A Switch, in Time*

Sanjiva Prasad[1] and Lenore D. Zuck[2]

1 **Department of Computer Science and Engineering, Indian Institute of Technology Delhi**
New Delhi, India
sanjiva@cse.iitd.ac.in

2 **Department of Computer Science, University of Illinois at Chicago**
Chicago, USA
lenore@cs.uic.edu

Abstract

Communication networks are quintessential concurrent and distributed systems, posing verification challenges concerning network protocols, reliability, resilience and fault-tolerance, and security. While techniques based on logic and process calculi have been employed in the verification of various protocols, there is a mismatch between the abstractions used in these approaches and the essential structure of networks. In particular, the formal models do not accurately capture the organization of networks in terms of (fast but dumb) table-based switches forwarding structured messages, with intelligence/control located only at the end-points.

To bridge this gap, we propose an extension of the axiomatic basis of communication proposed by Karsten et al. In this paper, a simple model of abstract switches and table-based prefix rewriting is characterized axiomatically using temporal logic. This formulation is able to address reconfigurations over time of the network. We illustrate our framework with simple examples drawn from SDNs, IPv6 mobility and anonymous routing protocols.

1998 ACM Subject Classification C.2.1 Network Architecture and Design. C.2.2 Network Protocols. D.2.4 Software/Program Verification. D.4.6 Security and Protection. E.1 Data Structures. F.3.1 Specifying and Verifying and Reasoning about Programs. F.4.1 Mathematical Logic.

Keywords and phrases Abstract switches, network protocols, data plane, control, time-dependent behavior, correctness.

1 Introduction

Communication networks are quintessential concurrent and distributed systems. Apart from issues pertaining to scale and efficient implementation, networks pose challenging verification and validation problems such as the correctness of protocols, reliability, resilience and fault-tolerance, and security. Indeed, the correct operation of most distributed systems depends on the correctness of networks and their protocols.

Networks are organized as a "stack" of several protocol layers, from the physical to application (and beyond), with the workhorse being the network layer and the suite of IP protocols configured as a reticulum of fast (but "dumb") table-based routers/switches that forward packets. "Intelligence" is confined to the peripheries of the data forwarding network [5]. Much of the robustness and efficiency of the Internet derives from this architecture. Proposals to embed greater intelligence into the data network often (i) suffer from inefficiency, (ii) are difficult to deploy, configure and maintain, and (iii) may introduce potential vulnerabilities.

* This work was partially supported by a grant from Microsoft Research to the first author, and NSF CCS-1228697 to the second author.

Conference title on which this volume is based on.
Editors: Billy Editor and Bill Editors; pp. 1–14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Several formal techniques have been used to model and verify different aspects of a wide variety of network protocols. Approaches based on automata and temporal logic have traditionally been used to reason about fundamental protocols. Modal logics (particularly of knowledge) have been used to formulate protocol correctness [10, 12]. However, verifying whether a system satisfies a given property usually requires considerable expertise since one has to formalize within the framework the behavior of all agents, including that of the adversary/environment.

The AVISPA project [3] and subsequent lines of research, e.g. [4, 7] comprise a large body of work involving the automated validation of internet security protocols. Several security properties have been expressed using LTL with a "sometime in the past" operator [17], following which model-checking techniques have been used to verify or detect errors in a large number of widely used protocols and their salient properties. Such *Safety Temporal Properties* are of the form $\Box(P \rightarrow \diamondsuit Q)$, which roughly states: *"For any reachable state satisfying a property P, there was a past state satisfying Q."*

An alternative approach uses process calculi to provide a formal operational model of the system, and where the adversarial context is mapped to the generic notion of an observer. For example, CCS has been used for modeling the alternating bit protocol [18, 6]; the π-calculus for mobile handover protocols [19]; and the applied π-calculus for security protocols [1]. While the process calculus approach is attractive as it comes equipped with notions of equivalence such as bisimulation and their associated proof techniques, stating logical properties about the protocols is not always easy.

Although these approaches have been successful in formally modeling and verifying various network protocols, they have done so by abstracting away much of the essential structure of communication networks. There is a mismatch between the structure of networks and the manner in which they are represented in these *ad hoc* (used in the original, non-pejorative sense) abstract models. While the hierarchical structure of the network stack presents apparently abstract and modular structures, these do not correlate well with the formal notions of abstraction and refinement. When attempting to refine the elegant abstract models to account for the nitty-gritty details of network structure and behavior, the formal approaches often end up "over-modeling" mundane or irrelevant details about, say, data delivery. Another issue is the lack of modularity when combining different analyses (e.g., routing and security) that were independently modeled and verified under assumptions of orthogonality, and often in quite different formalisms.

We therefore posit that it is important to have a simple modular framework for reasoning about the delivery of data messages that corresponds well with the actual structures of communication networks over which other analyses may be formalized. A presentation that is general enough to capture the complexities of existing data networks which is also compatible with existing formal operational and logical models (for modeling and specification) will facilitate robust proofs of correctness. Indeed, we hope that the many of analyses in the literature can be adapted and refined to work in a modular fashion with such a framework.

In this paper, we propose a framework in which data communication network structures and behavior may be represented at *any* appropriate level of detail. The framework naturally admits *refinements* in the description of the network elements as well as in the *logical specification* of their behavior. An organizing principle in our treatment is a systematic separation between the data and control planes. This factoring of network structure and function into a data plane and control plane has gained far greater importance with the advent of *Software Defined Networks* (SDNs) [15], which are motivated by the need to respond to the dynamics of network events and to enforce policies in continuously changing

environments.

The main contribution of this paper is an abstract presentation of the structure and operational behavior of the data plane. The description of the data plane is (in logical terms) an extension of the formal model of abstract switches presented in [14]. The advantages of choosing that framework lie in (1) its ability to formalize fundamental network concepts (names, addresses, links, scopes, etc.) at all layers of the so-called network stack as well as across layers; (2) its operational simplicity (being based on table-based prefix rewriting); (3) its axiomatic treatment of arbitrarily complex multi-layer forwarding systems [14]; (4) its natural support for refinement and compositionality [21]; (5) that it can be easily interfaced with process calculi, typically for expressing the control protocols.

The new contribution of this paper is a reformulation of the axiomatic basis for communication using temporal logic. This presentation enables a description of time-varying routing behavior and associated properties. It also supports a more natural form of reasoning about network invariants, abstracting from detailed reasoning about topological structure. The version of LTL that we employ in this paper is expressive enough for stating the desired time-dependent properties, and admits hierarchical reasoning about the model [16].

We illustrate our approach by presenting a fresh and concise analysis of the IPv6 mobility model using temporal logic. This is in contrast to the detailed process calculus model of [2], where the correctness properties were not explicitly stated, and the correctness of the protocol was expressed as barbed equivalence [22] to a simpler reference system. We next present an extension that enables switches perform cryptographic operations on messages. We model an anonymizing network service such as TOR [8], and outline how one may reason about the correctness of the design of such anonymizing network services by using knowledge modalities. In our exposition of the framework and approach, we have attempted to convey the intuitions, restricting the formalism to the bare essentials.

Paper structure.

In Section 2 we reformulate the basic model of abstract switches proposed in [14, 21] in temporal logic terms. We extend the framework to allow dynamic updates of switching tables, with an example drawn from simple updates in SDNs. Section 3 presents the IPv6 mobility protocol and a fresh proof of its correctness. Section 4 endows the switches with the ability to perform functional transformations on messages, the use of which is illustrated via a simplified version of the TOR anonymizing network service, and an outline of its correctness using knowledge modalities. We conclude in Section 5 with a recap of the main ideas of this paper, and some directions for future work.

2 The model

The architecture of our model reflects ideas from *Software Defined Networks* (SDNs). The network's operation is decoupled into a *data plane* — a collection of fast (unintelligent) table-based switching elements that relay messages — and a *control plane*, the part of a network that carries signaling traffic and is responsible for routing. The occurrence of *events* in the data plane triggers the control plane to initiate changes in switches/routers in the data plane. We describe here the structure and behavior of the data plane, which is designed to be a generic message delivery architecture, agnostic to the control operations. The data plane can be seen as a massive distributed system, with many orders of magnitude more messages than those in the control plane. The purpose of most control plane protocols is to configure or make changes in the data plane, which should continue to operate concurrently

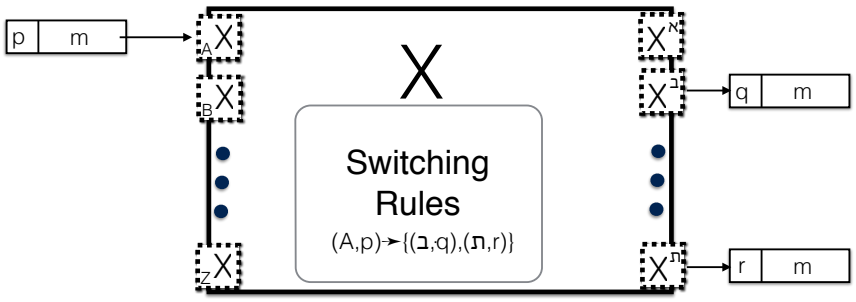

Figure 1 Schematic view of an Abstract Switch

while those changes are effected. Network protocols in the control plane may be described in any of a variety of process calculi or other formalisms. The examples in this paper do not focus on the specification and operation of the control protocols, but models their effects on the data plane. While our account omits the precise interface between the two planes, we believe this would be relatively straightforward to specify.

2.1 The data plane model

The data plane is a directed graph the *nodes* of which are *abstract switches* (ASs), denoted as $A, B, C, \ldots X, Y, Z$. Switches represent *any* processing elements — hardware or software — at *any* level of the network stack. Each AS has a number of named input and output *ports*. These ports are the end-points of simplex directed *edges* (written $e(A, B)$) between adjacent ASs. We use the notation A^B and ${}_AB$ to denote the output port at A directly connected to B, and the input port at B with an incoming edge from A. We write ${}_?B$ and $A^?$ when we wish to leave unspecified a port's connection. Generic ports are denoted x, y, z.

The ASs send *messages* along the edges. Messages are represented as *strings of identifiers* drawn from an arbitrary alphabet Σ. Typical messages are denoted $m, m', \ldots$ and *prefixes* of messages are usually denoted as $p, p', \ldots$ The presence of a message m at a port x is denoted as $m@x$. Creation and consumption of messages may be internalized within the system as follows: The *creation* of a message m at B is modeled as m being at a (virtual logical) port ${}_0B$, and its *consumption* as it being at the (virtual logical) port B^0.

An AS maintains a local *switching table*. The switching table at AS B, denoted as S_B, contains mappings of the form $\langle A, p\rangle \mapsto \{\langle C_i, p'_i\rangle\}$, which maps an AS-prefix pair to a *set of AS-prefix pairs*. The table represents a *finite-domain function*, which may not be defined for several $\langle A, p\rangle$ pairs. The switching table specifies a local prefix rewriting system at that AS. We assume that switching table entries are *exact matches*, i.e., if $\mathsf{S}_B[A, p] \neq \emptyset$, then for no prefix p' of p is $\mathsf{S}_B[A, p'] \neq \emptyset$[1]. The correctness of computer network protocols crucially hinges on the *distributed state of the switching tables collectively satisfying and maintaining* certain properties that ensure deliverability of messages.

[1] This assumption is to keep the model simple; more complex matching rules such as matching with the maximal prefix, or allowing for priorities among potential prefixes may be viewed as practical optimizations.

Informally, the operation of model is: When a message pm appears on the input port ${}_AB$ such that S_B has an entry $\langle A, p\rangle \mapsto \{\langle C_i, p'_i\rangle\}$, then for each i, a message $p'_i m$ is placed on the output port B^{C_i}. Note that the switching tables at different ASs may be quite different.

▶ **Definition 1.** A message m arriving at an input port ${}_XA$ is called a *barb*, written $m@{}_XA\uparrow$, if $\mathsf{S}_A[X, p] = \emptyset$ for each prefix p of m.

Barbs are the *observables* of the data plane, which occur when a switch is unable to handle an incoming message. Barbs can form the basis for an operational account of the data plane's behavior. In this paper, instead, we provide a logical account of the behavior.

2.2 Axioms

The following axiom schema specify the behavior of edges and switches.

RT1. (Direct Communication)

$$e(A, B) \wedge m@A^B \;\Rightarrow\; \Diamond\,(m@{}_AB)$$

RT2. (Local Switching)

$$pm@{}_AB \wedge \langle C, p'\rangle \in \mathsf{S}_B[A, p] \;\Rightarrow\; \Diamond\,(p'm@B^C)$$

Axiom RT1 describes direct communication between ASs. If there is a direct edge from A to B, a message on output port A^B eventually appears on the input port ${}_AB$. Axiom RT2 expresses the lookup and switching capability of an AS. If a message with prefix p is at an input port of B and the switching table there indicates that a message with this prefix at that input port should have its prefix rewritten to p' and placed on output port, then the transformed message eventually appears at that port. Note that RT2 also covers any form of multi-recipient forwarding, such as multicast, since $\mathsf{S}_B[A, p]$ may have multiple elements.

These axiom schema capture a notion of *deliverability*, and apply to message transmission between ASs at *any* two levels in the protocol stack in the data plane. For simplicity of exposition, we assume that messages do not get lost or corrupted at ASs or at edges. Note that loss and corruption of messages may be explicitly simulated within the model by using suitably defined ASs. The model naturally permits message reordering. The model is completely distributed: ASs may operate concurrently and independent of one another, and even the operations on messages at the same/different ports of an AS may be concurrent.

We write $m@x \rightsquigarrow m'@y$ (read *relays-to*[2]) as convenient shorthand for $m@x \Rightarrow \Diamond\, m'@y$. For brevity, we write, e.g., $m@x \rightsquigarrow m'@y \rightsquigarrow m''@z$ for $m@x \Rightarrow \Diamond\, m'@y \;\wedge\; m'@y \Rightarrow \Diamond\, m''@z$. The relays-to relation helps define a number of well-known communication concepts.

Let A and B be ASs and $p \neq \epsilon$. We then define:

Name. Prefix p *is a name for* B *at* A if

$$\exists X, Y, Z: \quad \exists p' \neq \epsilon: \quad \forall m : pm@{}_XA \rightsquigarrow p'm@{}_YB \rightsquigarrow m@B^Z.$$

The name for an AS B at another AS A is a non-empty prefix that is removed when a message is transmitted from A to B. By default, names are local and their meaning is

[2] In [14], this relation was inductively defined using four axiom schema, two of which – reflexivity and transitivity — are implicit in temporal logic.

relative to the AS where they are interpreted. Note that p can be a name at A for multiple ASs, and that the ASs denoted by a name can vary with changes to switching tables. The condition $p' \neq \epsilon$ ensures that B is indeed the AS where the prefix or any residual of it is removed.

Address. Prefix p *is an address for* B *at* A if

$$\forall X : \exists Y, Z : \ \forall m : \quad pm@_{X}A \rightsquigarrow pm@_{Y}B \rightsquigarrow m@B^{Z}.$$

An address is a special kind of name whose meaning is independent of the input port, and which does not change along the path traversed by a message (though the ASs between A and B may add prefixes to pm). Note that p can be an address for multiple ASs.

Tunnel. There is a *tunnel from* A *to* B if

$$\exists W, X, Y, Z : \exists p, p' \neq \epsilon : \forall m : \quad m@_{W}A \rightsquigarrow pm@A^{X} \rightsquigarrow p'm@_{Y}B \rightsquigarrow m@B^{Z}$$

A tunnel from A to B takes a message at A, "wraps" it with a prefix, and delivers it to B where it is unwrapped. Tunnels provide a means for abstracting away the series of intermediate edges that establish the connection between two nodes. Let $tunnel({}_{W}A, B^{Z})$ denote the existence of a tunnel between A and B with entry and exit ports at W, Z.

Tunnels can be composed (by connecting the output port of the first to the input port of the second):

▶ **Lemma 2** (Tunnel Composition)**.** *Suppose* $tunnel({}_{W}A, B^{X})$ *and* $tunnel({}_{Y}B, C^{Z})$*, then (by linking* B^{X} *to* ${}_{Y}B$*),* $tunnel({}_{W}A, C^{Z})$*.*

2.3 Table Updates

Network events necessitate dynamic updates to switching tables. New entries can be added and existing entries modified or deleted. Updates to switching tables occur relatively infrequently and are atomic. Consequently, we may safely assume that at each point in the execution the tables are stable and well defined.

An update S_B^1 of S_B is *monotone* if for every $\langle C, p' \rangle \in \mathsf{S}_B[A, p]$, $\langle C, p' \rangle \in \mathsf{S}_B^1[A, p]$. The update S_B^1 *preserves a name for* A *at* C if the name is not disturbed by the update. Similarly, the update S_B^1 *preserves a tunnel from* A *to* C if this tunnel is not disturbed by the update.

▶ **Lemma 3.** *Monotone updates preserve names and tunnels.*

Example: Routing updates in SDNs

The occurrence of a barb in the data plane indicates that at some AS there is no switching rule for handling a message that arrived there. This may indicate an error in the protocol. However, if the tables have been properly configured (as presumed), the barb is an event that necessitates action on part of the control plane to update the switching tables. Such an event may occur, for instance, when a new device has been introduced into the network, and messages addressed to it need to be forwarded. Typically, the control plane responds by placing output messages at some output ports and performing a *monotone update* to the switching table of that AS. Lemma 3 implies that such SDN updates preserve names and tunnels.

▶ **Corollary 4.** *Routing updates in SDNs preserve names and tunnels.*

Routing updates, being monotone, preserve deliverability of messages. Note though, that while they are intended to remove the barbs that triggered them, they may introduce new barbs at other nodes.

Further, they may introduce forwarding cycles, which cause messages to traverse cycles. A message m is said to traverse a *cycle* if for some A, p, p', p'', X, Y, Z, $pm@_x A \rightsquigarrow p'm@A^Y \rightsquigarrow p''m@_z A$. Note that a message traversing a cycle does not necessarily imply that it will remain in that loop indefinitely — by returning to A at a different input port, and/or with a different prefix p'', it may be switched differently the second time around. Another way that forwarding cycles may be broken is via updates to a switching table.

3 IP Mobility

Changes in the network topology may disrupt the "relays-to" relation, resulting in the invalidation of certain names or a change in their meaning. This may be repaired by updates to one or more switching tables. A good example of such updates occurs in IPv6 mobility [20], the essence of which we model here.

3.1 Tunnel maintenance

Recall that tunnels allow us to abstract from the details of all nodes and direct edges in the network graph. Thus we can focus on only IP layer nodes and tunnels between them. For each node A, let n_A be its universal address (a "well-known address" in networks parlance). We write $tunnel^{(n_B)}(X, Y)$ to indicate that at AS X the switching tables are configured to tunnel any message with prefix n_B to Y.

IPv6 support for mobility is based on a proxy architecture, where a mobile node B is associated with

- its *identifying* IP address n_B;
- its home router, denoted $H(B)$, which never changes;
- its current location, denoted $h(B)$, which may change over time. At any point in time, B is "hosted by" at most one router.

We assume that if a message intended for B reaches $h(B)$, it will eventually be delivered to B.

$$(\textbf{mIP}) \qquad n_B m@_? h(B) \Rightarrow \Diamond(m@B^?)$$

In a stable state of the network, a sender sends a message to a mobile node B by either

1. tunneling it to B's home router $H(B)$ (B's default location); (a) if B is at home ($h(B) = H(B)$), the message will be delivered to B. (b) if B is away ($h(B) \neq H(B)$), $H(B)$ tunnels it to $h(B)$, which $H(B)$ presumably knows.
2. alternatively, if the sender knows $h(B)$, it may tunnel the message to $h(B)$.

In all the above cases, Lemma 2 and (mIP) ensure that the message will be delivered to B.

However, when B moves, we cannot assume that $h(B)$ will be known to nodes, in particular to $H(B)$, wishing to communicate with it. With a small number of *control messages* that eventually lead to (re)establishing correct forwarding tables, the IPv6 mobility protocol achieves correct forwarding of messages. We focus here only on the effects of the control messages on the switching tables. It is the control protocol's responsibility to ensure that table update messages received out-of-order are ignored (this can be achieved using sequence numbers).

Let B be any node. We assume that initially for every $X \neq H(B)$, $tunnel^{(n_B)}(X, H(B))$. Moreover, at any time, if nodes X and Y are not in $\{H(B), h(B)\}$ and $tunnel^{(n_B)}(Y, X)$

exists, this tunnel can be removed and replaced by $tunnel^{(n_B)}(Y, H(B))$ (a "time-out reset"). Once registered at $h(B)$, B (or $h(B)$ on its behalf) may send control messages to any correspondent node A not necessarily involved in the move, telling A of its current location, in which case A *may* update to $tunnel^{(n_B)}(Y, h(B))$. The only possible other n_B-tunnel changes are triggered by B's moves, with the associated mandatory updates summarized in Fig. 2.

Move	**Remove Tunnels**	**Create Tunnel**
$H(B)$ to F_1	$tunnel^{(n_B)}(F_1, X)$, $tunnel^{(n_B)}(H(B), X)$ $X \neq F_1$	$tunnel^{(n_B)}(H(B), F_1)$
F_1 to $H(B)$	$tunnel^{(n_B)}(F_1, X)$ $X \neq H(B)$ $tunnel^{(n_B)}(H(B), X)$	$tunnel^{(n_B)}(F_1, H(B))$
F_1 to F_2	$tunnel^{(n_B)}(F_2, X)$ $tunnel^{(n_B)}(H(B), F_1)$	$tunnel^{(n_B)}(H(B), F_2)$ exactly one of $tunnel^{(n_B)}(F_1, F_2)$ or $tunnel^{(n_B)}(F_1, H(B))$

Figure 2 Tunnel updating on moves

The IPv6 protocol implementation is assumed to ensure that *control messages will get delivered*, especially to the home network, and consequently that there are no ways of creating $tunnel^{(n_B)}(X, Y)$ other than those in Fig. 2, or by "timing out". We assume that the removal and creation of tunnels at a single AS happens atomically. Observe that the updates mentioned are *not monotonic*.

3.2 Properties

By induction on the tunnel update moves, we can prove the following properties:

Tunnel Endpoint. Tunnels only end at previously visited hosts.

$$\forall X.\ tunnel^{(n_B)}(X, Y) \ \Rightarrow\ (Y = H(B) \ \vee\ \diamondminus(Y = h(B))$$

No Wandering. Messages (at tunnel end-points) can only visit their sender, receiver, the receiver's home or any node that hosted it: For every message m from sender A to receiver B,

$$\forall X. n_B m @_? X \ \Rightarrow\ (X = A \ \vee\ X = B \ \vee\ X = H(B) \ \vee\ \diamondminus(X = h(B)))$$

Note that all tunnels created in response to control messages involve a router that has been visited earlier.

No Forwarding Cycles. Tunnels do not form a true cycle[3]. Let $(tunnel^*)^{(n_B)}$ be the transitive closure of $tunnel^{(n_B)}$ as per Lemma 2.

$$\forall X :\ \neg((tunnel^*)^{(n_B)}(X, X))$$

▶ **Lemma 5.** *If the mobile node settles at its home, eventually there is permanent tunnel from all other nodes to its home.*

$$\Box(h(B) = H(B)) \Rightarrow \forall Y \neq H(B).\ \Diamond\, \Box\ tunnel^{(n_B)}(Y, H(B))$$

[3] Actually, this property can be weakened to one requiring only that any such forwarding cycle will eventually get broken.

Proof. Let σ be a computation satisfying $\Box(h(B) = H(B))$ and let $Y \notin \{B, H(B)\}$. Initially $tunnel^{(n_B)}(Y, H(B))$; from the "time-out" action or if B moves, it follows from the rules of Fig. 2 that there exists a suffix σ' of σ where $tunnel^{(n_B)}(Y, H(B))$. Now, since B remains at $H(B)$, Y creates no other tunnel for n_B. Thus, σ' satisfies $\Box(tunnel^{(n_B)}(Y, H(B)))$. ◀

Of course, there may be points in a computation where a n_B-tunnel does not exist to B's location. However, by the eventual delivery of control messages, or by time-out, such a tunnel will be constructed. Observe that the model does not place any buffer-space limitations or delivery-order restrictions on messages, so in an implementation, their deliverability relies on buffering them until a tunnel is (re-)constructed. The next lemma addresses n_B-tunnels when B settles at a non-home node.

▶ **Lemma 6.** *If the mobile node settles at a place, there is a tunnel from home to that place, and a tunnel from every previously visited node to either home or to that permanent host. The proof is similar to that of Lemma 5.*

$$\forall X, Y.\ \Box(h(B) = X \ \wedge\ X \neq H(B) \ \wedge\ Y \notin \{h(B), H(B)\}) \ \Rightarrow \\ \Diamond\Box\left(\begin{array}{l} tunnel^{(n_B)}(H(B), h(B)) \\ \ \wedge\ (tunnel^{(n_B)}(Y, h(B)) \ \vee\ tunnel^{(n_B)}(Y, H(B))) \end{array}\right)$$

Theorem 7 establishes that if a node remains at a single host for sufficiently long, then it will receive every message sent to it.

▶ **Theorem 7.** *If a mobile node settles, any message addressed to it eventually gets delivered.*

$$\Box(h(B) = X) \ \Rightarrow\ (n_B m @_? A \ \Rightarrow \Diamond(m @ B^?))$$

Proof. Let σ be a computation satisfying $\Box(h(B) = X)$, and assume $n_B m @_? A$ holds. From Lemmas 5 and 6 it follows that there exists a suffix σ' of σ such that $\sigma' \models \Box(tunnel^{(n_B)}(Y, X)$ for every $Y \notin \{X, H(B), B\}$. The No Wandering property establishes that at the initial state of σ', m has either already reached its destination B or is at some other node. In the latter case, once in σ', m will be forwarded through the permanent tunnel to X, and eventually reach B by (mIP). ◀

The theorem does not (incorrectly) claim that messages cannot go into a cycle. The No Forwarding Cycles property guarantees it does not stay in a cycle if the mobile node settles (rendered as "ever after" in LTL) at a host. However, if a mobile node B does not stay "long enough" at any node, a message may forever keep chasing it without ever catching up with it.

This theorem has been validated for a finite version of the model using TLV.

4 Security primitives: the TOR example

In addition to routing, security is an important issue in network protocols. Encryption and decryption operations on messages motivate an extension to the model of Section 2 by allowing switches to perform cryptographic and other functional transformations on messages. We illustrate the extension by presenting the essence of the TOR [8] architecture, which is designed to support anonymous communication. Formal notions of anonymity have always been among the hardest security properties to capture [13].

We modify the Local Switching axiom to

RT2′ $\quad pm @_A B \wedge \langle C, p', f_{ABC}\rangle \in \mathsf{S}_B[A, p] \ \Rightarrow\ \Diamond\,(p'm' @ B^C) \quad (m' = f_{ABC}(p, m))$

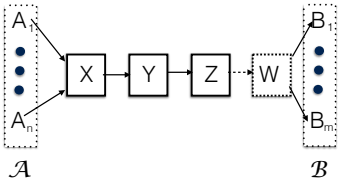

Figure 3 Schematic View of TOR

S_B, the switching table at AS B, now contains mappings of the form $\langle A, p\rangle \mapsto \{\langle C, p', f_{ABC}\rangle\}$, where C and p' are as before, and f_{ABC} is a function that transforms messages: a message placed on an output port is a functional transformation of the incoming message, where the function may depend on the switch B and on the input and output ports. This generalization permits expressing a variety of operations, such as hashing, route recording, etc.

4.1 TOR

TOR (The Onion Router) is a low-latency anonymous communication service. To send an anonymous message, a client chooses at least three intermediate routers to be used as a chain of relays to the recipient. The client establishes shared secret session keys with each of the intermediate TOR nodes, and encrypts the message successively with these keys in the inverse order with respect to the routers through which the message will pass. The TOR routers are only aware of their successor and predecessor nodes in the relay chain. The protocol is designed to ensure that none of the intermediate routers are aware of both the sender and receiver of the message.

For example, suppose sender $A \in \mathcal{A}$, where $\mathcal{A}$ is the set of potential senders, wishes to sends a message m to receiver $B \in \mathcal{B}$ ($\mathcal{B}$ is the set of potential destinations) via TOR nodes $X; Y; Z$. To simplify the presentation, we connect Z to an extra (non-TOR) "fan-out" router W to communicate with the actual recipient of the message. (See Fig. 3, where for simplifying the exposition, we depict the tunnels between the routers as direct links).

Client node A sends to the first TOR router X the message

$$m_A = n_X\, E_X(n_Y\, E_Y(n_Z\, \underbrace{E_Z(n_B m_B)}_{m_Z})) \quad \underbrace{\qquad\qquad}_{m_Y} \quad \underbrace{\qquad\qquad\qquad}_{m_Z}$$

where $E_i(m')$ is the cipher text obtained by encrypting m' with the key shared between A and i; each n_i is the "well-known" address of i. Upon receipt of a message each TOR node decrypts it using the shared key, and transmits the result to the next node in the chain. Here anonymity implies that neither Y nor Z is able to learn the identity of sender A, and neither X nor Y can learn the identity of receiver B from a set of potential senders and receivers of the message.

The TOR protocol involves a very special version of switching, where the switching decisions are *not* made based on the prefixes of the messages but rather are effected based on the content of the messages. Moreover, the function f_{ABC} at TOR nodes depends only on B (in fact on the key shared by B with the TOR client): for every A' and C', $f_{A'BC'} = f_{ABC}$. We therefore refer to the switch transformation function as f_B.

Suppose X is the ingress TOR router, with input messages from possible sources $A_1, \ldots A_k$. The switching table at the ingress router X has entries of the form:

$$\langle A_i, n_X \rangle \mapsto \langle Y, \epsilon, E_X^{-1} \rangle$$

where E_X^{-1} is the decryption function using the key shared with the client and X. When X receives the message $n_X E_X(n_Y m_Y)$, it places $n_Y m_Y$ on the output port to Y. Note that since X does not have the keys shared between A and Y, it cannot learn that the message is intended for B. At the intermediate TOR router Y, with predecessor X and successor Y, the switching table has entries of the form:

$$\langle X, n_Y \rangle \mapsto \langle Z, \epsilon, E_Y^{-1} \rangle$$

When Y receives the message $n_Y m_Z$, it places $n_Z m_Z$ on the output port to Z. Since Y does not have the keys shared between A, Z, it cannot learn that the message is intended for B, nor does it know the source A of this message, since the message was from the input port from X. At the TOR router Z, with predecessor Y and a single egress router W which connects to nodes in $\mathcal{B}$, the switching table is:

$$\langle Y, n_Z \rangle \mapsto \langle W, \epsilon, E_Z^{-1} \rangle$$

When Z receives the message $n_Z E_Z(n_B m_B)$, it places $n_B m_B$ on the output port to W. Router Z does not have any indication of the source A of this message, since all messages were on the input port from Y.

4.2 Proving Some TOR properties

Our main technique for proving correctness of TOR is derived from the *strand* formalism in [11], which supports a representation of the execution of a protocol in terms of local views of principals and the messages exchanged in that execution.

Security protocols are cast as cryptoalgebras in [11], where to decrypt an encrypted term $t = E_P(m)$, an adversary needs to have received prior messages from which E_P can be derived. Though the Dolev-Yao model [9] abstracts from more realistic assumptions regarding message structure and cryptographic schemes addressed by [11] (e.g., bounds on message length, the particular encryption/decryption functions, and ability to guess keys and ciphers), it is considered a standard adversarial model for formal security analysis. For the purpose of this section, we restrict to the Dolev-Yao model. Recall that a Dolev-Yao adversary can only encrypt/decrypt messages whose keys it possesses.

Following terminology roughly influenced by formal reasoning about knowledge (see, e.g., [10, 12]), let $K_i(z)$ denote that a participant i knows the value of a variable z. Initially, every participant P knows the names of all participants and the encryption/decryption keys to which it is privy, i.e, $K_P(n_Q)$ for every participant P and Q, and $K_P(E_P)$. The adversary may know additional keys, but we assume that it knows at most two of the three keys A shares with X, Y, and Z. During the course of the protocol, the adversary may decrypt any messages whose encryption keys it knows, as well as send messages that may be encrypted with keys known to it. Moreover, for every honest participant P, the encryption key E_P can never be derived, that is, honest participants never send any information that reveals their keys. In particular, if the adversary does not initially know E_i and cannot derive it from any message it had seen, then it cannot decrypt any message encrypted with E_i.

We assume that every participant knows the sets $\mathcal{A}$ and $\mathcal{B}$. For TOR message m, let $src(m) \in \mathcal{A}$ be its source and $dest(m) \in \mathcal{B}$ be its intended destination. Thus, when a TOR

message m is received, for every participant P, $K_P(src(m) \in \mathcal{A})$ and $K_{adv}(dest(m) \in \mathcal{B})$. Moreover, for every honest participant P, $\neg K_{adv}(E_P)$.

Our goal is to show that no participant, but for the source and destination, get to learn both source and destination of a TOR message even if adversary knows all but one key of an intermediate node.

▶ **Theorem 8.** *For every TOR message m_A and every participant $P \neq A, B$,*

$$\Box\, \neg(K_P(src(m_A)) \ \wedge \ K_P(dest(m_A)))$$

even if two of X, Y, and Z are compromised.

Proof Outline. We present the most commonly studied case, where X and Z are compromised, but not Y. Initially $K_{adv}(E_X)$ and $K_{adv}(E_Z)$. When $m_A = n_X m_Y @_? X$, the adversary can learn $src(m_A)$. We have to show that it never learns $dest(m_A)$. From our assumption it follows that $\Box\ \neg K_{adv}(E_Y)$. In the protocol, X sends the message m_Y, thus eventually $m_X = n_Y m_Y @_? Y$. While the adversary may know $m_X @_? Y$, from our assumptions it follows that it cannot decipher m_Y, and hence, cannot associate the event $m_Z = n_Z m_Z @ Y^?$ with the message m_A. That is, it cannot derive that $m_Z = n_Y E_Y(m_Z)$. From here the adversary cannot determine that $m_Z @ Y^?$ implies that $\diamondsuit\, m_A @_? X$, from which the claim follows. ◀

5 Conclusions

The disparity between the structure and behavior of computer networks and the abstract formal models used for modeling and verifying protocols motivated us to present a modular logical account of the data plane of networks. By separating the data plane from control protocol descriptions, we hope that formal analyses of various control protocols (routing, security, etc.) can be performed independently of modeling the details of message delivery within a given abstract framework. Our framework provides a general model of message relay at multiple layers of the network stack, thereby supporting formal analyses using appropriate abstractions that accurately capture real network behavior. By ensuring that the data plane model is presented in a manner that naturally admits refinements, both in temporal logical specifications and in the hierarchical structure, we claim that the resultant framework can support more robust proofs of correctness. In this paper, we have accounted for changes in the routing and forwarding topologies, and can specify and model network structure and behavior that changes over time.

To our knowledge there are few formal models of networks, and fewer that account for dynamic changes to network structure. The example we chose to illustrate our approach, i.e., modeling IPv6 mobility shows how one can construct concise and abstract proofs of correctness of a protocol, without modeling too many operational details of the protocol and the network. Our analysis is minimalist in that it does not mistakenly claim stronger properties than are necessary. For example, some formal analyses of IPv6 mobility claimed (incorrectly) that the protocol does not allow data messages traverse cycles, or that forwarding cycles are never created. The correctness of the protocol, however, hinges on a far weaker property: that any temporarily created message or forwarding cycles will be broken ("sooner or later" – though this argument is made without explicitly reasoning about time).

The other generalization made in this paper is to endow switches with the ability to perform functional transformations on messages. This supports the modeling of a large variety of security protocols within the data plane. We are unaware of any earlier formal logical account of the correctness of the TOR protocol in providing anonymity.

In this paper, the two extensions — dealing with dynamic changes in the forwarding topologies and providing switches with the ability to transform messages — have been treated orthogonally. We believe that there are a large number of time-dependent security protocols that can fruitfully be explored in a combination of these extensions, for instance, whether the TOR protocol provides *perfect forward secrecy* or *perfect forward anonymity*. Indeed, the notion of anonymity merits further investigation. We expect that any serious formal treatment of anonymity that respects the dynamics of networks will be based on notions of indistinguishability (of names, messages, router behavior, local states, etc.), which we believe can be characterized in a succinct yet robust way within our axiomatic basis for communication.

Acknowledgements The authors want to thank the anonymous referees for their valuable suggestions.

References

1. Martín Abadi and Cédric Fournet. Private authentication. *Theor. Comput. Sci.*, 322(3):427–476, 2004.
2. Roberto M. Amadio and Sanjiva Prasad. Modelling IP mobility. In *CONCUR '98: Concurrency Theory, 9th International Conference, Nice, France, September 8-11, 1998, Proceedings*, pages 301–316, 1998.
3. Alessandro Armando, David A. Basin, Yohan Boichut, Yannick Chevalier, Luca Compagna, Jorge Cuéllar, Paul Hankes Drielsma, Pierre-Cyrille Héam, Olga Kouchnarenko, Jacopo Mantovani, Sebastian Mödersheim, David von Oheimb, Michaël Rusinowitch, Judson Santiago, Mathieu Turuani, Luca Viganò, and Laurent Vigneron. The AVISPA tool for the automated validation of internet security protocols and applications. In *Computer Aided Verification, 17th International Conference, CAV 2005, Edinburgh, Scotland, UK, July 6-10, 2005, Proceedings*, pages 281–285, 2005.
4. Alessandro Armando, David A. Basin, Jorge Cuéllar, Michaël Rusinowitch, and Luca Viganò. Automated reasoning for security protocol analysis. *J. Autom. Reasoning*, 36(1-2):1–3, 2006.
5. David Clark. The design philosophy of the DARPA internet protocols. In *Proceedings of SIGCOMM '88*, pages 106–114, 8 1988.
6. Rance Cleaveland, Joachim Parrow, and Bernhard Steffen. The concurrency workbench. In *Automatic Verification Methods for Finite State Systems, International Workshop, Grenoble, France, June 12-14, 1989, Proceedings*, pages 24–37, 1989.
7. Giorgio Delzanno and Pierre Ganty. Automatic verification of time sensitive cryptographic protocols. In *Tools and Algorithms for the Construction and Analysis of Systems, 10th International Conference, TACAS 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29 - April 2, 2004, Proceedings*, pages 342–356, 2004.
8. Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium, August 9-13, 2004, San Diego, CA, USA*, pages 303–320, 2004.
9. Danny Dolev and Andrew Chi-Chih Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–207, 1983.
10. Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. Knowledge-based programs. In *Proceedings of the Fourteenth Annual ACM Symposium on Principles of Distributed Computing, Ottawa, Ontario, Canada, August 20-23, 1995*, pages 153–163, 1995.

11 Joshua D. Guttman, F. Javier Thayer, and Lenore D. Zuck. The faithfulness of abstract protocol analysis: Message authentication. *Journal of Computer Security*, 12(6):865–891, 2004.

12 Joseph Y. Halpern and Lenore D. Zuck. A little knowledge goes a long way: Knowledge-based derivations and correctness proofs for a family of protocols. *J. ACM*, 39(3):449–478, July 1992.

13 Dominic J. D. Hughes and Vitaly Shmatikov. Information hiding, anonymity and privacy: a modular approach. *Journal of Computer Security*, 12(1):3–36, 2004.

14 Martin Karsten, S. Keshav, Sanjiva Prasad, and Mirza Beg. An axiomatic basis for communication. In *Proceedings of the ACM SIGCOMM 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Kyoto, Japan, August 27-31, 2007*, pages 217–228, 2007.

15 Diego Kreutz, Fernando M. V. Ramos, Paulo Jorge Esteves Veríssimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2015.

16 Leslie Lamport. What good is temporal logic? In *IFIP Congress*, pages 657–668, 1983.

17 Orna Lichtenstein, Amir Pnueli, and Lenore D. Zuck. The glory of the past. In *Proceedings of the Conference on Logic of Programs*, pages 196–218. Springer-Verlag, 1985.

18 Robin Milner. *Communication and concurrency*. PHI Series in computer science. Prentice Hall, 1989.

19 Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, I. *Inf. Comput.*, 100(1):1–40, 1992.

20 Charles E. Perkins and David B. Johnson. Mobility support in ipv6. In *Proceedings of the 2nd Annual International Conference on Mobile Computing and Networking*, MobiCom '96, pages 27–37, New York, NY, USA, 1996. ACM.

21 Sanjiva Prasad. Abstract switches: A distributed model of communication and computation. In *Perspectives in Concurrency Theory*. CRC Press, 2009.

22 Davide Sangiorgi and David Walker. On barbed equivalences in pi-calculus. In *CONCUR 2001 - Concurrency Theory, 12th International Conference, Aalborg, Denmark, August 20-25, 2001, Proceedings*, pages 292–304, 2001.