the resulting graph can violate the precedence specified by the user. Moreover, any valid path that satisfies the precedence property in the original graph, also exists in the resulting graph, and these paths are exactly those in the original graph that satisfy the precedence property.

Note that the user can specify global and precedence properties independently. These transformations on the graph described above are commutative. Given global, response, and/or precedence requirements specified by the user, we combine the resulting graph with the trust model described next to find paths of highest “confidence.”

4 Trust model

Once a user transforms the graph (as described above) of the network satisfying the attribute requirements (e.g., via the user supplied $\mathbf{G}p$ LTL formula or precedence formula), the user would like to set up a route to a destination. A naïve solution would be to obtain the shortest path (in terms of hops) to the destination. However, if the network is under attack, some paths are more trustworthy than others. For example, it may be known that there are intruders in the system with physical access to machines. One would like to degrade trust in routers that have lower physical security. It may be known that certain machines have been compromised without knowing the specific machines. In such a case, users may degrade trust for machines run by certain administrators, or for those machines that are running out of date software.

Such a scenario suggests that we integrate this notion of threats in the graph-based formalism we proposed so far. We propose a quantitative measure of this interplay between threat and trust as the *confidence* a user has in the validity of the attribute under question. Users can assign confidence levels to routers as a function of one or more of their attributes of interest.

Definition 1. Given a router $s \in S$ with attributes $L(s)$, a user’s **confidence function** $C : S \times \mathcal{P}(AP) \rightarrow [0, 1]$ returns the **confidence level** for a router. We abbreviate the confidence level $C(s, L(s))$ of router s as c_s .

The exact nature of this confidence function will depend on the types of attributes and how these values

can be composed to compute the confidence value of a path. The confidence function we choose will depend on this composition operator as we explain in the next section. We believe that assigning these confidence levels to routers is an important area of study and propose to work on it in the future. In the next subsection, we discuss how we can compute paths of high overall confidence based on confidence levels of routers along the path.

4.1 Trusted paths

We refer to any path from router a to router b as an a, b -path. We assume that the user/sender is connected through access point a , and that $c_a = 1$ since the user has to use that as their first hop, and that the destination is either b or a user whose access point is b . In either case we treat a and b as the endpoints of communication.

Definition 2. The **path confidence** C_π of an a, b -path π is obtained by applying a **combiner function** $K(c_1, \dots, c_n)$ that takes all the confidence levels c_i of the n routers s_i along the path π from a to b ($s_1 = a, s_n = b$), and returns a confidence value for the path in $[0, 1]$.

We explore different combiner functions in this context. To illustrate, consider the concept of “weakest link.” There may be routers that are highly vulnerable, and it is extremely likely that they will be chosen for attack. The path confidence in this case can be defined as the *minimum* of all confidence values of routers along the path. Here $K(c_1, \dots, c_n) = \min\{c_1, \dots, c_n\}$. So when a user needs to pick a path based on its combined confidence value, he or she can avoid paths with the lowest path confidence levels.

Also consider the following example. A user may conclude that the DoS vulnerability of a router is proportional to the number of incoming links. Hence the user would like a path that minimizes the average sum of incoming links over all routers along a path, and does not include any nodes with very high connectivity. In this case, the user can use a second order statistic such as variance to decide which path has the best “Quality of Protection” for the given scenario.

One combiner function we focus on in the next subsection is the multiplication function. A multiplicative measure of path confidence can be used to model various properties of interest to a user: probability of success of delivery, probability of no information leakage, probability that routers along a path will not collude, etc. In the next subsection, we explore this in some detail and describe efficient algorithms to compute path confidence values using a multiplicative combiner function.

We note that there may be policies that combine several such models and seeks optimal paths based on several constraints (additive, multiplicative, weakest-link, etc.). We are currently classifying such policies and their corresponding algorithms.

4.2 Multiplicative combiners

We consider the case when $K(c_1, \dots, c_n) = c_1 \dots c_n$, the product of confidence levels of routers along a path. This multiplicative model of path confidence that we focus on, applies to confidence levels that were computed independently along a path. In this model, a user assigns confidence levels based on the probability of “good things happening” at each node. Assuming independence, the probability of the desired property being true along the entire path is simply the product of all the confidence levels. We now present an efficient method for computing paths of high path confidence under the multiplicative model.

The main idea behind computing paths of high confidence is that by applying the correct weights to edges in a network connectivity graph, we can use shortest path algorithms (that use additive weights) to find paths with highest overall confidence (based on multiplicative weights).

Consider the directed graph G that represents the connectivity of routers specified by the Kripke structure M . For each $s \in S$, we now assign $-\ln(c_s)$ to be the *weight* of all incoming edges to s . Note that all weights are non-negative since confidence levels are in the range $[0, 1]$. We now have a weighted directed graph G . Consider a source node a and a destination node b .

Lemma 1. *Let s be the sum of weights on the a, b -path π in G . The path confidence \mathcal{C}_π of π is equal to e^{-s} .*

Proof. Let c_1, \dots, c_n be the confidence levels of all the routers in π except a . $\mathcal{C}_\pi = c_1 c_2 \dots c_n$ since $c_a = 1$. Now $s = \sum_{i=1}^n -\ln(c_i) = -\sum_{i=1}^n \ln(c_i) = -\ln(c_1 c_2 \dots c_n)$. Hence $e^{-s} = \frac{1}{e^{\ln(c_1 c_2 \dots c_n)}} = c_1 c_2 \dots c_n = \mathcal{C}_\pi$.

Note that if there exists a $c_i = 0$, then the path confidence is 0. Here, $s = \infty$ since $-\ln(0) = \infty$ and $e^{-s} = 0$, and there is no discrepancy for confidence levels of 0. Essentially, any path which includes a router of 0 confidence will not be chosen by the user. \square

Lemma 2. *For any two a, b -paths π_1, π_2 with total weights w_1, w_2 , we have $w_1 \leq w_2$ if and only if $\mathcal{C}_{\pi_1} \geq \mathcal{C}_{\pi_2}$.*

Proof. From Lemma 1 we have that $w_1 \leq w_2 \Leftrightarrow -w_1 \geq -w_2 \Leftrightarrow e^{-w_1} \geq e^{-w_2} \Leftrightarrow \mathcal{C}_{\pi_1} \geq \mathcal{C}_{\pi_2}$. \square

Theorem 1. *The k shortest a, b -paths in G correspond to the k a, b -paths of highest path confidence in G .*

Proof. This follows from Lemma 2 since if we order all the a, b -paths in G in increasing order of weight, they are ordered in decreasing order of path confidence. \square

Since all edge weights are non-negative, Theorem 1 allows us to apply k shortest simple (loopless) path algorithms to find cycle-free paths of highest confidence. For example, Dijkstra’s algorithm is the special case when $k = 1$ and will yield a path with maximum path confidence. Several algorithms have been proposed for obtaining the k shortest simple paths in a directed graph. The best known worst case time complexity of these algorithms is $O(kn(m + n \log n))$ [11, 12]. Hershberger et al. [5] propose an algorithm that provides a $\Theta(n)$ improvement in most cases. For small k (for example, the user may want the 3 highest confidence paths) these algorithms are efficient for all practical purposes. Hershberger et al. [5] provide results of their algorithm for large graphs (e.g., 5000 nodes, 12000 edges) based on real GIS (Geographic Information Services) data for road networks in the United States.

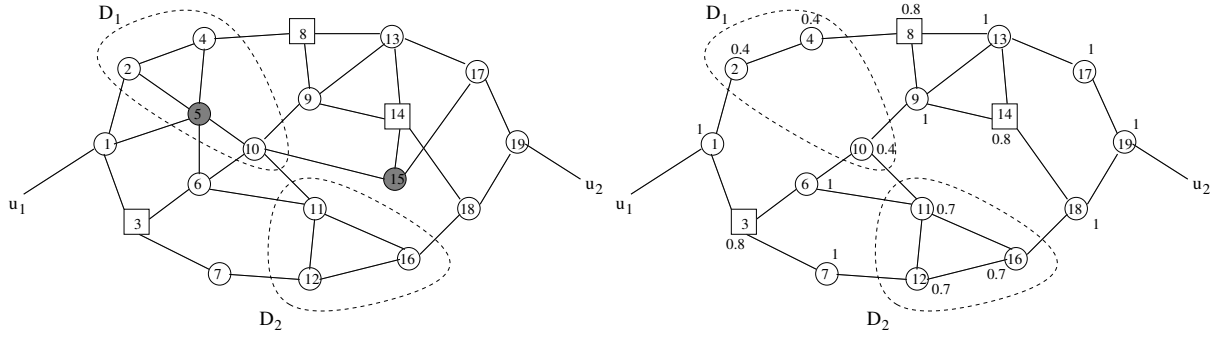
In addition to the models we present in this section, we argue that the ability to specify both threat and trust relationships using a combined metric is extremely powerful. We plan to study how these values can vary over time, using sensitivity analysis, stochastic analysis and other techniques.

5 Applications

We present three concrete examples that showcase the benefits of our new framework. We present the first example in more detail, and suggest two other uses.

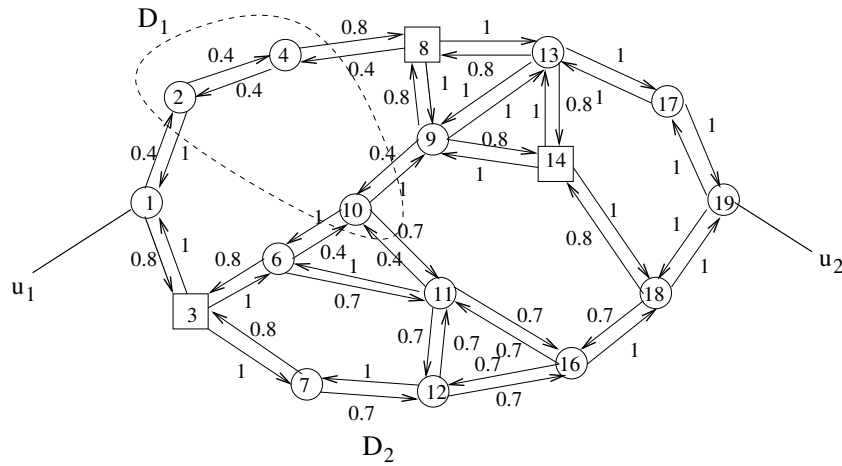
5.1 High performance and military environments

Consider an MLS (Multilevel Security system) user u_1 with sensitivity level *Confidential* in compartment $\{Navy\}$, connected at the access point s_1 . User u_2 has security clearance $\{Confidential, \{Army\}\}$ and is connected at access point s_{19} . Based on u_1 ’s clearance (u_1 chooses to only reveal this, not its identity), the system presents the user with a logical view of the network as shown in Figure 2(a). All routers in this system are cleared for $\{Confidential, \{Army, Navy\}\}$. For simplicity we look at only two inherent attributes: *physical security*, which can be *high* (unshaded nodes) or *low* (shaded nodes), and *domain*, which can be D_1, D_2, D_3 or D_4 (we only show D_1 and D_2 in Figure 2(a) since we



(a) Logical view for u_1

(b) Resultant view based on user's policy, including confidence levels



(c) Resultant digraph for use with k shortest path algorithms

Figure 2. Military network example

use them as one of the constraints later). D_1 can correspond to a confidential network owned by the Army for example.

User u_1 desires to communicate with u_2 and determines that u_2 's access point is s_{19} . We assume a network component analogous to dynamic DNS which can respond with a user's current access point (u_2 has chosen to register its access point with the service). Now u_1 has been informed by trusted sources that there is an intruder physically located on the premises, and that low physical security routers should be excluded. u_1 specifies the following LTL formula $\mathbf{G} \textit{physical security} = \textit{high}$. This eliminates s_5, s_{15} from the logical view and results in the graph shown in Figure 2(b).

By means of network probes, u_1 determines the inferred attribute *OS version*, which can be *outdated* (square nodes) or *latest* (round nodes). u_1 assigns a confidence level of 1 to all routers. Routers with outdated

operating system versions (*OS version = outdated*) have their confidence levels multiplied by 0.8 since they may be compromised. Lastly, u_1 would like to avoid machines in domain D_1 because of a suspected insider attack in that domain. User u_1 multiplies the confidence levels of routers in this domain by 0.4. User u_1 has experienced large delays when routing through D_2 , and degrades confidence in those routers by multiplying their confidence levels by 0.7. Figure 2(b) shows these confidence levels for each node. Figure 2(c) shows the resulting digraph with multiplicative weights. As described in Section 4.1 we replace these weights by their negative natural logarithm, and then apply k shortest path algorithms [5] to obtain the three paths of highest confidence. In this example it is easy to see that the following are paths with the three highest path confidences: $\langle 1, 3, 7, 12, 16, 18, 19 \rangle$, $\langle 1, 3, 6, 11, 16, 18, 19 \rangle$, and $\langle 1, 3, 6, 10, 9, 13, 17, 19 \rangle$. The first two paths have a

path confidence of 0.392 (with respect to the logarithmic weights, the total weight is 0.9365), and the third has a path confidence of 0.32 (weight 1.139).

Once these three paths are obtained, the user needs to set up a path through the routers. This is done using a scheme that encrypts the packet multiple times, based on the routers in the path, similar to *onion* routing [7], since public keys of routers are assumed to be well known to u_1 . The user encrypts the path in reverse order using the keys of the routers in the reverse path. Each subsequent router decrypts the received route setup packet to obtain the next hop and an encrypted route setup packet for the next router. This technique hides the path from the routers, which only know the previous and next hops in the path. By means of this route setup, u_1 can establish the chosen path to u_2 . Packets from u_2 to u_1 are simply forwarded on the reverse path.

5.2 Ubiquitous computing

The previous example provided a detailed overview on how our system works in a military environment. In this section we briefly discuss applications to ubiquitous computing. Users in ubiquitous computing environments seamlessly interact with numerous devices and services. In such an environment *discovery* of services is one of the main applications. However, with such an environment it is very easy for the ubiquitous system to track a user's movements or record user patterns. Using our system as a basic infrastructure or service, users can maintain their privacy. Users only need to reveal as much information as necessary to get a logical view of the ubiquitous environment. Again, this is achieved by trust negotiation as described in [15]. In a university setting, a user may want to avoid using routers or services that belong to other research groups, and eliminate these by using global property specifications. While connecting to certain services, a user may choose to maintain location anonymity (for example, using Mist [1]) by creating a route that is hard to trace back. The user can assign lower confidence levels to the domains that the user does not trust since routers within a domain can presumably collaborate to expose the location of the user. This will give paths of higher confidence that the user trusts for higher location privacy.

5.3 Peer-to-peer overlay networks

We consider peer-to-peer networks where it is feasible for users to obtain topological information of the overlay. We assume the user can form a logical view of the overlay network based on information available to the user. The user desires to perform searches for

content available at each peer. Applications include distributed file systems, file sharing, etc. Based on attributes of peers in the overlay, the user can choose to only search for content at routers that satisfy global property specifications. That is, the user can avoid performing searches at untrusted nodes for privacy reasons. Additionally, the user may assign lower confidence levels to nodes on which it expects the search to fail. This can be based on past performance (inferred attribute). If certain nodes seldom have content of interest to the user, the user can assign lower confidence levels. On the other hand, if a biology student is searching for research papers related to cell division, confidence levels for peers in the computer science department can be degraded since there is a very low chance of finding useful content. Hence the user can set up a search path with the highest confidence, so that the probability of the directed search succeeding is high.

6 Discussion and future work

In this paper we focus on particular LTL specifications and provide efficient algorithms for finding paths that meet those specifications. Pnueli et al. [6] argue that global, response and precedence properties are of most interest as path properties, and this provides some justification to this focus. We are currently exploring more complex user specifications in LTL and possible algorithms for computing such paths. We are also interested in using model checkers as a blackbox. The user can provide complex LTL formulas and the model checker would return paths according to the specification. Sheyner et al. [8] use a similar technique for generating attack graphs in a network, although they are susceptible to the state space explosion problem because of the way they model attack paths. Despite its benefits, LTL may be somewhat limiting since all requirements do not have an equivalent LTL specification. For example, a user may want a path that passes through a certain threshold number of domains, which cannot be represented without quantification on the domain attributes, which is not allowed in LTL. In anonymous routing, this would ensure that multiple domains would have to cooperate to expose a users location. This suggests the use of a higher level language (possibly graphical) coupled with efficient algorithms.

We also want to explore how a user can assign confidence levels to routers through a confidence function. We believe that this is a powerful model for intrusion-tolerant routing in networks under attack conditions. Users can assume a certain threat model and update confidence levels of routers that are suspected of being compromised. Our algorithm will compute paths of highest confidence, which can then be used for intrusion tolerant

routing by routing along multiple high-confidence paths. From a usability perspective, it is unreasonable to expect users to be experts in policy definition languages. Users would need a more intuitive interface for making discretionary demands, which would then be translated to LTL formulas. We defer higher-level policy languages for future consideration.

We assume global knowledge of router connectivity within a network. While this assumption is reasonable for private networks, a scalable solution for larger networks may require aggregation of routers, where attributes are applied to aggregates. Applying our model to ad-hoc networks would require further consideration based on the degree of mobility of nodes. Users may have to work with partial information on global connectivity, and would need to reestablish routes more often.

7 Conclusion

We extend the mandatory access control nature of PBN systems by empowering users to make discretionary routing choices within a network. We propose a formal model based on Kripke structures and define a metric we call *confidence level* that captures both trust and threat attributes quantitatively. We present techniques for computing routes with high *path confidence* based on a user's preferences, within the constraints of a mandatory organizational policy, for two cases—global and precedence specifications. Based on confidence levels provided by the user, we provide a transformation of the graph using negative logarithmic weights that allows us to use k shortest path algorithms to efficiently obtain paths of highest confidence.

8 Acknowledgments

We thank John Fischer, Erin Wolf and Mahesh Viswanathan for their helpful comments.

References

- [1] J. Al-Muhtadi, R. Campbell, A. Kapadia, D. Mickunas, and S. Yi. Routing Through the Mist: Privacy Preserving Communication in Ubiquitous Computing Environments. In *Proceedings of The 22nd IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2002.
- [2] R. Alur and T. A. Henzinger. Logics and models of real time: A survey. In *Real Time: Theory in Practice, Lecture Notes in Computer Science 600*, Springer-Verlag, pp. 74-106., 1992.
- [3] R. H. Campbell, Z. Liu, M. D. Mickunas, P. Naldurg, and S. Yi. Seraphim: Dynamic Interoperable Security Architecture for Active Networks. In *OPENARCH 2000*, Tel-Aviv, Israel, March 26–27, 2000.
- [4] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 2000.
- [5] J. E. Hershberger, M. Maxel, and S. Suri. Finding the k shortest simple paths: a new algorithm and its implementation. In *Proceedings, 5th Workshop Algorithm Engineering & Experiments (ALENEX)*. SIAM, Jan 2003.
- [6] Z. Manna and A. Pnueli. The temporal logic of reactive and concurrent systems. *Springer-Verlag*, 1992.
- [7] M. G. Reed, P. F. Syverson, and D. M. Goldschlag. Anonymous Connections and Onion Routing. *IEEE Journal on Selected Areas in Communication: Special Issue on Copyright and Privacy Protection*, 1998.
- [8] O. Sheyner, S. Jha, and J. M. Wing. Automated Generation and Analysis of Attack Graphs. In *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, May 2002.
- [9] M. Sloman and E. Lupu. Security and Management Policy Specification. *Special Issue on Policy-Based Networking*, 16(2), March 2002.
- [10] A. Westerinen, J. Schnizlein, J. Strassner, M. Scherling, B. Quinn, S. Herzog, A. Huynh, M. Carlson, J. Perry, and S. Waldbusser. Terminology for Policy-Based Management. RFC 3198, November 2001.
- [11] J. Y. Yen. Finding the K shortest loopless paths in a network. In *Management Science*, volume 17, pages 712–716, 1971.
- [12] J. Y. Yen. Another algorithm for finding the K shortest loopless network paths. In *Proceedings of 41st Mtg. Operations Research Society of America*, volume 20, 1972.
- [13] S. Yi, P. Naldurg, and R. Kravets. Security-Aware Ad Hoc Routing for Wireless Networks. Poster presentation, ACM Symposium on Mobile Ad Hoc Networking & Computing (Mobihoc), 2001.
- [14] S. Yi, P. Naldurg, and R. Kravets. Integrating Quality of Protection into Ad Hoc Routing Protocols. In *The 6th World Multi-Conference on Systemics, Cybernetics and Informatics (SCI)*, Orlando, Florida, August 2002.
- [15] T. Yu, M. Winslett, and K. E. Seamons. Supporting Structured Credentials and Sensitive Policies through Interoperable Strategies in Automated Trust Negotiation. *ACM Transaction on Information and System Security*, February 2003.
- [16] L. Zhou, F. B. Schneider, and R. van Renesse. COCA: A Secure Distributed On-line Certification Authority. *ACM Transactions on Computer Systems*, 20(4):329–368, November 2002.