Elements of Application-layer Internetworking for Adaptive Self-organizing Networks

Jörg Liebeherr Majid Valipour Tony Yu Zhao

***Abstract*—By providing a global infrastructure for information exchange, the Internet has had a transformational impact on society at large. At the same time, a number of indicators, among them an observed ossification phenomenon, raise concerns about the ability of the Internet to support future communication needs and stipulate interest in alternative methods for internetworking. This paper considers an internetworking approach based on self-organizing application-layer networks. Rather than building a single global infrastructure that provides universal access, these networks take advantage of a diverse collection of network infrastructures to interconnect users and devices participating in a networked application. In this paper we discuss aspects such as scalability, ability to adapt after disruptions, heterogeneous substrate networks, distributed security, and dynamically created network services. The discussions are supported by numerous measurement experiments.**

***Index Terms*—Internetworking, adaptive networks, application-layer networks, multi-substrate networks, self-organizing networks.**

I. Introduction

The origins of the Internet resulted from a vision of interconnecting decentralized independent networks using arbitrary designs and protocols [53]. This concept of *internetworking* or *interworking* was achieved by providing a uniform abstraction, that of a *subnet*, to represent any lower-layer network, such as a single point-to-point link, a local area network, or a wide-area switched network. The protocol architecture designed to realize the interconnection of subnets turned out to be capable of supporting a global communication infrastructure connecting more than a billion devices.

While the Internet has been celebrated as one of the premier engineering achievements of the 20$^{\text{th}}$ century [28], its sheer scale, as well as trends in technology and business, have created a set of formidable challenges.

- *Ossification:* Starting in the early 2000s, it has been noticed that the ability to introduce new services or incorporate major technological upgrades in the Internet infrastructure has been largely lost. This phenomenon, referred to as *Internet ossification* [3], [69], [84], inhibits the ability to meet the demands of novel applications and to face emerging security threats. For example, the new version of the Internet protocol, IPv6, was specified in the 1990s [34], yet it still accounts for only about 2% of the traffic at one of the major exchange points for Internet traffic [4].
- *Regulatory constraints:* Some innovations in Internet technology, among them the ability to offer different types of service to network traffic, have raised concerns of being used for purposes that do not benefit users. This, in turn, has attracted the attention of regulatory agencies. An example of regulatory policy is *net neutrality*, which refers to a directive to network service providers to treat all Internet traffic equally, thus disallowing any service differentiation [29].
- *Proprietary infrastructures:* Many large content providers, such as Google, Facebook, and Netflix, now operate proprietary network infrastructures that deliver services close to customer networks, thereby bypassing most of the Internet infrastructure [3], [42]. Other proprietary infrastructures offer wide-area wireless services for low-power low-bandwidth data services [92]. Operating outside the scope of the Internet, the commercial interests of these providers are no longer a driving factor for advancing the Internet.
- *End-to-end principle:* The end-to-end principle [18], [75], which is central to the design philosophy of the Internet, views the collection of subnets as a transparent forwarding medium without considering location or resource availability when forwarding data in the network. Doing so can create counterintuitive situations, where data between co-located devices, e.g., a smartphone and a laptop in the same office, traverses multiple remote networks, even though the devices have multiple modalities available to exchange information directly.
- *Global addresses:* A prerequisite for using the communication services of the Internet is the allocation of a unique – possibly shared – global Internet address. Once a device has acquired such an address, it can send to and receive from any other global address. A drawback of using global addresses is that it creates security risks, whereby any remote node with a global address has the opportunity to access a device and exploit vulnerabilities.
- *Internet of Things:* The extension of Internet connectivity to network-enabled sensors, actuators, and other physical devices, referred to as the *Internet of Things* (IoT) [9], and the resulting explosive increase of Internet enabled devices possibly presents the most significant and immediate set of challenges for the Internet architecture. Among them are the large projected number of network enabled devices [49], limited computing capabilities of embedded devices, the narrower application-specific scope of computational tasks, delay-sensitivity of transmitted data, and security. Standardization efforts for network-enabled

J. Liebeherr is with the Department of Electrical and Computer Engineering, University of Toronto, M. Valipour is with Google, Waterloo, and T. Y. Zhao is with Ethoca Financial Services, Toronto. The work of M. Valipour and T. Zhao was done when they were with the University of Toronto.

devices abound [77], [79] but leave the internetworking framework of the current Internet largely in place [65].

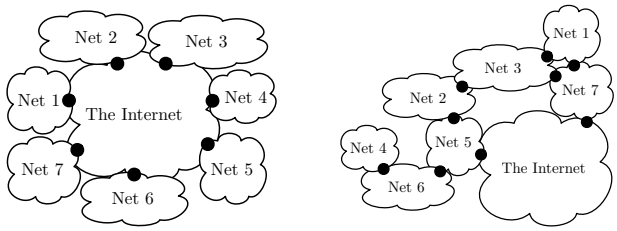
With these considerations it is plausible that future communication services will be provided by a variety of public and private network infrastructures. While the Internet will continue to be available as a global infrastructure, it will be supplemented by special purpose networks that serve applications with demand for assurances on bandwidth availability, security, low latency, high energy efficiency, and others. In such a setting, the need arises to interconnect the available network infrastructures, which, in turn, stipulates the exploration of new approaches to internetworking.

In this paper, we contemplate internetworking through self-organizing application-layer networks, which we refer to as *application networks*. Self-organizing means that devices use all communication modalities available to them to detect and establish connectivity with other devices to form a network. In contrast to centrally planned and managed network infrastructures, self-organizing networks are formed and operated in a fully distributed manner, and do not rely on centralized mechanisms for network formation, management, or operation [8]. They may exist only for a limited time to perform a particular task or provide connectivity to a group of devices. Their topology can be highly dynamic as nodes may enter or leave the network at any time. Furthermore, network and application services in self-organized networks are provided by the nodes themselves. In particular, all nodes of a network participate in relaying data between sources and destinations of information. As a consequence, the canonical distinction in communication networks between endsystems as producers and consumers of data, and relay switches that forward data between endsystems falls away. Self-organization can be deployed on top of an existing network infrastructure, as is the case for Internet peer-to-peer networks [62]. Alternatively, network nodes can self-organize to build their own network infrastructure, as in the case of wireless sensor networks [1] and mobile ad-hoc networks [41].

Application-layer network protocols are easy to deploy, since there is no need for compatibility at the operating system or hardware level. There is no need for universal deployments of protocols and services, since the scope of an application network is limited to the devices participating in the same network. Moreover, since application networks operate independently of each other, each application network can be adapted to meet the requirements of a given application scenario. In particular, application networks can be customized to the constraints of low-power embedded devices and micro controllers in IoT deployments. The latter can help with resolving interoperability issues between IoT devices [46].

With application networks, each instantiation of a networked application may result in the formation of a separate network. All nodes of an application network participate in the same application, and devices may participate in an arbitrary number of such networks. Multiple application networks involving multiple overlapping groups of users may co-exist on the same underlying infrastructure.

The application networks considered in this paper evolved from self-organizing peer-to-peer networks, which emerged in the early 2000s for deploying application-layer services for multicast delivery [26], distributed lookup [80], and failure resilient delivery [5]. These networks generally assume that the Internet provides a permanently available and universally accessible substrate network. More recently, researchers have considered constructing peer-to-peer networks over multiple heterogeneous substrate networks, where any data link, network layer, or even application network can constitute a separate substrate network [59], [67]. The goal of these efforts is to achieve an interconnection of applications running on mobile and stationary endsystems and using a diverse set of networking modalities, that may include the Internet but do not require permanent access to its infrastructure.



(a) Internet as the central network. (b) No central network.

Fig. 1. Interconnected collection of substrate networks.

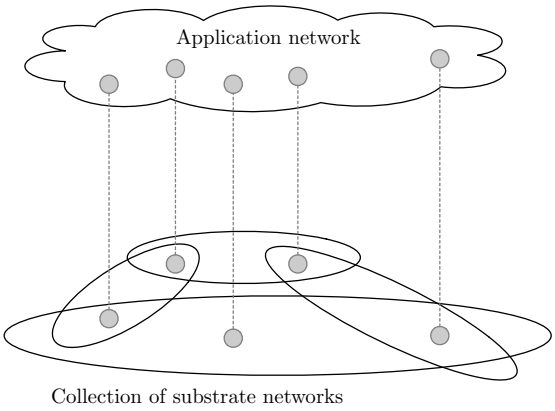


Fig. 2. Multi-substrate application network.

We refer to a collection of multiple substrate networks as a *multi-substrate network*. Examples of substrate networks are sensor networks, multi-hop vehicular networks, mobile ad-hoc networks, the Internet, and Internet-protocol based private networks. The interconnection of nodes with attachment points in different substrate networks by an application network will be referred to as *application-layer internetworking*. A common view for a connected collection of substrate networks is that all substrate networks are connected to the Internet, leading to a view of connectivity where the Internet is the central network, as illustrated in Fig. 1(a). Here, all substrate networks connect to the Internet, but not to each other. The assumption of permanent connectivity to the Internet can be limiting in environments with mobile nodes, intermittent or unavailable Internet connectivity, or alternative network infrastructures. This leads to a network model where the Internet is connected

to some but not all substrate networks, as illustrated in Fig. 1(b). Fig. 2 illustrates an application network in a multi-substrate setting, where nodes (drawn as small circles) located in a collection of substrate networks form an application network. The application network provides a network view to users, where members can exchange information without regard to their location in substrate networks.

Multi-substrate application networks impose numerous challenges with respect to addressing, discovery of nodes, network topology maintenance, and security. The challenges increase with the size of a network, the mobility of nodes, and the number of substrate networks. The objective of this paper is to survey central problem areas in multi-substrate application networks, and evaluate solutions approaches to some of the problems. Among the topics that are highlighted are the scalability and agility of protocols for application networks, the ability to support different types of substrate networks, dissemination of address information, distributed security, and adaptation of network services to the needs of applications. For each studied problem area, we include measurements of point solutions. The measurements are done on a Java-based software system, which has been used by our research group to investigate and evaluate concepts in single- and multi-substrate application networks [55], [56].

Internetworking between heterogeneous substrate networks has been considered before. Approaches to virtualize substrate networks often rely on network-layer encapsulation ('tunneling'), e.g., [11], [87]. Network architectures that relax the assumption of permanent connectivity to the Internet [27], [30] address similar problems as this paper in that they do not assume a hierarchy of substrate networks or a central entity. Concepts of a non-centralized architecture are studied in [15], [50], however, without addressing the computation of paths across multiple networks. Pathlet routing [43] proposes a packet forwarding method for arbitrarily connected networks. We note that there are alternatives to the distributed self-organizing approach to application networks considered in this paper. For example, the construction and operation of an application network can be supported by centralized services that can be accessed by some or all of its members. Further, application networks can be aligned with a software defined network architecture [52], whereby centralized controllers coordinate routing paths in the application network. Also, support for the creation and operation of application networks can be provided by packet switches a the data link or network layer.

The remainder of this paper is structured as follows. In Sec. II we address the topology formation in application networks and discuss measurements of topology formations for different types of substrate networks. In Sec. III we discuss solutions to disseminate address information of nodes in a multi-substrate setting. In Sec. IV we consider distributed solutions to data confidentiality and integrity in an application network. In Sec. V we present the design for dynamic additions of network services in application networks. We present conclusions in Sec. VI.

II. Building Application Networks

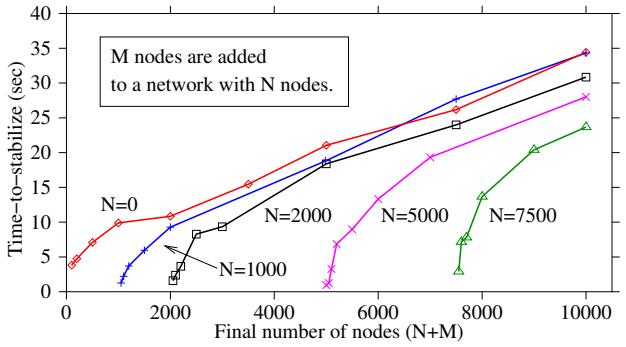
In this section, we discuss building blocks of application networks that can operate over a variety of different substrate networks. We refer to an entity of an application network as a *node*. A node is an application program that is running on a device. Each device can have arbitrarily many nodes that each participate in a different application network. We assume a message-passing mode of communication, which either extends message-based services in lower layer substrate networks, e.g., a point-to-point data link or a packet-switched network, or which inserts message boundaries in stream-oriented substrate networks, e.g., a TCP or Bluetooth RFCOMM connection.

A. Protocols to Maintain the Network Topology

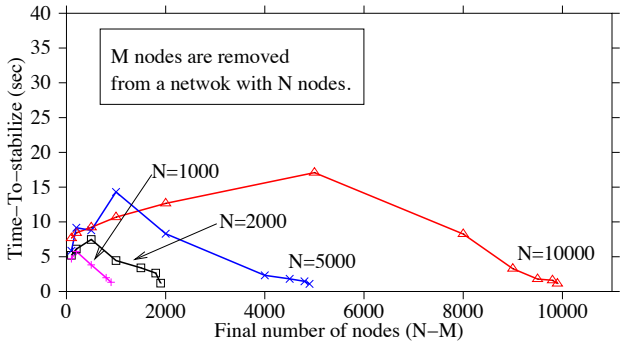
Application networks require protocol mechanisms to discover other nodes, to join and leave the network, and to exchange messages. Each application network is viewed as a collection of nodes that share a common network identifier, which is associated with the configuration of the network, e.g., its network topology, its security properties, and the types of supported substrate networks. An application network is created implicitly by the first node that joins the network with a specific network identifier. If the application network becomes partitioned, e.g., due to communication failure or mobility of nodes, each partition operates as an independent network with identical identifiers. Also, a newly created node that has not yet discovered other nodes of the topology is considered a separate partition. When nodes in different partitions discover each other, the partitions merge to form a single application network.

The nodes of an application network cooperate to establish a network topology graph. Application data is then routed along the edges of the graph. Establishing and maintaining a network topology involves (1) a discovery process by which a node not connected to an application network can find nodes of the network; (2) a neighbor selection method by which a node determines the subset of reachable nodes that become its neighbors in the network topology; and (3) a routing protocol that determines the forwarding paths for application data in the application network.

The topology of an application network is either *structured* or *unstructured*. A structured topology refers to a network graph, where the set of neighbors of each node and the routing tables for forwarding data in the network are determined by node identifiers. Examples of structured topologies include rings [19], hypercubes [22], distributed hash tables (DHTs) [80] and Delaunay triangulation [58]. In unstructured application networks, a distributed routing protocol is needed to set up forwarding tables at nodes that establish paths between nodes [26]. To maintain the neighborhood relationships in a topology, nodes periodically exchange control messages with their neighbors. The time interval between the transmission of successive control messages, referred to as the *heartbeat time*, determines the responsiveness and rate of convergence after a change of the network topology.

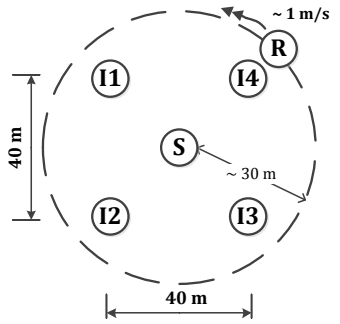


(a) Joining a network.

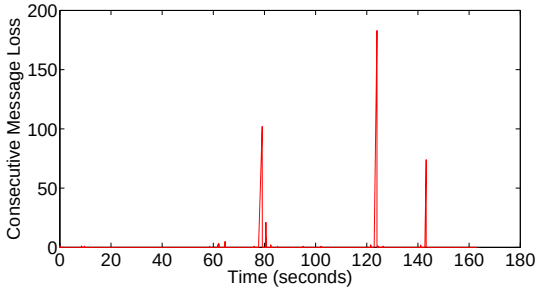


(b) Leaving a network.

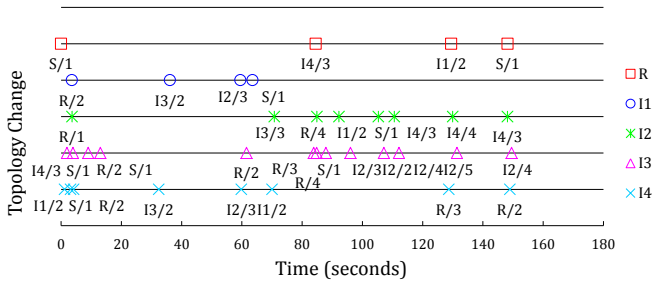
Fig. 3. Time period until the network topology stabilizes. (a) M nodes are added to an existing network with N nodes. (b) M nodes instantly depart a network with N nodes [58].



(a) Location of nodes.



(b) Consecutive losses.



(c) Topology changes.

Fig. 4. Mobility experiment.

B. Measurement Experiments

We next present a set of experiments that measure the construction and maintenance of application network topologies. The first experiment shows that application-layer protocols can support large-scale network topologies and adapt quickly to major disruptions.

Experiment 1 (Scalability): The experiment, from [58], measures the elapsed time until a set of nodes forms a stable network topology. The topology is a Delaunay triangulation [78]. In addition to the ability to accommodate concurrent changes to the topology, a Delaunay triangulation has a well defined stability criterion (see [58, Sec. II-C]). Each node is identified by (x, y) coordinates, which, in this experiment, are selected randomly. The experiment is run on a cluster of 100 servers connected by a switched Ethernet network. Nodes are evenly distributed across the servers. The heartbeat time is set to 250 ms if a node satisfies the stability criterion locally, and to 2 s otherwise. In Fig. 3(a) we show the time required to add M new nodes to a stable topology of N nodes, resulting in a topology with $M + N$ nodes. Each data point represents the average of five repetitions of an experiment. The figure shows that starting without an initial topology ($N = 0$), the protocols can set up a stable topology with 10 000 nodes in less than 40 seconds. Fig. 3(b) depicts a scenario where M nodes simultaneously leave a stable topology with N nodes. The graphs illustrate the time until the resulting topology of $N - M$ nodes has stabilized, where each data point again represents an average of five repetitions. The graphs show that the topology is quickly repaired even when a large number of nodes depart.

Obviously, the measurements above are specific to a particular network topology and subject to a considerable number of configuration parameters. Also, the experiments are not subjected to longer latencies as seen in a wide-area network. Yet, they establish the feasibility of maintaining large-scale application networks. The next experiment investigates the viability of application network topologies in a mobile setting.

Experiment 2 (Mobility): The experiment, from a series of outdoors measurements in 2012 on Android smartphones with ad-hoc enabled Wifi radios, explores topology changes and resulting performance degradations due to mobility of nodes. Other measurement studies of ad-hoc networks with Android smartphones are found in [35], [40], [68], [83], [85]. The neighbor selection protocol maintains the topology of an unstructured spanning tree, similar to Perlman's spanning tree algorithm [70]. Here, one node is the root of the tree. Every other node selects a node that offers the best path to the root as its parent node. Neighbor discovery and selection is based on a link quality metric, which is computed from periodically transmitted broadcast messages [57, Sec. 4.1].

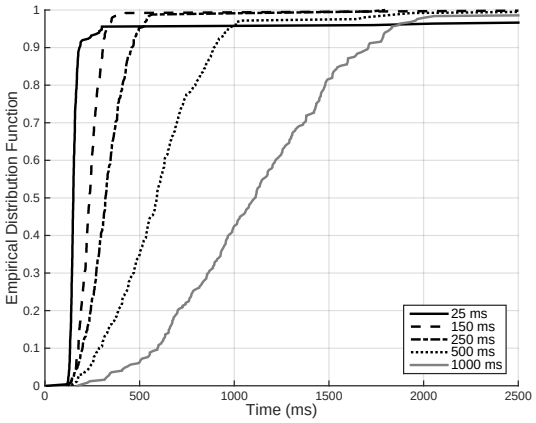
The setup of the experiment is shown in Fig. 4(a). Four stationary devices (*I1, I2, I3, I4*) are placed in a square with a side length of 40 m between them, and with a fifth stationary device (*S*) placed in the center of the square. A mobile device (*R*) moves at walking speed ($\sim$1 m/sec) in a circle pattern as indicated in the figure. There is one node running on each device, which allows us to interchange the terms 'node' and 'device'. The node in the center (with label *S*) is set to be the root of the spanning tree topology. Node S is the sender and the mobile node R is the receiver of a data transmission, which consists of 20 000 messages with a length of 512 Bytes sent at a rate of 500 kbps.

The duration of the experiment is approximately 160 s, during which the mobile node travels about one and a half rounds of the circle. Fig. 4(b) presents a time series of consecutively lost messages, where we observe that most losses in the experiment occur in bursts. Fig. 4(c) shows the changes of the network topology during the experiment. For each node (except root node *S*), the graph has a line with markers indicating time instants where a node selects a new parent node in the topology. Each marker has a label describing the result of the topology change. For example, the label of the first marker of node *R*, given by '*S*/1', states that node *R* has selected node *S* (the root node) as its parent node, and that the length of the path to the root node is one hop, i.e., node *R* is directly connected to the root node. The second marker of node *R*, '*I4*/3', states that node *R* has selected node *I4* as its parent node and that the length of the path to the root node is three hops. Fig. 4(c) indicates that the topology changes frequently, even for nodes that are not mobile. The topology changes are due to the time-varying link quality metric. A comparison of Figs. 4(b) and 4(c) shows that periods of burst losses (around 80, 130, and 150 seconds into the experiment) coincide with time instants where the mobile node *R* changes its position in the network topology (first horizontal line in Fig. 4(c)). The experiment illustrates the challenge in mobile application networks of balancing the desire to adapt the network topology to the variability of wireless communications with the performance degradation that results from frequent topology changes.

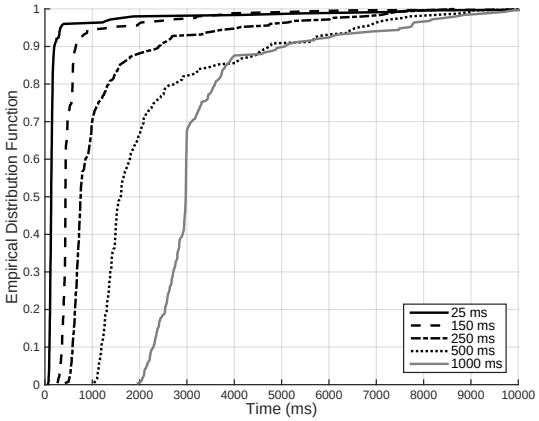
Different types of substrate networks may impose additional limitations on the maintenance of an application network. The final examples in this section evaluate the compatibility of cellular networks and Bluetooth with the creation of application networks.

Experiment 3 (Cellular Network): The experiment measures the latencies introduced by cellular networks in the creation of an application network [23]. It involves only two devices, where one of the them, node A, has a LTE cellular adapter that connects to a cellular service provider and the other, node B, has wired connectivity to the Internet. In the measurement scenario node A seeks to form an application network with node B. Note that, with only two nodes, the type of topology is not very relevant. For discovery, node A is configured with the Internet address of node B.

Initially, node B creates an application network, where it is the only member, and transmits a short message once every 10 ms. As long as node B has no neighbors, its messages are dropped and not sent out. We measure the elapsed time at node A between initiating the process of joining the application network with node B and the arrival of the first application message from node B, which we refer to as *join latency*. Measurements of the join latency are conducted for different values of the heartbeat time. Each experiment is repeated 250 times.



(a) Cellular network [23].



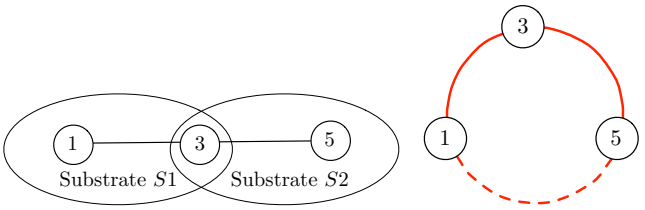
(b) Bluetooth network.

Fig. 5. Empirical distribution of the join latency for different heartbeat times.

TABLE I
STATISTICS OF THE JOIN LATENCY [23].

Heartbeat Time	Median	95^{th} Percentile	Maximum
25 ms	150 ms	300 ms	13520 ms
150 ms	235 ms	324 ms	3118 ms
250 ms	320 ms	496 ms	1797 ms
500 ms	590 ms	969 ms	2760 ms
1000 ms	1109 ms	1831 ms	5819 ms

Fig. 5(a) depicts the empirical distribution of the join latency. Table I shows the median, 95^{th} percentile, and the maximum values. We first observe that the join latency increases with the value of the heartbeat time. This is expected since the convergence time of the network topology depends on the heartbeat time. For heartbeats times between 250 and 1000 ms, the 95^{th} percentiles are close to twice the heartbeat time, whereas for heartbeat times of 25 ms and 150 ms,



(a) Substrate networks. (b) Application network.

Fig. 6. Ring topology in a multi-substrate network.

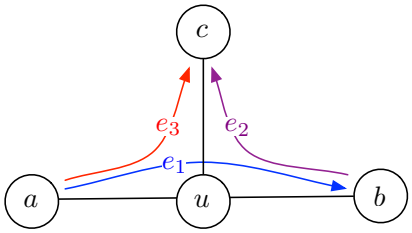


Fig. 7. Forwarding loop created by nested virtual links.

the 95^{th} percentile does not exhibit a strong correlation to the heartbeat time. This indicates that for heartbeat times of 150 ms or less, the delays incurred by the LTE network dominate the join latency. It is worth pointing out that the maximum join latency listed in Table I exceed 10 s. These outliers, which are clearly not correlated with the heartbeat time, may need to be addressed when operating application networks that involve cellular networks.

Experiment 4 (Bluetooth): This experiment measures the join latency over a Bluetooth link. It involves two Raspberry Pi 3 devices that establish an application network over their built-in Bluetooth 4.1 interfaces. The substrate network in this case is a bidirectional Bluetooth RFCOMM connection [44]. The setup of the experiment is identical to the previous experiment, where a node on one device transmits messages every 10 ms and a second node on another device joins the application network and measures the time until the first payload message arrives. The RFCOMM connection between the devices is set up before measurements commence. Fig. 5(b) shows the distribution of the join latency for different values of the heartbeat time, where the distribution shows the results of 250 repetitions. Compared to the measurements over the cellular networks, the join latency typically spans multiple cycles of the heartbeat time. As in the measurements of the cellular network, the tail of the distributions grows with the value of the heartbeat time.

C. The Need for Virtual Links

In a multi-substrate network, two nodes can exchange messages directly only if they are attached to the same substrate network. This imposes restrictions when protocols require that neighbors in the network topology must exchange messages with each other.

The issue is illustrated in Fig. 6 for an application network with a ring topology with three nodes. A ring is a structured topology, where a node identifier is a number selected from a given range, and the neighbors are the nodes with the next highest and next lowest numbers (modulo the largest number in the range). As shown in Fig. 6(a), node 1 has an attachment to substrate $S1$, node 5 has an attachment to substrate $S2$, and node 3 has attachments to both substrates. Fig. 6(b) shows the topology of the application network. Since, nodes 1 and 3 as well as nodes 3 and 5 share a common substrate network, they can exchange messages and become neighbors in the topology. On the other hand, nodes 1 and 5 cannot directly exchange messages, and therefore, without additional mechanisms, cannot become neighbors in the topology. This is indicated in Fig. 6(b) by a dashed line.

The problem of establishing nodes without a common substrate network as neighbors in a structured topology can be solved with *virtual links*. A virtual link is constructed from a multi-hop path in an application network and appears after its construction as a new edge in the network. Let $T = (V, E)$ denote the network topology graph where V is the set of nodes and $E \subseteq V \times V$ the set of edges. Given two edges $(u, v) \in E$ and $(v, w) \in E$ we can create a virtual link (u, w), denoted by $(u, w) = \mathsf{vl}\langle(u, v), (v, w)\rangle$. Virtual links can be constructed from more than two edges. In Fig.6, the construction of the ring can be completed with the virtual link $(1, 5) = \mathsf{vl}\langle(1, 3), (3, 5)\rangle$. Examples of structured network topologies with virtual links are the Unmanaged Internet Protocol (UIP) [37], [38], which builds a DHT, and Virtual Ring Routing (VRR) [19], which establishes a ring.

There are two methods to realize virtual links. One method sets up routing table entries for the virtual link at all nodes that are traversed by the virtual link. The other method employs source routing, where the nodes of a virtual link are added as a list to the message header. Each traversed node removes its identifier from the list and then passes the message to the next node in the list. Both methods require an additional encapsulation header, and hence, establish a routing layer between substrate networks and the application network.

Since virtual links appear just as regular links in the application network topology graph, it is conceivable to construct virtual links involving previously constructed virtual links. This is referred to as *virtual link nesting*. An issue with nested virtual links is that it may result in forwarding loops, as illustrated in Fig. 7. Here, the network graph consists of four nodes and three edges. Suppose there are two virtual links e_1, e_2 which are constructed by

$$e_1 = \mathsf{vl}\langle(a, u), (u, b)\rangle\,,$$
$$e_2 = \mathsf{vl}\langle(b, u), (u, c)\rangle\,.$$

If we build a nested virtual link e_3 from e_1 and e_2, that is, $e_3 = \mathsf{vl}\langle e_1, e_2\rangle$, then the messages that are forwarded on e_3 traverse the path $a \to u \to b \to u \to c$, meaning that u is traversed twice.

Without mechanisms that prevent or remove forwarding loops, nested virtual links quickly become impractical. With source routing, forwarding loops are easily detected as long as the source routes consist of a list of all nodes that are traversed by the nested virtual link. This approach is taken in VRR, where experiments showed that, after loop removal,

the paths between pairs of nodes are typically no longer than twice the shortest path lengths in the topology. For a substrate network with a grid topology, this finding has been corroborated in an analysis of scaling properties of VRR [66]. It is not known whether these scaling properties generalize to arbitrary substrate networks and other structured topologies. Overall, the know-how of benefits and limitations of a virtual link system in support of multi-substrate application networks is very limited, and the topic awaits further exploration.

III. Address Dissemination in Multi-substrate Application Networks

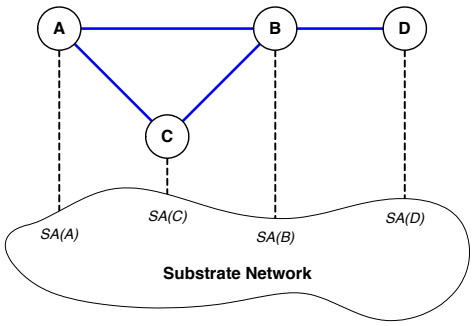
A key problem in multi-substrate networks arises from the more complex address bindings, where a node identifier is associated with multiple addresses in different substrate networks. Unless broadcast operations are available, a node can exchange messages with another node on a particular substrate network only if it knows the address of the other node on that substrate network. The construction of application networks can be facilitated by adding protocol mechanisms that exchange address information between nodes.

A. Address Bindings in Application Networks

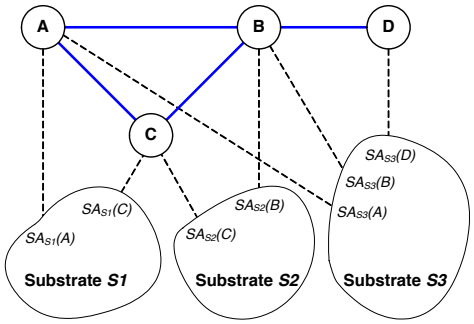
We consider that each node in an application network has a unique identifier, which, in some topologies, may also play the role of a locator address. Dependent on the network topology, identifiers can be binary strings [80], coordinates [58], or arbitrary identifiers [26]. For each attachment point to a substrate network, a node has one *substrate address*. In application networks with the Internet as substrate network, the substrate address consists of an IP address and a TCP or UDP port number. In data link networks, e.g., Bluetooth, the substrate network is a MAC address and a demultiplexing number, where the latter plays the role of a port number in the sense that it enables multiple application on the same device to share a network interface. We refer to the association of a node identifier with a substrate address as an *address binding*.

1) Address bindings in single-substrate networks: In a single-substrate scenario, each node identifier is associated with one substrate address, resulting in a one-to-one binding. For illustration, Fig. 8(a) depicts an application network with four nodes with node identifiers A, B, C, and D. Nodes connected by a link are neighbors in the application network topology. Each node has an attachment to the same substrate network, with substrate addresses $SA(A)$, $SA(B)$, $SA(C)$, and $SA(D)$, respectively. The address bindings are $\langle A; SA(A)\rangle$, $\langle B; SA(B)\rangle$, and so forth. When node A sends a message to its neighbor B, the message is addressed to $SA(B)$ in the substrate network.

With a single substrate, a node can exchange messages with every other node, as long as it has the address binding of the desired destination. In a single-substrate network, a node that receives a message can often infer the address binding of the sending node. For example, when B receives a message from A, it can extract $SA(A)$ from the source address in the encapsulating header of the arrived message to create the address binding $\langle A; SA(A)\rangle$.



(a) Single-substrate application network.



(b) Multi-substrate application network.

Fig. 8. Bindings of identifiers and substrate addresses.

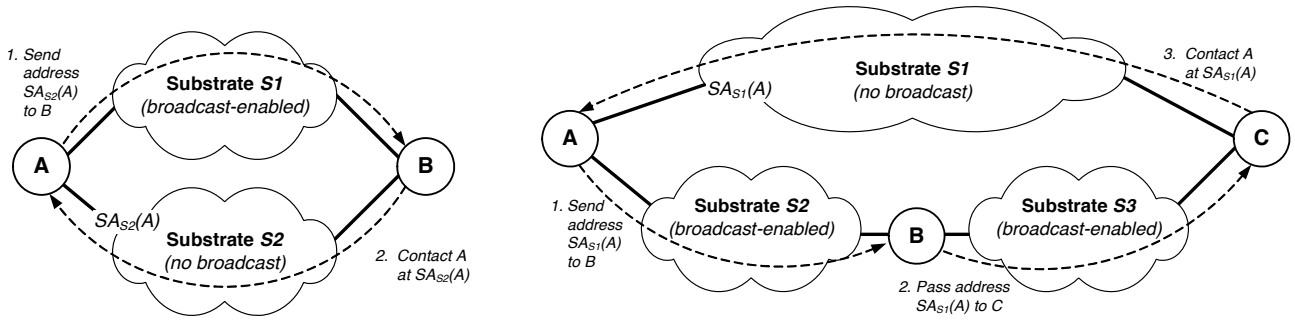
2) Address bindings in multi-substrate networks: In a multi-substrate application network, nodes can communicate with each other directly only if they share a common substrate network. Here, nodes have multiple *address bindings*, with one substrate address for each connected substrate network, resulting in a one-to-many mapping of node identifier to substrate addresses. The set of all address bindings of a node constitutes its *address list*.

Fig. 8(b) depicts the topology of a multi-substrate application network with three substrate networks $S1$, $S2$, and $S3$. Denoting by $SA_{S1}(A)$ the substrate address of node A on substrate $S1$, the address list of node A is $\{\langle A; SA_{S1}(A)\rangle, \langle A; SA_{S3}(A)\rangle\}$. Whereas in a single substrate network an address list can be extracted by inspecting encapsulation headers of incoming messages, this is not the case in a multi-substrate network. Hence, for nodes to take full advantage of multiple substrate networks, additional mechanisms are required by which nodes can disseminate address lists. In the simplest case, each message sent between nodes contains the complete address list of the sender, however, this may incur unreasonable overhead. In practice, the dissemination of address lists must trade off the benefit of having available address list information with the cost to disseminate the lists.

B. Address Dissemination Methods

The following examples motivate protocol mechanisms for exchanging address information across different substrate networks.

Example 1. In Fig. 9(a), two nodes, A and B, are both attached to substrate networks $S1$ and $S2$. Only $S1$ supports a broadcast delivery, while $S2$ is a non-broadcast network. Suppose nodes A and B prefer to connect in the application network via



(a) Example 1: Exchange over connected broadcast substrate network

(b) Example 2: Exchange over multiple substrate networks

Fig. 9. Exchange of substrate addresses across substrate networks.

$S2$, possibly because it offers a higher capacity or a higher level of security. In this scenario, A and B can discover each other on $S1$ using broadcast messages. Then, A can use $S1$ to send its substrate address in $S2$, $SA_{S2}(A)$, to B. Once B receives A's address in $S2$, it can contact A using the preferred substrate network.

A scenario as in Fig. 9(a), where two nodes connected to the same substrate network perform a direct exchange of address information, is referred to as a *direct address list exchange*.

Example 2. In Fig. 9(b), there are three nodes (A, B, C) and three substrate networks $(S1, S2, S3)$, where $S1$ is a non-broadcast network. Suppose, A has a neighborhood relation established with B over substrate $S2$, and B is a neighbor of C in substrate $S3$. Further, let the preferences of nodes A and C be such that they rather connect over $S1$. Here, A and C require each others' substrate addresses in $S1$ to establish communication over the non-broadcast substrate $S1$. This can be done by sending the substrate address of A on substrate $S1$, $SA_{S1}(A)$, to node C across substrates $S2$ and $S3$. Different from Example 1, this scenario requires the support of an intermediate node, node B.

A scenario as in Fig. 9(b), where a third-party forwards address bindings for other nodes, is referred to as a *relayed address list exchange*.

The dissemination of address information can be coupled with the protocols for discovery and topology maintenance (see Sec. II) by inspecting incoming and outgoing messages at a node. Address information can be sent as standalone messages, e.g., in response to a request, or piggybacked to an outgoing message from the node.

Direct address list exchanges are relatively straightforward. Simply, a node can attach its address list to each outgoing message. Alternatively, whenever a node receives a message from a remote node, it can check whether it has recent information on the address list of the sender of the message, and, if required, request the address list. In a broadcast network, there is the additional option to broadcast a request for an address list, similar as in the Address Resolution Protocol [71].

For a relayed address list exchange, there exist a broader range of options for address dissemination.

1) Gossip Communication: In gossip communication [45], a node that holds a piece of information periodically exchanges it with a randomly selected node. In this fashion, all nodes eventually obtain the information. In the context of address dissemination, the information consists of address lists. Each node periodically sends one or more address lists to one or more randomly selected destinations, where the time period between transmissions is referred to as the *gossip interval*. A drawback of gossip communication is that it does not take into consideration whether the disseminated information is actually needed by the receiver. Conversely, there is no guarantee that an address list of a remote node is available when needed. Flooding of address lists can be viewed as an extreme form of gossiping. With flooding, each node periodically disseminates its address list to all neighbors in the topology, which, in turn, forward them to their own neighbors. Because of the inherent high traffic volume and unavoidable duplication of messages, we do not consider flooding as a viable solution.

2) Protocol-driven dissemination: In virtually all protocols for maintaining a network topology, neighboring nodes exchange information about (non-neighbor) nodes. Such third-party node advertisements are used to learn about other nodes in the network and identify potential new neighbors in the network topology. By including address lists in third-party node advertisements, the receiver of a third-party node advertisement obtains the complete address information of the advertised nodes. Since this dissemination method is governed by the protocols that maintain the topology, we refer to it as *protocol-driven dissemination*. For example, in Fig. 9(b), node B is a neighbor of A, and therefore can learn its address lists with a direct address exchange. If B sends a message to C containing a third-party node advertisement of A and attaches the address list of A, node C obtains the substrate address needed to send a message to node A using $S1$.

Protocol-driven dissemination can operate in a proactive or on-demand fashion.

Push. In a proactive approach, referred to as *Push*, the complete address list of an advertised node is piggybacked to each third-party node advertisement.

Pull. In an on-demand approach, referred to as *Pull*, address lists are explicitly requested by a node. When a node wants to send a message to an advertised node, it sends a request for the address list to one of its neighbors. If the node receiving a request holds the requested address list, it replies to the requesting node. Otherwise, the receiving node itself issues a request to resolve the address list. In this fashion, the request is iterated until a node can respond to the query. A request for an address list of a node can be sent either to the neighbor

that had earlier sent a third-party advertisement for the node or it can be made to the next-hop neighbor on the path to the requested node. The rationale for the former is that address information about a node is more likely to be found at the node that has sent an advertisement for this node. An argument for the latter is that address information about a node is more likely to be found closer to the location of the requested node in the network topology.

The described *Pull* method shares aspects with existing address resolution protocols for non-broadcast networks. For example, the Next-Hop-Resolution-Protocol (NHRP) [63] for IP-to-ATM address resolution, for situations where multiple IP subnets are realized on a common ATM substrate, follows the IP routing table to the subnet where the requested Internet address is located.

C. Experimental Evaluation

We next present experiments that compare the effectiveness of different address dissemination methods. The experiments are conducted on a cluster of about 20 servers, where the workload is distributed approximately evenly across the servers. The substrate networks in the experiments are set up as UDP/IP networks, where each substrate network is assigned a unique identifier. Two nodes share a common substrate network and can exchange messages directly only if they have a substrate address with the same substrate identifier. All substrate networks are set up as non-broadcast networks.

Arrangement of substrate networks. Each substrate network is associated with a square in a two-dimensional plane, covering an area of $(\ell \times \ell)$. The areas of substrate networks are laid out to form an overlapping tiling, with the length of overlap given by $\ell/2$. In this fashion, a system of $N \times N$ substrate networks creates $(N+1) \times (N+1)$ tiles, which we refer to as *regions*.

The arrangement of overlapping substrate networks is illustrated in Fig. 10. The top of the figure shows the area associated with substrate S_4. Below it, substrates S_1, S_2, and S_3 overlap to form an overlapping tiling. The overlapping areas of substrate networks $S_1, \ldots, S_4$ create nine regions, where each region is associated with one or more substrate networks. In the figure, the regions are labeled as $R_{11}, \ldots, R_{33}$. By distributing nodes to the regions, we associate the nodes with substrate networks. For example, a node located in the labeled region R_{11} is only attached to substrate network S_1, while a node in region R_{22} is connected to all four substrates.

Network topology. Nodes are assigned (x, y)-coordinates such that the nodes are evenly distributed across the created regions (see Fig. 11(a)). The application networks establish a structured Delaunay triangulation topology [58], with the (x, y)-coordinates as node identifiers (see Fig. 11(b)). The protocol for Delaunay triangulations in [58] uses a central server for the discovery process (see [58, Sec. IV.2]). The experiments here use a different version of the protocol, which performs node discovery using preconfigured addresses of other nodes, the so-called *buddies*. Each node is assigned two buddies, one is located in the same region, the other buddy is node in a neighboring region. The buddies are configured

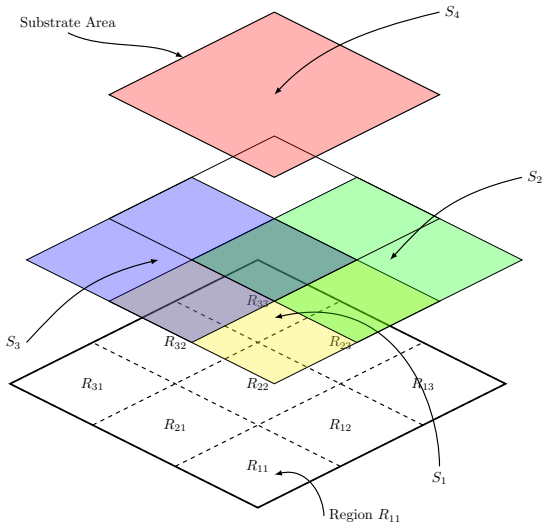
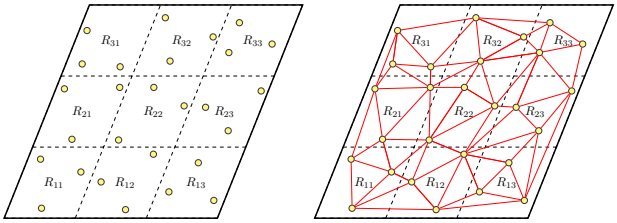


Fig. 10. Substrate arrangement in a two-dimensional plane.



(a) Placement of nodes. (b) Delaunay triangulation.

Fig. 11. Delaunay triangulation topology in a (2x2) grid of substrate networks with 9 regions.

such that it is in principle feasible to form a single connected application network.

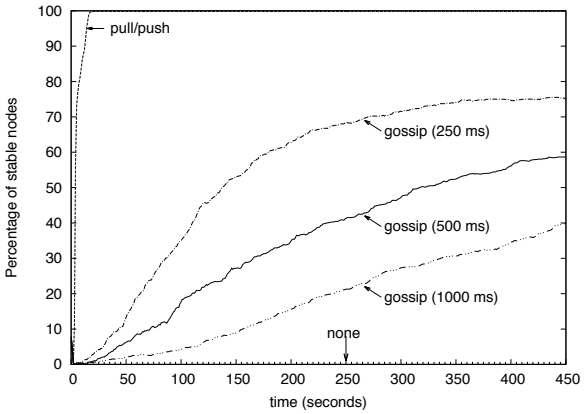
The *heartbeat* time is set to 500 msec. After each such time, a node in the Delaunay triangulation protocol sends a *HelloNeighbor* message to each of its neighbors, which includes third-party node advertisements of the closest neighbors of the node in a clockwise and counter-clockwise direction. The Push and Pull methods add to these messages the address lists of the closest clockwise and counter-clockwise neighbors.

Performance metrics. The experiments evaluate two performance metrics:

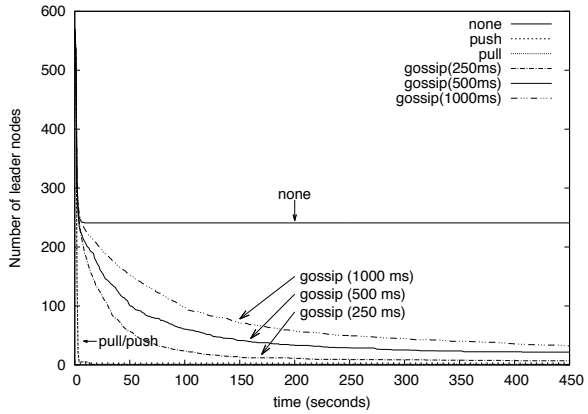
- *Stability:* For a Delaunay triangulation topology, there exists a local stability condition for each node (see [58, Sec. II-C]). If all nodes satisfy this condition, it is assured that the network has formed a stable Delaunay triangulation topology. The percentage of nodes that satisfy the stability condition measures the progress towards completing the desired topology. This metric does not, however, detect if the resulting network is partitioned in multiple disconnected networks.
- *Connectivity:* The second performance metric is the number of partitions, that is, the number of disconnected topologies that have formed.

A single stable network with a Delaunay triangulation topology has formed if and only if all nodes satisfy their stability condition and there is only one partition.

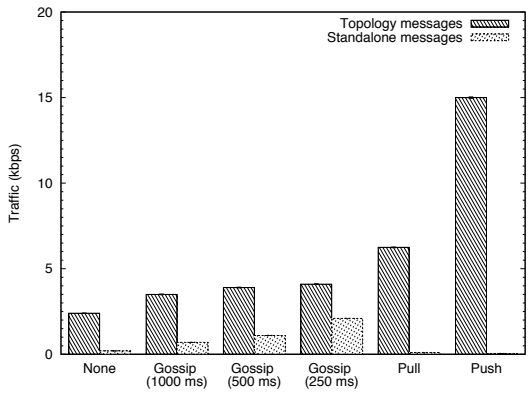
We next present results for the address list exchange methods reviewed in the previous subsection. *Gossip* refers to



(a) Stability.



(b) Connectivity.



(c) Protocol overhead.

Fig. 12. Experiment 1: Network topology with 64 substrate networks and 648 nodes [59].

the gossip protocol, with gossip intervals set to $250, 500$ and 1000 msec. *Push* and *Pull* denote the protocol-driven dissemination methods. For comparison we include results without address dissemination and refer to it as *None*. Each experiment is repeated three times, and each data point presents the average of the results.

Experiment 1 (64 Substrate Networks): The experiment involves 64 substrate networks, laid out as an 8×8 overlapping tiling, creating 81 regions. In each region, eight nodes are started simultaneously resulting in an application network

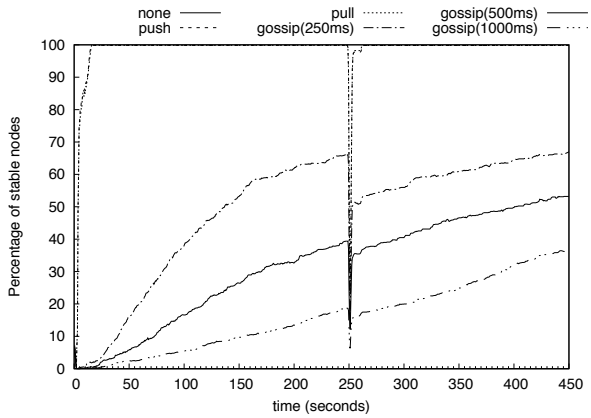
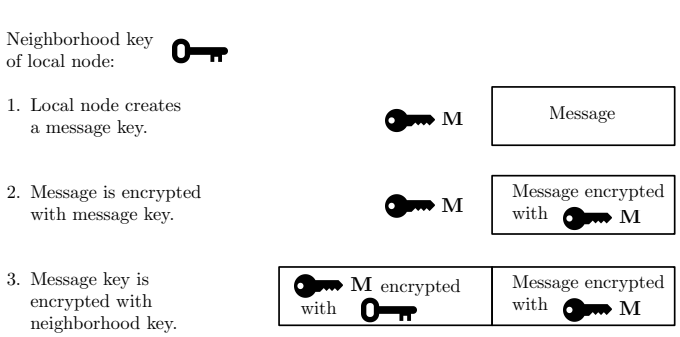


Fig. 13. Experiment 2: Stability in a scenario with churn (64 substrate networks, 648 nodes; at t=250 sec, 25% of nodes in each region leave the network).

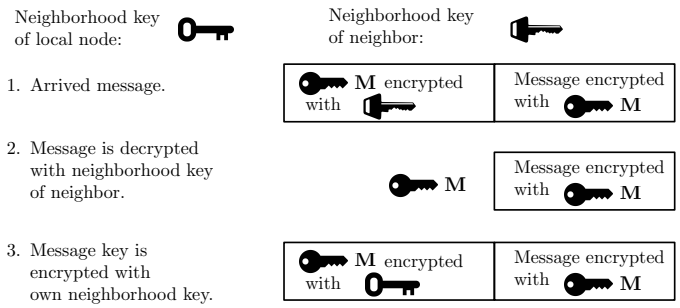
of 648 nodes. Figs. 12(a) and 12(b), respectively, depict the percentage of stable nodes in the application network and the number of partitioned topologies as functions of time. The graphs show that *Push* and *Pull* quickly establish a single stable application network. The *None* option does not lead to a stable network, thus providing evidence that address dissemination is needed to establish a single connected topology. Gossip communication shows a slow increase of the stability and connectivity measures, with better results for shorter gossip intervals.

Fig. 12(c) compares the overhead in terms of message transmissions incurred by the different methods. The figure depicts the average amount of traffic due to address dissemination received by a node, averaged over the length of the experiment. Since the Push and Pull methods piggyback address lists to messages that maintain the topology (the *HelloNeighbor* messages), we account for all messages of the protocol that build the Delaunay triangulation. These messages are labeled as *topology messages*. The Gossip method exchanges address lists in separate messages, which are labeled as *standalone messages*. As expected, *Push* incurs the most overhead, while the overhead for all other methods is modest. Since *Pull* generates considerably less protocol traffic than *Push*, it offers the best tradeoff overall.

Experiment 2 (Performance under churn): This experiment examines the address dissemination methods in a situation when the network experiences a major disruption. The setup for this experiment is the same as in the previous experiment. Half-way through the experiment, at 250 sec, 25% of randomly selected nodes in each region instantly leave the network. Fig. 13(a) presents the stability measure with different dissemination methods over the duration of the entire experiment. The figure shows that the departure event majorly disrupts the network topology, as the percentage of stable nodes drops to 30% or below. With *Push* and *Pull*, the network quickly recovers to full stability. With gossip, we observe that after an initial quick recovery, the stability settles at a level that is below that just before the departure event. The difference between the gossip and protocol-driven methods is noteworthy. After the departure event, the protocol for the network topology repairs the topology, resulting in increased



(a) Transmission of a message.



(b) Forwarding of a message.

Fig. 14. Transmission and forwarding of a message with neighborhood keys.

protocol traffic in areas where a repair is needed. Since protocol-driven methods piggyback address information to protocol messages, they deliver address information where it is needed. In contrast, gossip dissemination does not specifically direct address lists to nodes affected by the departure event. Here, the address information lost in the departure event must be slowly re-acquired.

Obviously, the experiment outcomes above are subject to the chosen protocol and their configuration parameters. Nonetheless, the results indicate that efficient address dissemination methods are a crucial component in large-scale multi-substrate application networks.

IV. Security in application networks

A security architecture for self-organizing application networks should provide security services, such as integrity, confidentiality, non-repudiation, authentication, authorization and availability in a decentralized fashion without requiring permanent availability of a global network infrastructure. Security in these networks can be approached in the conventional end-to-end fashion where any two nodes that are endpoints of a data exchange are responsible for establishing secure channels between each other [12]. In application networks there is an alternative perspective on security, where security attributes are associated with the entire application network, meaning that each member of the network is entitled to the information sent across the network. Particular requirements emerge in this setting for *backward secrecy*, that is, a new node in an application network should not be able to access data transmitted before the node joined, and *forward secrecy*, that is, a node that has left an application network should not be able to access data that is transmitted after the node departed.

In many ways, the problem of integrity and confidentiality in application networks is related to secure group communication which has been investigated in depth in the context of network-layer multicasting [72]. In most approaches to secure group communication, all members of the group share a single symmetric key, called the group key, which is used for encrypting and decrypting data between group members. Numerous methods are available for updating and distributing group keys [24], [72], [89], [90]. An issue with group keys, when applied to very large, highly dynamic groups with forward and backward secrecy is that a new key must be distributed to the entire group any time the group membership changes.

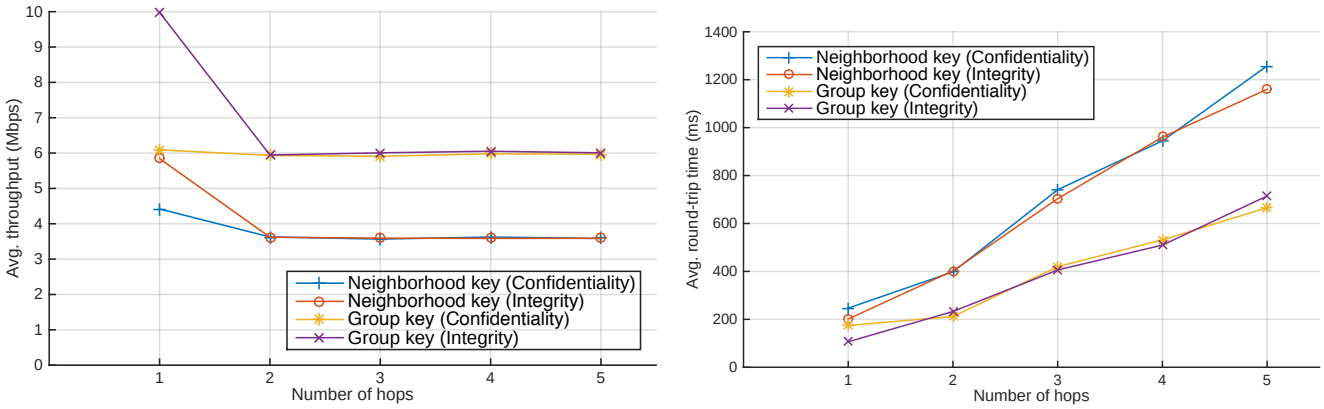
An alternative to group key management is to treat each node and its neighbors in the application network as a separate entity. Each node maintains a symmetric key, called the *neighborhood key*, that it shares with its neighbors in the network topology. Each time the set of neighbors of a node changes, that is, a new neighbor appears or an existing neighbor disappears, the node computes a new neighborhood key and securely exchanges this new key with all its current neighbors. Since nodes share keys only with their immediate neighbors in the application network, whenever a new node joins, leaves, or changes its position in the topology, only the neighbors of this node need to update their own keys.

With neighborhood keys, only neighbors in the network topology can read each others' messages. When an encrypted message is transmitted and forwarded by other nodes, the message must be decrypted and re-encrypted at each hop, which becomes impractical for large networks. The practicality of neighborhood keys can be much improved with a small twist that involves separate keys for each message, referred to as *message keys* [57]. The method is illustrated in Fig. 14. Fig. 14(a) shows the operations when a local node sends a message. The node first creates a message key for this message and uses it to encrypt the message. Then, the message key is encrypted with the neighborhood key of the node and appended to the message. This message is then transmitted to a neighbor. The forwarding of a message is shown in Fig. 14(b). When a local node receives a message from a neighbor, it first decrypts the message key with the neighborhood key of that neighbor. Then, the local node encrypts the message key with its own neighborhood key, and transmits the message to one of its neighbors. As shown in the illustration, a forwarding node only needs to decrypt and re-encrypt the message key, which is much faster than decrypting and re-encrypting the entire message. In addition to encrypting messages, neighborhood keys can also be used to ensure the integrity by adding digital signatures.

The following measurement experiment investigates the viability of this distributed security scheme.

Experiment: In the measurement experiment, six servers are configured in a line topology using dedicated links as shown in Fig. 16.

Each server runs one node of an application network, which also has a line topology. The substrate network between two nodes is a TCP connection over a dedicated Ethernet



(a) Average throughput.

(b) Average round-trip time.

Fig. 15. Performance of security algorithms.

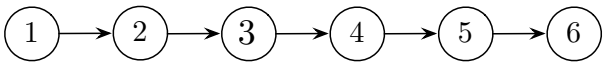


Fig. 16. Network topology with five hops.

link. Node 1 transmits 10 000 messages with a payload of 2 048 Bytes back-to-back to a receiver, where the receiver is one of nodes $2, \ldots, 6$. For each received message, the receiver sends a 32-Byte long acknowledgement to node 1.

Measurements are done for two levels of security:

1) *Integrity:* The payload and header of a message have digital signatures.
2) *Confidentiality:* In addition to signing the message headers and payloads, the payload is encrypted.

We also evaluate a group key scheme, where all nodes are provided with a common group key. This method does not use message keys. The payload of each message is signed and, for confidentiality, encrypted at the source and decrypted at the destination. The digital signatures of the message headers must be recomputed at each intermediate hop, since intermediate hops modify the message headers.

All methods use the AES algorithm [32] for data encryption and the SHA-1 algorithm [51] for digital signatures, with a key length of 128 bits. Neighborhood keys are securely exchanged between neighbors using the RSA algorithm [73]. Nodes authenticate themselves with each other by an exchange of certificates, which include the public keys of the nodes. Additional details on the security algorithms in the experiment can be found in [54].

Figs. 15(a) and 15(b), respectively, show the average throughput and round-trip times for different numbers of hops. With one hop, node 2 in Fig. 16 is the receiver, with two hops, node 3 is the receiver, and so on. The figure illustrates the performance penalty of the neighborhood key method compared to a shared key approach. Note that the average throughput for all methods is higher with one hop, when there are no intermediate nodes on the path from the sender to the receiver. The experimental data shows that the neighborhood key method results in lower throughput and increased delays compared to the group key method. Note, however, that to ensure forward and backward secrecy, the group key method requires an additional protocol for disseminating group keys.

The above neighborhood key method falls short in achieving fully distributed security, since it uses certificates for trust establishment, which borrows from a global approach. There are several methods for establishment of trust without a centralized function. In a web-of-trust [33], each network node is given the public keys of some other members, with which it has a trust relationship, and relies on them to certify the public keys of other members. A member accepts a signed public key of another member if it can find a path of trust relationships that leads to this member. Another method is to distribute the function of the certificate authority, which can be achieved with threshold cryptography [25], [64]. In (K, N) threshold cryptography [76], a secret number D is added to a randomly selected polynomial of degree $K-1$ and evaluated at N positions. Then, D can be computed by obtaining K out of N values. To build a distributed certificate authority, the private key of the authority is distributed using (K, N) threshold cryptography, yielding N partial authorities. A new member is authenticated when K partial authorities sign the new member.

V. Dynamically Created Network Services

Possibly the biggest challenge for a standardization of application networks is the diversity of networked applications, which creates a need for a wide range of algorithms and control mechanisms. Since the potential variety of services for data dissemination, data fusion, data aggregation, as well as support for flow control, error control, and congestion control are boundless, deploying application networks with implementations of all conceivable services is not plausible. On the other hand, fixing the set of services supported by application networks almost certainly results in application networks being poorly matched for some applications. A better approach is to give application networks the capability for customization, so that they can take into consideration the specific needs of applications and the resource constraints of a specific network environment.

We next discuss a design that permits protocol services to be dynamically added to an operational application network. In particular, nodes can modify services on the fly to adapt to changing application requirements. The design is motivated by the observation that many protocol services can be concisely described by finite-state machines. This can be exploited by providing mechanisms whereby nodes of an application network instantiate finite-state machines on-demand when a particular protocol service is required. Realizing services in this fashion adds the flexibility to adapt protocols to the needs of applications. Services are instantiated only when and where they are needed without requiring a universal deployment. Descriptions of protocols by finite-state machines also help with asserting correctness and other properties of a service.

A. Approaches to Protocol Customization

The concept of protocol customization and dynamic configuration is frequently revisited in networking research. Approaches to protocol customization include modularization, active networks, and automatic protocol generation. An example of the first group is the x-kernel [48], which offers protocol components, so-called micro-protocols, that can be configured by a user. Micro-protocols are also used in [16] to assemble a transport layer protocol architecture from smaller modules. Protocol boosters, described in [36], are software modules that seek to improve protocol performance by adapting network processing to applications and the network environment.

Active networking, which blossomed in the 1990s, envisioned that users inject programs into the network with the goal of customizing network services [81]. In SwitchWare [2], custom programs can be loaded on network nodes. In Active IP [88] and ANTS [82], users inject programs into the network by adding software to network packets. In ephemeral state processing [20], network packets are allowed to create and manipulate temporary state information at network switches using predefined operations.

Research on automatic protocol generation dates back to efforts to create protocol implementations from specification of the OSI protocols, such as Esterel [13], Estelle [17], and LOTOS [14], using custom compilers [21], [86]. An alternative to finite-state machines is described in [6], where network protocols are expressed in terms of a grammar. Several projects have built execution engines that can instantiate network services from protocol specifications [21], [39]. There have also been efforts to automatically generate software for application-layer peer-to-peer networks. For example, a finite-state machine approach for automatically generating network code for peer-to-peer networks with a structured topology is adopted in MACEDON [74]. Declarative overlays [61] is a methodology for automatically generating peer-to-peer networks using a declarative language based on database query languages.

B. Characterization by Finite-state Machines

Our discussion focuses on data delivery services that enhance a basic best-effort delivery in an application network. With a best-effort service, messages may be delivered in a different sequence than they were transmitted, and messages

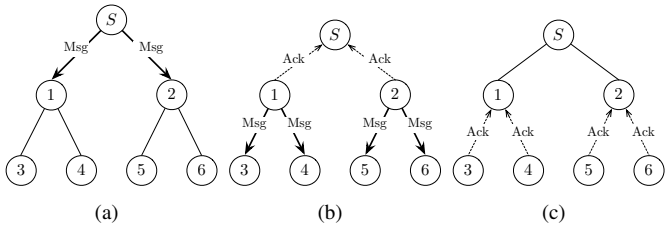


Fig. 17. Hop-to-Hop Acknowledgements: (a) The sender (S) delivers a payload message ('Msg') to its child nodes (nodes 1 and 2). (b) When nodes 1 and 2 receive the message, they send an acknowledgement ('Ack') to parent node S, and forward the message to their respective child nodes. (c) When nodes $3-6$ receive the message, they send an acknowledgement to their respective parent nodes.

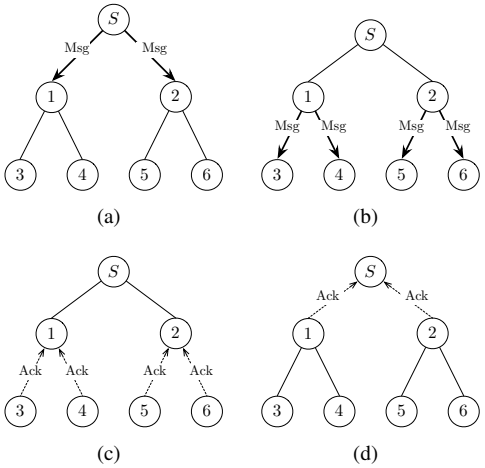


Fig. 18. End-to-End Acknowledgements: (a) The sender (S) delivers a payload message ('Msg') to its child nodes (node 1 and node 2). (b) When node 1 and node 2 receive the message, they forward the message to their respective child nodes. (c) When leaf nodes $3-6$ receive the message, they send an acknowledgement to their respective parent nodes. (d) When non-leaf nodes 1 and 2 have received acknowledgments from all child nodes, they send an acknowledgment to their parent node.

may get dropped or duplicated. As examples, we discuss services that perform broadcast transmissions in an application network with improved reliability semantics. A broadcast transmission of a message can be thought of as occurring in a rooted spanning tree that is embedded in the topology of the application network, with the sender of the message as the root of the tree. The embedded tree involves all nodes of the application network. Starting at the sender, a node transmits a payload message to neighbors that are downstream in the tree, the so-called *child nodes*. The upstream node of a child node is referred to as its *parent node*. Nodes that do not have child nodes are called *leaf nodes*. We consider two variations of the service:

- *Hop-to-Hop Acknowledgement (H2HACK):* A node that receives a payload message from its parent node immediately sends an acknowledgment to the parent node, and forwards the payload message to its child nodes. The service is illustrated in Fig. 17. If a node does not receive an acknowledgement it retransmits the payload message. When the network topology changes during a transmission, it may happen that a node receives an acknowledgement without ever having received the

corresponding payload message. In this case, the node transmits a negative acknowledgement to its parent node, which triggers a (re-)transmission of the payload message.

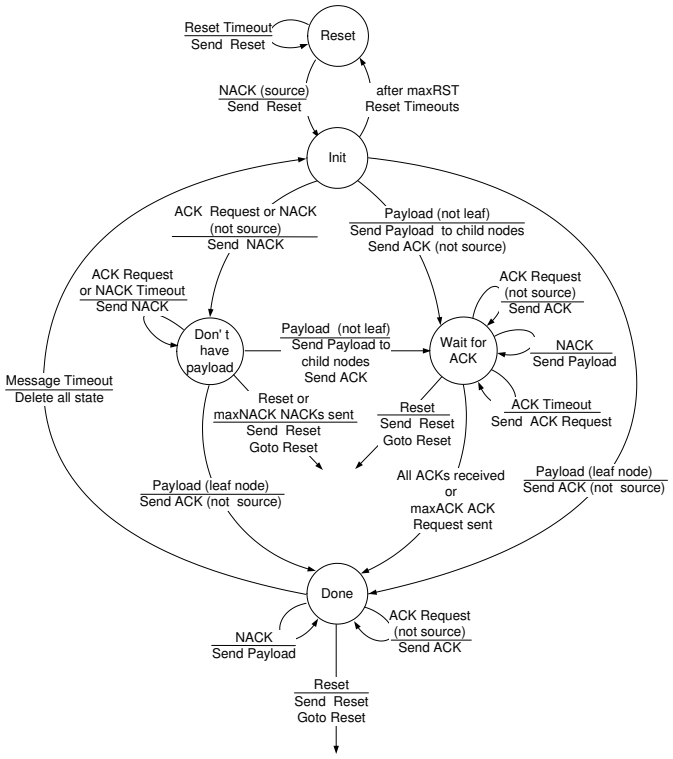
- *End-to-End Acknowledgement (E2EACK):* A node that receives a message from the parent node immediately forwards the message to its child nodes. A node sends an acknowledgement to its parent node only when (a) it is a leaf node, or (b) it is not a leaf node (and not the sender) and it has received acknowledgments from all its child nodes. We refer to Fig. 18 for an illustration. Rules for retransmissions and negative acknowledgments are similar as for the H2HACK service.

The semantics of the two services are quite different. With H2HACK, the only assurance is that each node has passed a payload message to its child nodes. With E2EACK, there is an assurance that all nodes have received the payload message, as long as the network topology has not changed. These services can be realized by the finite-state machines given in Figs. 19(a) and 19(b). We refer to the appendix for a detailed description of the services. A separate finite-state machine is created for each message. The finite-state machines govern the transmissions of positive and negative acknowledgments for any node, be it the sender, a leaf node, or a non-leaf node.

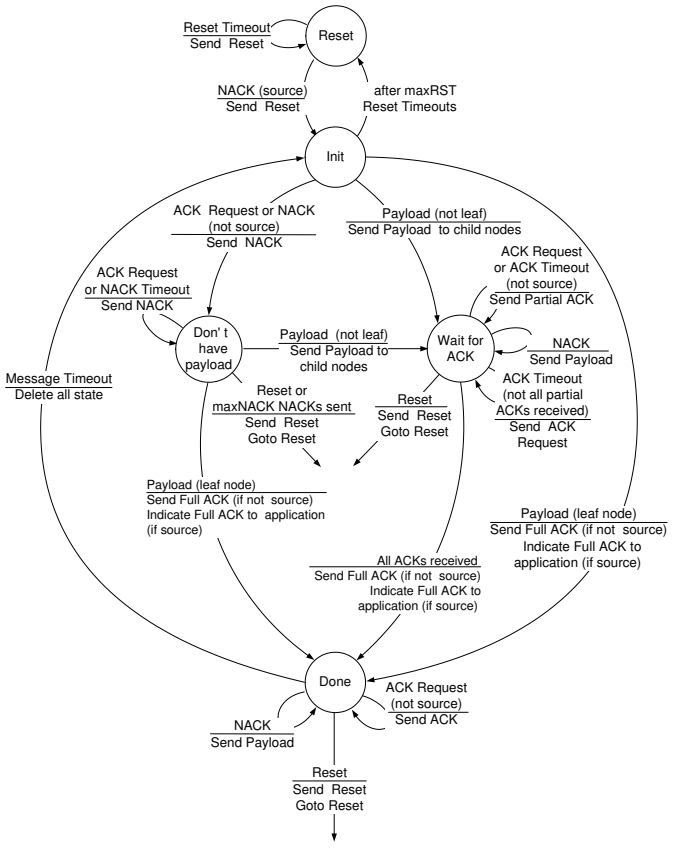
C. Executable Specifications

Finite-state machine characterizations of services, as given in Fig. 19, offer high-level design guidelines for developing software. They can also provide a basis for verifying properties of a service, e.g., using model checkers such as Spin [47]. Recently, the availability of execution environments for generic finite-state machines [10], [31] has made it possible to execute a service directly from its finite-state machine description, which we refer to as an *executable specification*. As they do not assume a specific data networking concept, execution environments for generic finite-state machines are more elementary, and at the same time more general than special-purpose execution environments for network services and protocols [21], [39], [60]. Most of all, no implementation work is required to manage the finite-state machines. The required effort is limited to building interfaces between the execution environment and the application network software.

An executable specification contains information on accepted inputs, generated outputs, and the finite-state machine representing the behaviour of the service. For executable specifications of services considered here, the accepted inputs consist of message arrivals, timer expirations, and combinations of such events. The generated outputs are invocations of tasks, such as the transmission of a message, setting a timer, and passing a message to the application. We group all inputs into *basic* events and *composite* events. A basic event is either the arrival of a message or the expiration of a timer. Composite events are constructed as first-order logic expressions involving past and current basic events. They are built by querying a database of past basic events. By adding timestamps to basic events, the database can supply the system with the concept of time. This is useful when the database



(a) Hop-to-hop acknowledgement.



(b) End-to-end acknowledgement.

Fig. 19. Finite-state machines. Each state transition lists a condition (above the horizontal line) and the required actions (below the horizontal line). A discussion of the state machines can be found in the appendix.

needs to be queried for events that occur over a time interval, for example, whether a message arrived within the last five seconds.

As an example of a composite event, consider a scenario from the services described in the previous subsection, where a node, which has sent a payload message to its child nodes, waits for an acknowledgement message from each child node. At any time after the message is sent, the following outcomes are possible:

1) Acknowledgements have been received from all child nodes;
2) Acknowledgements have been received from some but not all child nodes;
3) No acknowledgement has been received.

These events can be expressed as first-order logic expressions consisting of the basic event E_{ACK} describing the arrival of an acknowledgement message. Note that there is a separate event E_{ACK} for each child node from which an acknowledgement has been received. The relation $\mathsf{from}(n, E_{\text{ACK}})$ expresses that an acknowledgement has been received from a child node n. For the first-order logic expressions, we denote negation by $\neg$, a conjunction by $\wedge$, and the universal and existential quantifiers by $\forall$ and $\exists$, respectively. Then, the three outcomes can be described by:

1) $\forall n : (\exists E_{\text{ACK}} : \mathsf{from}(n, E_{\text{ACK}}))$;
2) $\exists n : (\exists E_{\text{ACK}} : \mathsf{from}(n, E_{\text{ACK}})) \wedge \exists n : \neg(\exists E_{\text{ACK}} : \mathsf{from}(n, E_{\text{ACK}}))$;
3) $\neg(\exists n : (\exists E_{\text{ACK}} : \mathsf{from}(n, E_{\text{ACK}})))$.

Restricting composite events to first-order logic expressions introduces limitations, since first-order logic cannot make statements associated with sets of sets or quantifications over predicates. We believe that the need for such expressions is rare in a communication context. For example, the expressions that "two messages have at least one property in common" or "if message A has the correct sequence number, then message A has a property in common with message B" are not typical. There is, however, a need to order events, as in "message A arrived before message B". Even though first-order logic cannot express such a relation, the database of timestamped basic events provides the necessary information.

The finite-state machine for a service creates output symbols, which we refer to as *actions*. Each action triggers a function call. The actions related to a service can be reduced to a small number of network primitives for creating a message, setting the content of a message, sending a message, passing a message to the local application, setting a timer, and updating information stored at a node (e.g., modifying the set of neighbors).

The services considered in this section are limited to data delivery services with special semantics. However, it should be apparent that state machines using first order logic can be extended to functions for traffic control (e.g., traffic shaping), topology management, and network discovery, since they are all based on sending and receiving messages, and operations triggered by timers. Services that the above design cannot express are those that require operations on a synchronized clock.

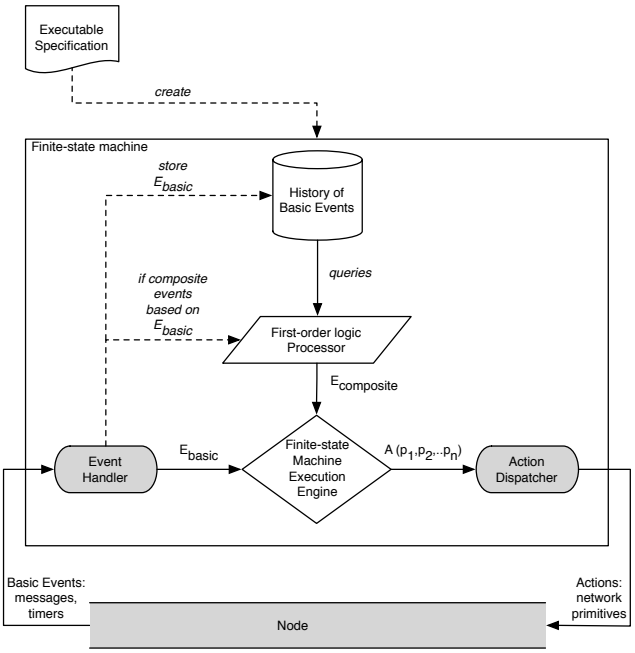


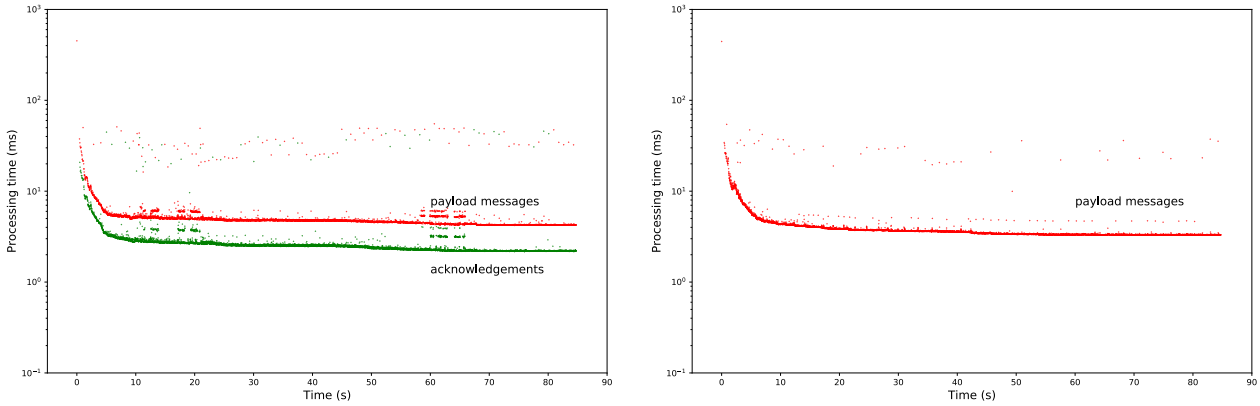
Fig. 20. Architecture of the finite-state machine execution.

D. Service Creation and Deployment

A service is created when a node supplies a new executable specification and associates a unique identifier with the service. Other nodes obtain the executable specification by querying their neighbors in the network topology. For each service, a node only requires one executable specification, which is used for all finite-state machines instantiated for this service.

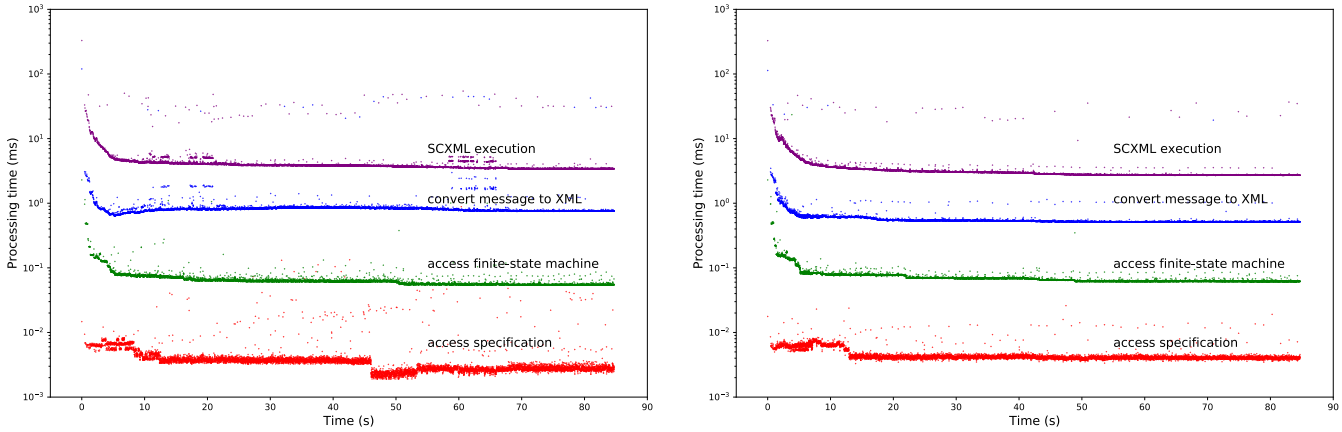
An application invokes a service by transmitting a message that is labeled with the service identifier. When a node receives a message with a service identifier from an application or from another node, it first checks if it has the executable specification for this service. If the specification is not available, the node sends a *service request message* to all its neighbors to retrieve the executable specification. If the specification is available, the node instantiates a finite-state machine for the message. The finite-state machine is labeled with a payload identifier, which is carried by all control messages associated with the same payload message. If a finite-state machine with the payload identifier already exists, it is updated based on the incoming message.

Fig. 20 provides an overview of the service architecture with finite-state machines. At the heart of the system, indicated in the figure by a rhombus, is an execution engine that manages all finite-state machines instantiated at a node. A node communicates with finite-state machines via an *event handler* and an *action dispatcher*. Basic events are delivered from the node to the event handler, which checks them against the executable specification and produces an input E_{basic} for the finite-state machine. The event E_{basic} is also stored in the history of basic events. The event handler also verifies if any composite events can be built with E_{basic} by using the first-order logic expressions defined for composite events and by querying the database with the history of basic events. Then, the finite-state machine execution engine tests if the



(a) Sender (S). (b) Receiver (R).

Fig. 21. Processing times for messages in H2HACK service [91].



(a) Sender (S). (b) Receiver (R).

Fig. 22. Detailed processing times of payload messages for H2HACK service [91].

event E_{basic} and newly built composite events lead to state transitions, which, in turn, trigger the action dispatcher to make function calls that execute network primitives.

E. Experimental Evaluation

We next present empirical measurements of an implementation of the design from Subsec. V-C and V-D. The implementation uses the State Chart XML (SCXML) [10] markup language to express executable specifications. The SCXML documents are executed on an unmodified Apache Commons SCXML execution environment [7]. Apache Commons SCXML parses SCXML specifications and creates finite-state machine instances, which are executed by a single-threaded execution engine.

The experiments are conducted on a cluster of networked servers, where each server runs one node of an application. Nodes are arranged in a line topology as shown in Fig. 16. The substrate network between any pair of nodes is a TCP connection running over a dedicated Ethernet link. Experiments of broadcast transmission with the H2HACK and E2EACK service with one sender and up to five receivers showed that the Apache Commons SCXML implementation cannot support a data rate of more than 1 Mbps [91]. We next present detailed measurements that provide insight into the system bottleneck.

The measurements only consider two nodes, where the first node (sender) transmits 10 000 messages with a payload of 1 024 Bytes to the second node (receiver) using the H2HACK service. Fig. 21 presents semi-log graphs of the processing times for each message at the sender and the receiver. Assuming a message does not need to be retransmitted, the sender invokes the finite-state machine of a message exactly twice: First, when it transmits the payload message and second, when it receives the acknowledgement for the message. The two graphs in Fig. 21(a) show the processing time of each invocation. The receiver invokes the finite-state machine only once for each message to process an incoming payload message, yielding the graph in Fig. 21(b). Due to an initialization of Java classes in the run-time system, processing times are initially higher. After the initial phase, the total processing at the sender for most payload messages and acknowledgments settle at a steady state, with a small numbers of outliers. The measurements indicate that the processing times at the sender limit the throughput of payload transmissions to a rate of around 1 Mbps.

Fig. 22 shows a repetition of the same experiment with a detailed breakup of the processing times of a payload message, measuring the latencies of four stages of the execution of a message:

1) *Access specification:* Extraction of the service identifier from a message and retrieval of the executable specification;
2) *Access finite-state machine:* Retrieval of the finite-state machine for a message;
3) *Message conversion to XML:* Creation of an XML formatted event for the execution engine;
4) *SCXML execution:* Execution of the event by the Apache Commons SCXML engine.

The graphs in Fig. 22 show that the processing times are dominated by the finite-state machine execution engine. (Note that the sum of processing times is larger than the payload processing time in Fig. 21(a). This is due to the frequent accesses to the system clock, which has an impact on the processing of messages.)

While the above experiments show the feasibility of providing network services using execution environments for generic finite-state machine, they also illustrate limitations in terms throughput and delay performance. The breakup of the processing times suggests that a faster execution engine may boost the achievable throughput by almost an order of magnitude.

VI. Conclusions

As the Internet continues to evolve to serve the world's communication needs, the emergence of proprietary network infrastructures for content delivery, low-power low-bandwidth data services, and mobile services indicates that it does not meet the needs of all applications and services. The Internet also does not exploit the ability of devices – especially, mobile devices equipped with multiple network modalities – to establish networks through self-organization without access to any infrastructure. If special-purpose networks continue to proliferate, the question arises if and how networked applications can take advantage of the presented plurality of substrate networks. We have suggested that the internetworking needs in such a setting can be addressed through self-organizing application networks, where devices use their communication modalities and accessible network infrastructures to form a network. Each instantiation of a networked application leads to the creation of a separate application network, and each application network only involves the devices that participate in the same application. We surveyed problem areas and solution approaches in such networks, and evaluated their scalability properties, their ability to adapt after failures and to support different types of substrate networks. We discussed the importance of disseminating address information in support of network formation and we studied distributed security approaches. We also discussed a design that enables application networks to adapt the set of offered services on the fly, without requiring software updates. For each of the problem areas we presented measurements of implemented solution approaches. The measurements indicated the capabilities as well as the limitations of the evaluated designs. All measurement experiments involved the same software system, which may serve as proof of concept that the presented solutions are complementary to each other and suitable for an integration into a single architecture.

Acknowledgements

The research in this paper has been supported in part by the US National Science Foundation, the US Department of Defense, the Natural Sciences and Engineering Research Council (Canada), Defence Research and Development Canada, Solana Networks, and Thales Canada Transportation Systems. We acknowledge the collection of measurements included in this paper by Mei Ya Chan, Jiyu Chen, Zian Hu, Michael Nahas, Yongbo Tang, Haizhou Wang, and Jianping Wang. The authors thank Mostafa Ammar for feedback on this manuscript.

References

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *IEEE Communications Magazine*, 40(8):102–114, Aug. 2002.

[2] D. S. Alexander, W. A. Arbaugh, M. W. Hicks, P. Kakkar, A. D. Keromytis, J. T. Moore, C. A. Gunter, S. M. Nettles, and J. M. Smith. The switchware active network architecture. *IEEE Network*, 12(3):29–36, May 1998.

[3] M. Ammar. Ex Uno Pluria: The service-infrastructure cycle, ossification, and the fragmentation of the internet. *SIGCOMM Computer Communication Review*, 48(1):56–63, Apr. 2018.

[4] AMS-IX. Statistics (Ether type), 2018. https://ams-ix.net/technical/statistics/sflow-stats/ether-type/ (Accessed: 2018-06-30).

[5] D. G. Andersen, H. Balakrishnan, F. M. Kaashoek, and R. Morris. Resilient overlay networks. In *Proc. Symposium on Operating Systems Principles*, pages 131–145, Oct. 2001.

[6] D. P. Anderson. Automated protocol implementation with RTAG. *IEEE Transactions on Software Engineering*, 14(3):291– 300, Mar. 1988.

[7] Apache Software Foundation. Commons SCXML, 2018. "https://commons.apache.org/proper/commons-scxml/ (Accessed: 2018-06-30).

[8] A. P. Athreya and P. Tague. Network self-organization in the Internet of Things. In *Proc. IEEE SECON*, pages 25–33, June 2013.

[9] L. Atzori, A. Iera, and G. Morabito. The Internet of Things: A survey. *Computer Networks*, 54(15):2787–2805, Oct. 2010.

[10] J. Barnett, R. Akolkar, R. J. Auburn, M. Bodell, D. C. Burnett, J. Carter, S. McGlashan, T. Lager, M. Helbing, and R. Hosn. State Chart XML (SCXML): state machine notation for control abstraction. *W3C working draft*, 2007.

[11] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford. In vini veritas: realistic and controlled network experimentation. In *Proc. ACM Sigcomm*, pages 3–14, Sept. 2006.

[12] M. Behringer. End-to-end security. *The Internet Protocol Journal*, 12(3):20–26, Sept. 2009.

[13] G. Berry. The foundations of Esterel. In G. Plotkin, C. Stirling, and M. Tofte, editors, *Proof, Language, and Interaction*, pages 425–454. MIT Press, 2000.

[14] T. Bolognesi and E. Brinksma. Introduction to the ISO specification language LOTOS. *Computer Networks and ISDN Systems*, 14(1):25–59, 1987.

[15] G. Bouabene, C. Jelger, C. Tschudin, S. Schmid, A. Keller, and M. May. The autonomic network architecture (ANA). *IEEE Journal on Selected Areas in Communications*, 28(1):4–14, Jan. 2010.

[16] P. G. Bridges, G. T. Wong, M. Hiltunen, R. D. Schlichting, and M. J. Barrick. A configurable and extensible transport protocol. *IEEE/ACM Transactions on Networking*, 15(6):1254–1265, Dec. 2007.

[17] S. Budkowski and P. Dembinski. An introduction to Estelle: a specification language for distributed systems. *Computer Networks and ISDN Systems*, 14(1):3–23, Mar. 1987.

[18] R. Bush and D. Meyer. Some Internet architectural guidelines and philosophy. RFC 3439, Internet Engineering Task Force (IETF), Dec. 2002.

[19] M. Caesar, M. Castro, E. B. Nightingale, G. O'Shea, and A. Rowstron. Virtual ring routing: network routing inspired by DHTs. *SIGCOMM Computer Communication Review*, 36(4):351–362, Aug. 2006.

[20] K. L. Calvert, J. Griffioen, and S. Wen. Lightweight network support for scalable end-to-end services. *SIGCOMM Computer Communication Review*, 32(4):265–278, Aug. 2002.

[21] C. Castelluccia, W. Dabbous, and S. O'Malley. Generating efficient protocol code from an abstract specification. *IEEE/ACM Transactions on Networking*, 5(4):514–524, Aug. 1997.

[22] M. Castro, M. B. Jones, A.-M. Kermarrec, A. R. M. Theimer, H. Wang, and A. A. Wolman. An evaluation of scalable application-level multicast built using peer-to-peer overlays. In *Proc. IEEE Infocom*, pages 1510–1520, Apr. 2003.

[23] M. Y. Chan, S. Baroudi, J. Siu, and J. Liebeherr. Application-layer overlay networks for communication-based train control systems. In *Proc. IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6, Apr. 2018.

[24] I. Chang, R. Engel, D. Kandlur, D. Pendarakis, and D. Saha. Key management for secure internet multicast using boolean function minimization techniques. In *Proc. IEEE Infocom*, pages 68–79, Mar./Apr. 1998.

[25] J.-H. Cho, A. Swami, and R. Chen. A survey on trust management for mobile ad hoc networks. *IEEE Communications Surveys & Tutorials*, 13(4):562–583, Fourth quarter 2011.

[26] Y.-H. Chu, S. G. Rao, S. Seshan, and H. Zhang. A case for end system multicast. In *Proc. ACM Sigmetrics*, pages 1–12, June 2000.

[27] D. Clark, R. Braden, A. Falk, and V. Pingali. FARA: reorganizing the addressing architecture. In *Proc. ACM Sigcomm Workshop on Future Directions in Network Architecture (FDNA)*, pages 313–321, Aug. 2003.

[28] G. Constable and B. Somerville, editors. *A Century of Innovation: Twenty Engineering Achievements that Transformed our Lives*. The National Academies Press, Washington, DC, 2003.

[29] J. Crowcroft. Net neutrality: the technical side of the debate: a white paper. *SIGCOMM Computer Communication Review*, 37(1):49–56, Jan. 2007.

[30] J. Crowcroft, S. Hand, R. Mortier, T. Roscoe, and A. Warfield. Plutarch: An argument for network pluralism. In *Proc. ACM Sigcomm Workshop on Future Directions in Network Architecture (FDNA)*, pages 258–266, Aug. 2003.

[31] R. B. Cruise, M. C. Hockenheimer, T. H. Mishler, P. L. Schmidt, T. H. Busch, L. A. Kittinger, K. E. Turpin, and M. A. Tokarsky. Finite state machine architecture for software development, June 2011. US Patent 8,429,605.

[32] J. Daemen and V. Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002.

[33] A. Datta, M. Hauswirth, and K. Aberer. Beyond the 'web of trust': Enabling P2P E-commerce. In *IEEE International Conference on E-Commerce*, pages 303–312, June 2003.

[34] S. Deering and R. Hinden. Internet protocol, version 6 (IPv6) specification. RFC 1883, IETF, Dec. 1995.

[35] P. M. Eittenberge, M. Herbst, and U. R. Krieger. Rapidstream: P2P streaming on android. In *Proc. IEEE 19th Packet Video Workshop*, pages 37–42, May 2012.

[36] D. C. Feldmeier, A. J. McAuley, J. M. Smith, D. S. Bakin, W. S. Marcus, and T. Raleigh. Protocol boosters. *IEEE Journal on Selected Areas in Communications*, 16(3):437–444, Apr. 1998.

[37] B. Ford. Scalable internet routing on topology-independent node identities. Technical report, Massachusetts Institute of Technology, Oct. 2003.

[38] B. Ford. Unmanaged internet protocol: taming the edge network management crisis. *SIGCOMM Computer Communication Review*, 34(1):93–98, Jan. 2004.

[39] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker. Frenetic: A network programming language. *ACM Sigplan Notices*, 46(9):279–291, 2011.

[40] P. Gardner-Stephen and S. Palaniswamy. Serval mesh software-WiFi multi model management. In *Proc. 1st International Conference on Wireless Technologies for Humanitarian Relief (ACWR)*, pages 71–77, Dec. 2011.

[41] M. Gerla. Ad hoc networks: Emerging applications, design challenges and future opportunities. In *Ad Hoc Networks*, pages 1–22. Springer, Boston, 2005.

[42] P. Gill, M. Arlitt, Z. Li, and A. Mahanti. The flattening Internet topology: Natural evolution, unsightly barnacles or contrived collapse? In *Proc. 9th International Conference on Passive and Active Network Measurement (PAM)*, pages 1–10. Springer, Apr. 2008.

[43] P. B. Godfrey, I. Ganichev, S. Shenker, and I. Stoica. Pathlet routing. In *Proc. ACM Sigcomm*, pages 111–122, Aug. 2009.

[44] B. S. I. Group. RFCOMM with TS 07.10 serial port emulation. Version 1.2, Nov. 2012.

[45] S. M. Hedetniemi and A. L. Liestman. A survey of gossiping and broadcasting in communication networks. *Networks*, 18(4):319–349, Winter 1988.

[46] S. Higginbotham. Wi-Fi vs. Internet of Things (Internet of Everything). *IEEE Spectrum*, 55(4):22, Apr. 2018.

[47] G. J. Holzmann. *The SPIN model checker: Primer and reference manual*, volume 1003. Addison-Wesley Reading, 2004.

[48] N. C. Hutchinson and L. L. Peterson. The x-kernel: An architecture for implementing network protocols. *IEEE Transactions on Software Engineering*, 17(1):64–76, Jan. 1991.

[49] IHS. Internet of Things (IoT) connected devices installed base worldwide from 2015 to 2025 (in billions), 2018. https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/ (Accessed: 2018-06-25).

[50] T. Koponen et al. Architecting for innovation. *SIGCOMM Computer Communication Review*, 41(3):24–36, July 2011.

[51] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-hashing for message authentication. RFC 2104, IETF, Feb. 1997.

[52] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, Jan. 2015.

[53] B. M. Leiner, V. G. Cerf, D. D. Clark, R. W. Kahn, L. Kleinrock, D. C. Lynch, J. Postel, L. G. Roberts, and S. Wolff. A brief history of the Internet. *SIGCOMM Computer Communication Review*, 39(5):22–31, Oct. 2009.

[54] J. Liebeherr. Security architecture (HyperCast 3.0), 2005. http://www.comm.utoronto.ca/hypercast/design/SecArchv6.pdf (Accessed: 2018-06-15).

[55] J. Liebeherr. Hypercast (Version 3). https://sourceforge.net/projects/hypercast, 2005. Accessed: 2019-1-15.

[56] J. Liebeherr. Hypercast (Version 4). https://github.com/hypercast, 2018. Accessed: 2019-1-15.

[57] J. Liebeherr and G. Dong. An overlay approach to data security in ad-hoc networks. *Ad Hoc Networks Journal*, 5(7):1055–1072, Sept. 2007.

[58] J. Liebeherr, M. Nahas, and W. Si. Application-layer multicast with Delaunay triangulations. *IEEE Journal on Selected Areas in Communications*, 40(8):1472–1488, Oct. 2002.

[59] J. Liebeherr and M. Valipour. Dissemination of address bindings in multi-substrate overlay networks. In *Proc. 23rd International Teletraffic Congress*, pages 270–277, Sept. 2011.

[60] B. T. Loo. *The design and implementation of declarative networks*. PhD thesis, University of California Berkeley, 2006.

[61] B. T. Loo, T. Condie, M. N. Garofalakis, D. E. Gay, J. M. Hellerstein, P. Maniatis, R. Ramakrishnan, T. Roscoe, and I. Stoica. Declarative networking. *Communications of the ACM*, 52(11):87–95, 2009.

[62] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Surveys & Tutorials*, 7(2):72–93, Second Quarter 2005.

[63] J. Luciani, D. Katz, D. Piscitello, B. Cole, and N. Doraswamy. NBMA next hop resolution protocol (NHRP). RFC 2332, IETF, Apr. 1998.

[64] H. Luo, J. Kong, P. Zerfos, S. Lu, and L. Zhang. URSA: ubiquitous and robust access control for mobile ad hoc networks. *IEEE/ACM Transactions on Networking*, 12(6):1049–1063, Dec. 2004.

[65] H. Ma, L. Liu, A. Zhou, and D. Zhao. On networking of Internet of Things: Explorations and challenges. *IEEE Internet of Things Journal*, 3(4):441–452, Aug. 2016.

[66] D. Malkhi, S. Sen, K. Talwar, R. F. Werneck, and U. Wieder. Virtual ring routing trends. In *Distributed Computing (DISC 2009), Lecture Notes in Computer Science, Vol. 5805*, pages 392–402. Springer, Berlin, Heidelberg, 2009.

[67] S. Mies, O. P. Waldhorst, and H. Wippel. Towards end-to-end connectivity for overlays across heterogeneous networks. In *Proc. IEEE ICC (Workshops)*, pages 1–6, June 2009.

[68] P. Meroni et.al. An opportunistic platform for Android-based mobile devices. In *Proc. ACM MobiOpp*, pages 191–193, Sept. 2010.

[69] D. A. Patterson, D. D. Clark, A. Karlin, J. Kurose, E. D. Lazowska, D. Liddle, D. McAuley, V. Paxson, S. Savage, and E. W. Zegura. Looking over the fence at networks: A neighbors view of networking research. *Computer Science and Telecommunications Board, National Academy of Sciences, Washington, DC*, 2001.

[70] R. Perlman. An algorithm for distributed computation of spanning trees in an extended LAN. In *Proc. of 9th Data Communications Symposium*, pages 44–53, Sept. 1985.

[71] D. C. Plummer. An Ethernet address resolution protocol. RFC 826, Internet Engineering Task Force (IETF), Nov. 1982.

[72] S. Rafaeli and D. Hutchison. A survey of key management for secure group communication. *ACM Computing Surveys*, 35(3):309–329, Sept. 2003.

[73] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, Feb. 1978.

[74] A. Rodriguez, C. Killian, S. Bhat, D. Kostić, and A. Vahdat. Macedon: methodology for automatically creating, evaluating, and designing overlay networks. In *Proc. Symposium on Networked Systems Design and Implementation (NSDI)*, pages 267–280, Mar. 2004.
[75] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems*, 2(4):277–288, 1984.
[76] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, Nov. 1979.
[77] Z. Sheng, S. Yang, Y. Yu, A. Vasilakos, J. McCann, and K. Leung. A survey on the IETF protocol suite for the internet of things: Standards, challenges, and opportunities. *IEEE Wireless Communications*, 20(6):91–98, Dec. 2013.
[78] R. Sibson. Locally equiangular triangulations. *The Computer Journal*, 21(3):243–245, Jan. 1977.
[79] R. Stackowiak, A. Licht, V. Mantha, and L. Nagode. Internet of things standards. In *Big Data and the Internet of Things*, pages 185–190. Apress, Berkeley, 2015.
[80] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proc. ACM Sigcomm*, pages 149–160, Aug. 2001.
[81] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, and D. J. W. amd G. J. Minden. A survey of active network research. *IEEE Communications Magazine*, 35(1):80–86, Jan. 1997.
[82] D. L. Tennenhouse and D. J. Wetherall. Towards an active network architecture. *SIGCOMM Computer Communication Review*, 37(5):81–94, Oct. 2007.
[83] J. Thomas, J. Robble, and N. Modly. Off grid communications with Android meshing the mobile world. In *Proc. IEEE Conference on Technologies for Homeland Security (HST)*, pages 401–405, Nov. 2012.
[84] J. S. Turner and D. E. Taylor. Diversifying the Internet. In *Proc. IEEE Globecom*, pages 755–760, Nov./Dec. 2005.
[85] N. Vun and Y. H. Ooi. Implementation of an android phone based video streamer. In *Proc. GreenCom (IEEE/ACM CPSCom)*, pages 912–915, Dec. 2010.
[86] S. T. Vuong, A. C. Lau, and R. I. Chan. Semiautomatic implementation of protocols using an estelle-c compiler. *IEEE Transactions on Software Engineering*, 14(3):384–393, Mar. 1988.
[87] Y. Wang and J. Touch. Application deployment in virtual networks using the X-Bone. In *Proc. DANCE: DARPA Active Networks Conference & Exposition*, pages 484–493, May 2002.
[88] D. J. Wetherall and D. L. Tennenhouse. The active IP option. In *Proc. 7th Workshop on ACM SIGOPS European workshop: Systems support for worldwide applications*, pages 33–40, Sept. 1996.
[89] C. K. Wong, M. G. Gouda, and S. S. Lam. Secure group communications using key graphs. *IEEE/ACM Transactions on Networking*, 8(1):16–30, Feb. 2000.
[90] X. B. Zhang, S. S. Lam, D.-Y. Lee, and Y. R. Yang. Protocol design for scalable and reliable group rekeying. *IEEE/ACM Transactions on Networking*, 11(6):90–922, Dec. 2003.
[91] T. Y. Zhao. Customizable services for application-layer overlay networks. Master's thesis, University of Toronto, Canada, Apr. 2013.
[92] J. C. Zuniga and B. Ponsard. Sigfox system description. LPWAN@IETF97, Nov. 14th, 2016.

APPENDIX

This appendix provides additional details on the finite-state machines illustrations in Figs. 19(a) and 19(b).

A. Hop-to-Hop Acknowledgements

Hop-to-Hop Acknowledgement is a broadcast service for the reliable transfer of payload messages across a single hop in the topology of the application network. An acknowledgment from a child node indicates that the child node has received the message.

A sender that has transmitted a payload message expects to receive an acknowledgement from its child nodes. When a node receives a payload message, it immediately sends an acknowledgment (ACK) to its parent node. If the node is not a leaf node, it also forwards the payload message to its child nodes. If a sender has not received an acknowledgment from a child after a time limit (Timeout_{Ack}), it sends the child a request for an acknowledgment (ACK Request). If a child node receives an ACK Request, but does not have the payload message, it replies with a negative acknowledgment (NACK).

A retransmission by a node is triggered by a NACK that arrives from a child node. If the payload message cannot be recovered by the child node, it will eventually give up. Parent nodes are responsible for requesting acknowledgments from their child nodes using ACK Request messages. If multiple ACK Requests are not followed up by an ACK or a NACK from a child node, the parent node will eventually give up.

The Hop-to-Hop Acknowledgement service uses four types of control messages shown in Table II.

TABLE II
MESSAGES FOR THE HOP-TO-HOP ACKNOWLEDGMENT SERVICE

Message type	Description
ACK	Acknowledges the receipt of a payload message. Sent to the parent node after receiving a payload message or an ACK Request.
ACK Request	Requests the transmission of an ACK or NACK from a child node.
NACK	Confirmation that a payload message has not been received. Sent to the parent node after receiving an ACK Request from the parent or an ACK or NACK from a child node.
Reset	A message sent to child nodes to reset the finite-state machine.

Table III provides the states of the finite-state machine in Fig. 19(a), and Table IV explains the timers of the service.

TABLE III
STATES OF THE HOP-TO-HOP ACKNOWLEDGMENT SERVICE.

State	Description
Init	Initial state.
Wait for ACK	Node has received the payload message and is waiting for ACKs from children.
Don't have payload	Node has received an ACK or NACK for a payload message, but did not receive the payload.
Done	Node has performed all required operations for a payload message.
Reset	Node deletes all state information about a message.

TABLE IV
TIMERS IN THE HOP-TO-HOP ACKNOWLEDGMENT SERVICE.

Timer	Description
ACK Timer ($\text{Timeout}_{\text{ACK}}$, maxACK)	Started when a payload message is transmitted and cancelled when an ACK is received from every child node. Upon timeout (after $\text{Timeout}_{\text{ACK}}$ seconds), an ACK Request is sent to child nodes with missing ACKs and the timer is re-started. If some ACK messages are missing after transmitting maxACK ACK Requests, the node enters the Reset state and sends Reset messages to each child node.
NACK Timer ($\text{Timeout}_{\text{NACK}}$, maxNACK)	Started when a NACK is transmitted and cancelled when the corresponding payload message is received from the parent node. Upon timeout (after $\text{Timeout}_{\text{NACK}}$ seconds), another NACK message is sent and the timer is re-started. If the payload message is not received after sending maxNACK NACKs, the node enters the Reset state and sends Reset messages to each child node.
Reset Timer ($\text{Timeout}_{\text{RST}}$, maxRST)	Started when a node enters the Reset state. Upon timeout, the node sends Reset messages to its child nodes. After maxRST timeouts, all state information about the message is deleted.
Message Timer ($\text{Timeout}_{\text{Msg}}$)	Started when all required operations for a payload message have been performed and the Done state is entered. Upon timeout, all state information about the message is deleted.

B. End-to-End Acknowledgement

The End-to-End Acknowledgement broadcast service tries to achieve a reliable transfer of a payload message to all nodes in the network. The transmission of acknowledgments proceeds in a recursive fashion. A leaf node sends a *full acknowledgment* message (Full ACK) to its parent as soon as it receives the payload message. A node that is not a leaf node transmits a Full ACK to its parent only after it has received a Full ACK from all child nodes. If the sending node receives a Full ACK from each child node, then all nodes in the application network have acknowledged the receipt of the payload message, as long as the network topology has not changed since the first transmission of the payload message.

If a non-leaf node has received some acknowledgments, but has not received a Full ACK from all of its child nodes, it sends a *partial acknowledgment* message (Partial ACK) to its parent node. This ensures that some kind of acknowledgement is transmitted upstream, even though not all nodes have acknowledged the receipt of the payload message. The time until the transmission of Partial ACKs is enforced by a timer. After a time limit, if a node has not received an acknowledgment from a child node, it sends to the child node a message to request an acknowledgment (ACK Request). If the child node does not have the payload message, it replies with a negative acknowlegdment (NACK). The message types of this service are as shown in Table II, with exception of the ACK message, which is replaced by the messages in Table V.

In addition to the timers of the hop-to-hop acknowledgement service given in Table IV, the end-to-end acknowledgement service has one additional timer as given in Table VI.

TABLE V
ADDITIONAL MESSAGES FOR THE END-TO-END ACKNOWLEDGMENT SERVICE.

Message type	Description
Full ACK	Sent by a leaf node to its parent node after it receives a payload message. A non-leaf node sends a Full ACK to its parent node after it receives Full ACKs from all of its child nodes.
Partial ACK	Sent by a node that has received some acknowledgements (Partial and/or Full ACKs), but has not received Full ACKs from all child nodes. A non-leaf node sends Partial ACKs periodically to its parent node.

TABLE VI
ADDITIONAL TIMERS IN THE END-TO-END ACKNOWLEDGMENT SERVICE.

Timer	Description
Partial ACK Timer ($\text{Timeout}_{\text{PACK}}$)	Started upon receiving a payload message for the first time. Upon timeout (after $\text{Timeout}_{\text{PACK}}$ seconds) a non-leaf node sends a Partial ACK to its parent node and restarts the timer.

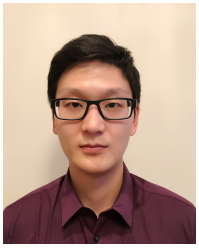
C. Other Dynamic Services

Table VII lists additional services for which executable specifications have been developed.

TABLE VII
OTHER SERVICES.

Network Service	Description
Duplicate Elimination	This service discards received payload messages at a receiver if it is a duplicate of an earlier received payload message.
Synchronization	Each node permanently stores each transmitted and received payload message and periodically synchronizes the stored payload messages with its neighbors in the network topology.
InCast	This service merges the payload of unicast payload messages with identical destination addresses and message identifiers.
Best Effort Ordering	This service attempts to pass received payload messages to the application program in the order of sequence numbers. Payload messages that do not arrive in sequence must be passed to the application program when a buffer time is exceeded. This service has one finite-state machine for each sequence of messages.

Jörg Liebeherr (S88–M92–SM03–F08) received the Ph.D. degree in computer science from the Georgia Institute of Technology, Atlanta, in 1991. He was on the faculty of the Department o Computer Science at the University of Virginia, Charlottesville, from 1992 to 2005. Since Fall 2005, he has been with the University of Toronto, as Professor of electrical and computer engineering and Nortel Chair of Network Architecture and Services.



Tony Yu Zhao received his M.A.Sc. in Electrical and Computer Engineering from the University of Toronto in 2013, with a focus on application-layer networks. He is currently working on stream processing, distributed state stores, and machine learning enabled real-time anomaly detection for big data systems.

Majid Valipour received the M.A.Sc degree in Electrical and Computer Engineering from the University of Toronto in 2010 where his research focused on self-organizing overlay networks. He is currently at Google working to improve the Web Platform.