An Overlay Approach to Data Security in Ad-Hoc Networks

Jörg Liebeherr   *Guangyu Dong*
Department of Computer Science
University of Virginia
Charlottesville, Virginia 22904

***Abstract*— Recently, application-layer overlay protocols have been considered for enhancing delivery services in mobile ad-hoc networks. This paper shows that overlay networks can provide forward and backward secrecy for application data in an ad-hoc network. We present a key management and encryption scheme, called *neighborhood key method*, where each node shares a secret with authenticated neighbors in the ad-hoc network. The neighborhood key method avoids expensive global re-keying operations when the membership in the network changes or when the network is partitioned. The method is evaluated in a newly developed application-layer ad-hoc routing protocol. Both the ad-hoc routing protocol and the security scheme are implemented in a software system for application-layer overlay networks. Extensive indoor and outdoor measurement experiments with handheld wireless devices evaluate the effectiveness of the neighborhood key method and the performance of application-layer ad-hoc networks.**

I. Introduction

A common characteristic of mobile ad-hoc networks and application-layer overlay networks is that they do not make a distinction between endsystems and relay systems (routers), that is, endsystems relay traffic for which they are neither the sender nor the receiver. In addition, both types of networks must be able to cope with frequent changes of the network topology and the set of nodes attached to the network. These similarities have stimulated interest in leveraging solutions gained in one type of network to the other. Notably, several studies recently applied application-layer overlay protocol solutions in a mobile ad-hoc context to run ad-hoc routing protocols at the application layer [10], [24] or to realize a multicast service in an ad-hoc network [4], [8], [9], [27].

An advantage of building ad-hoc networks at the application layer is that they are easy to deploy, since there is no need for compatible mesh radios or operating systems. Further, application layer solutions make it easy to add or customize network services, such as multicast, streaming, or security. The main drawbacks of ad-hoc routing at the application layer is an expected loss of performance and a reduced ability to interact with lower layers of the protocol stack.

This paper shows how application-layer overlay networks can effectively ensure backward secrecy (a new member of the network cannot access data transmitted before the member joined) and forward secrecy (a member cannot access data that is transmitted after the member left) in a mobile network. We present a key management and encryption method, called *neighborhood key method*, where each node shares a secret key with authenticated neighbors in the ad-hoc network. The neighborhood key method avoids network wide re-keying operations, without requiring that payload data be re-encrypted at each hop.

In addition to the novel security scheme, we are the first to present empirical measurement data that show the performance of application-layer ad-hoc networking on commercially available portable wireless devices (PDAs).[1] The paper also presents a new spanning tree routing protocol for ad-hoc networks which supports unicast and multicast transmissions, and evaluates its performance with the proposed security scheme.

In Section II we discuss an overlay software system that is the basis for the protocol implementation presented in this paper. In Section III we present the neighborhood key method and in Section IV we present a tree-based ad-hoc routing protocol. Each section includes a performance evaluation. In Section V we present experiments with mobile nodes that measure the performance of the routing protocol from Section IV combined with the security mechanisms of Section III. We provide brief conclusions in Section VI.

II. Overlay Performance in Ad-hoc Networks

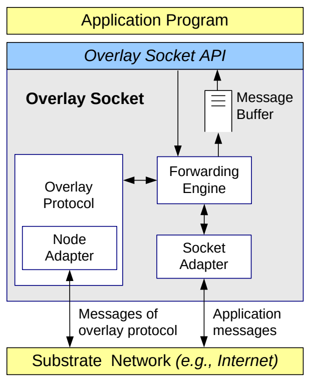
In this section we evaluate the delay and throughput performance of application-layer overlay networks in an ad-hoc environment. The implementation and experimentation of the protocols presented in this paper are realized in a software system for application-layer overlay networks [16], called HyperCast, which is described in detail in [2].

A. The Overlay Software System

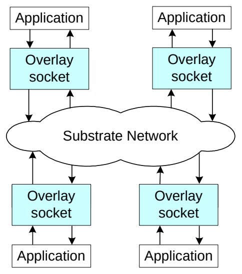
The HyperCast software uses the concept of an overlay socket as an endpoint of communication in an overlay network. An overlay network is viewed as a collection of overlay sockets (see Figure 1). The overlay socket has a message-based API for unicast and multicast transmissions that is independent of the overlay topology and the substrate (underlay) network. An overlay socket is configured with attributes from a configuration file that specify the name of the overlay network to be joined, the type of overlay topology, the type of substrate network, as well as detailed information on the size of internal buffers, protocol-specific timers, and security properties. Overlay sockets must have compatible configuration attributes to join the same overlay network.

Each overlay socket has a *logical address* and a *physical address*. The logical address is a unique identifier of the socket in the overlay network, with a format that is specific to the

[1]The measurement data presented in this paper are a selection of a large set of measurement data available at [2].

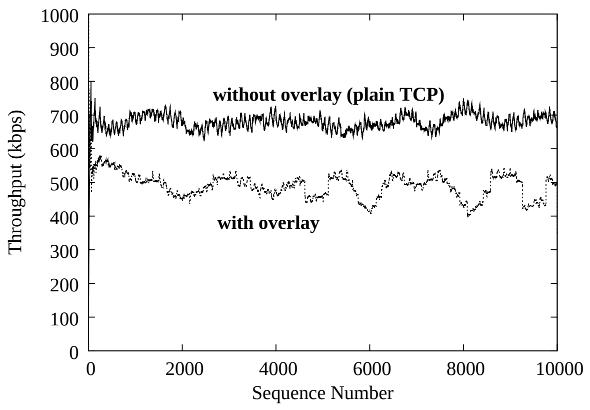


(a) Overlay socket

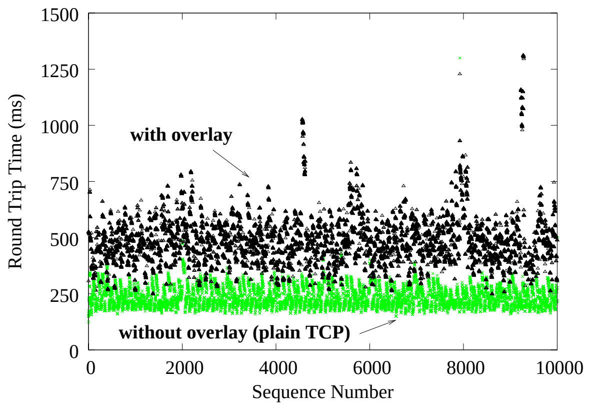


(b) Overlay Network (Collection of overlay sockets)

Fig. 1. Components of overlay sockets.



(a) Throughput.



(a) Round-trip delay.

Fig. 2. Single-hop measurements.

topology of the overlay. The physical address is a transport layer address in the substrate network. When the overlay socket runs over an IP network, physical addresses consists of IP addresses and TCP or UDP port numbers.

In Figure 1(a) we show the main components of an overlay socket and their interactions. The overlay protocol component establishes and maintains the overlay network topology. The ad-hoc routing protocol presented in Section IV is implemented as a new type of overlay protocol. Other available overlay protocols include a triangulation graph [15], a hypercube [14], and the Pastry distributed hash table [23].[2] The forwarding engine is responsible for sending and receiving formatted application messages in the overlay network. Application messages have a header of 26 bytes or more. Overlay messages can be transmitted to a single overlay socket (unicast) or to all overlay sockets in the network (multicast). The components used to access the substrate network are called adapters. Each overlay socket has two adapters: a node adapter and a socket adapter. The former handles messages of the overlay protocol and the latter transmits and receives application messages. The two interfaces to the substrate network reflects a separation of the control path (for routing messages) and the data path (for application data).

Note the similarity of the overlay socket to a software implementation of IP router functions: the forwarding engine corresponds to the IP module, the overlay protocol corresponds to a routing protocol, and the adapters play a similar role as device drivers.

B. Measurements in a Static Ad-hoc Network

Next we present measurement experiments that evaluate the performance of an overlay network on wireless hand-held devices. The experiments involve up to eight HP iPAQ 5550 PDAs, each with a 400 MHz XScale CPU, 128 MB SDRAM, 48 MB Flash ROM memory, and a 802.11b wireless network card. The 802.11b card is configured to run in peer-to-peer mode, where data is exchanged directly between wireless cards without access points. The software platform is Windows Mobile 2003 and the Jeode Runtime 1.9 Java Virtual Machine (JVM). The Jeode JVM implements the PersonalJava specification, which is a subset of Java 1.1. External class libraries are added for functions not supplied by the Jeode JVM.

[2]The implementation of Pastry is based on the FreePastry distribution [1].

The experiments in this section evaluate the throughput and delay performance of a static ad-hoc overlay network with fixed topologies. All PDAs run a single application program, each with a single identically configured overlay socket. The following experiments attempt to give insight in the performance limitations of the PDAs and the overhead of the overlay software.

Single-Hop (Unicast). In this experiment, two PDAs located in a room at a distance of about 30 feet exchange traffic. One PDA (*sender*) transmits 10,000 messages with a payload of 512 bytes to another PDA (*receiver*). For each message, the receiver transmits a short acknowledgment with a payload of 32 bytes. The sender transmits messages in a greedy fashion.

We use TCP connections over an IP network as substrate network (i.e., the overlay sockets are configured with socket adapters that establish TCP connections for transmitting application messages). With this choice, the flow and congestion control algorithms of TCP settle the transmission rate of the sender to the maximum sustainable transmission rate. We compare the results with a data transfer over a plain TCP connection without an overlay network.

Figure 2(a) depicts the throughput values, where the throughput is calculated at the receiver by computing the number of messages received over a sliding window of

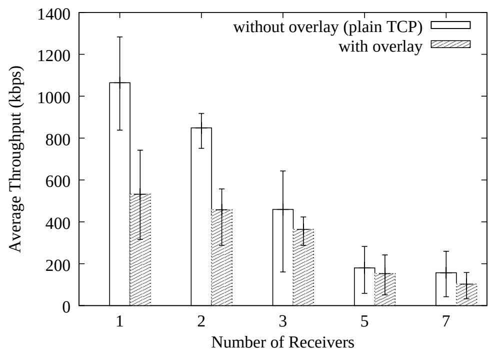


Fig. 3. Single-hop measurements (Multicast).

500 messages. Figure 2(b) shows the round-trip time for each transmitted message. The round-trip time is the elapsed time between the transmission of a message at the sender and the return of the corresponding acknowledgment. The measured data exhibits a high degree of variability, which is typical for IEEE 802.11b traffic measurements. The throughput hovers around 500 kbps when an overlay network is used, and around 700 kbps for a direct TCP connection. We conclude that the overhead of the overlay software is noticeable, but does not stymie performance.

Single-Hop Measurements (Multicast). We repeat the previous experiment with multiple receivers. We transmit application data with a multicast operation of the overlay socket API, which is translated into unicast TCP transmissions to each receiver by the overlay socket. The distance between the sending PDA and the receiving PDAs is again 30 feet, and all receivers are placed next to each other. To avoid that the sender becomes overwhelmed with acknowledgments, receivers do not send acknowledgments.

Figure 3 shows the average throughput of all data transmission, averaged over all receivers, as a function of the number of receivers. The error bars indicate the range of throughput values for any window of 500 messages. As receivers are added, the throughput expectedly declines. The performance difference of the results with and without an overlay decreases with the number of receivers. The reason is that, with many receivers, the bottleneck is the transmission of multiple copies of the same message, regardless of the presence of an overlay network.

Multi-Hop Measurements (Unicast). Here we present the results from a multi-hop outdoor experiment. For the measurement experiments, we set up six PDAs in a line as shown in Figure 4 at a distance of 30, 60, or 90 feet. The leftmost PDA in the scenario is the sender. (Increasing the distance between PDAs to 120 feet or more did not result in reliable measurements as the communication between neighboring PDAs became intermittent.) Each PDA is fixed to a pole at a height of approximately 4 feet off the ground. Figure 5 depicts the physical setup. As in the first experiment, the sender transmits 10,000 unicast messages to a single receiver, which are each acknowledged by the receiver. We compute the round-trip time and the throughput of the transmissions as described in the first experiment.

Figure 6 depicts the average throughput values when in-

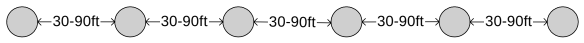


Fig. 4. Six PDAs in a line topology.

Fig. 5. Outdoor experimental setup with PDAs.

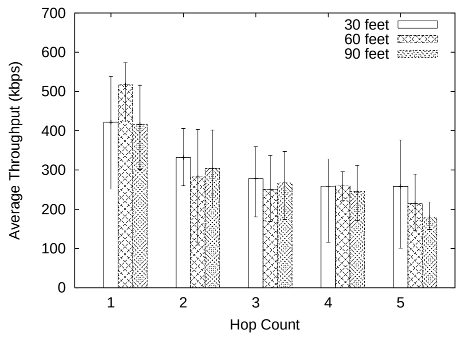


Fig. 6. Multi-hop throughput.

creasing the number of hops, for different distances between PDAs. With 1 hop, the sender and the receiver communicate directly, with 2 hops, there is 1 PDA between the sender and the receiver, and so forth. The fact that the performance sometimes seems to improve when the distance is increased can be attributed to the limitations of running wireless experiments. Our experiments were conducted over a period of several days, since the battery lifetime of the PDAs limited the number of experiments that can be conducted at one time. However, as has been observed elsewhere [3], [7], conducting the identical wireless measurement at different times of the day or under different weather conditions may have widely varying outcomes.

III. Neighborhood Key Method

We present a new key management and encryption scheme, called the *neighborhood key method*, that can assure integrity and confidentiality of application data in overlay networks, with backward secrecy and forward secrecy. Even though our evaluation will focus on ad-hoc networks, we note that the method can be applied to overlays connected to a network infrastructure. We remark that the solutions presented in this section are orthogonal to the problem of secure routing, which seeks protection against attacks to the routing protocol [11], [13].

The security mechanisms discussed in this section are implemented as a layer between the adapters and the overlay protocol in the overlay socket (see Section II). This has the benefit that the implementation of the mechanisms is independent of a particular overlay protocol. Details of the security protocols and their implementation can be found in [2].

In most approaches to secure group communications in overlays or network-layer multicast, members share a single symmetric group key for encrypting and decrypting messages [26]. The group key is updated and distributed each time the group membership changes [25], [28], [31]. This is referred to as group re-keying. Since group re-keying can be complex and often assumes access to a common server, we resort to a different solution. In our approach, each group member has its own key that it shares only with its immediate neighbors in the overlay network. As a result, re-keying becomes a local operation. The drawback of a straightforward implementation of local keys is that each message must be decrypted and re-encrypted each time it is forwarded. Our proposed scheme avoids this pitfall by encrypting message with message-specific keys which are attached to a message. In this way, only the message key must be decrypted and re-encrypted when a message is forwarded.

A. Authentication

The difficulty of authentication and trust management without access to an infrastructure with trusted third parties or intermediaries is well-documented [11], [24], [32]. Several solutions have been proposed, including advance dissemination of private keys for all node pairs, threshold cryptography approaches [18], [32], and many more, each offering a particular trade-off with respect to overhead, scalability, availability, and the ability to perform trust revocation.

We employ an authentication method based on public key certificates. We assume that each node has a certificate that has been previously signed by a trusted third party. Each node also holds certificates of trusted third parties. Without online access to certificate authorities, trust revocation is not resolved by this method, unless it is enhanced by a distributed authentication protocol, e.g., [18].

In our scheme, an exchange and verification of certificates between neighbors in the overlay occurs only when needed. When a node receives an overlay protocol message from another node for the first time it requests a signed certificate from this node and includes its own certificate in the request. Once certificates are exchanged, the nodes exchange secret keys that are used to encrypt or sign messages. Each node accepts protocol and application messages only from authenticated nodes.

The exchange of certificates is illustrated in Figure 7 for two nodes A and B. When B receives an overlay protocol message from A and the certificate of A is unknown, node B discards the message (Step 1), and sends a certification request (*CertRequest*) message to A (Step 2), which includes B's certificate. When A receives the request, it verifies the signature of B's certificate and, if valid, stores the certificate. Verification of the signature requires that the private key that signed the certificate in question be the private counterpart of the public key known to belong to a trusted third party. In the next step (Step 3), A sends a certification reply (*CertReply*) message containing its own certificate. In Figure 7, B's authentication at A is completed in Step 2, and A's authentication at B is completed in Step 3. Once authenticated, the nodes can process each others protocol messages and application messages.

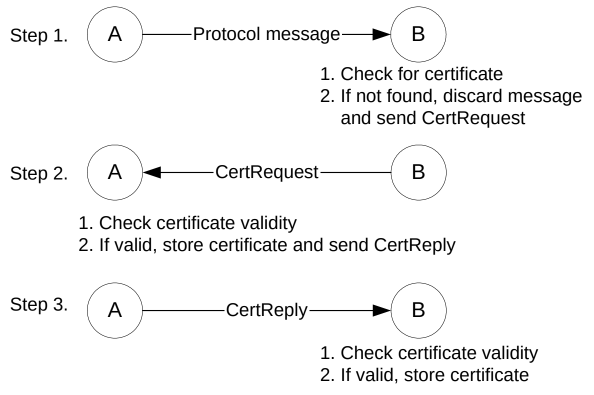

Step 1.
A
Protocol message
B
1. Check for certificate
2. If not found, discard message and send CertRequest
Step 2.
A
CertRequest
B
1. Check certificate validity
2. If valid, store certificate and send CertReply
Step 3.
A
CertReply
B
1. Check certificate validity
2. If valid, store certificate

Fig. 7. Authentication of nodes.

B. Exchange of Private Keys and Data Encryption

Encryption of data and the signing of hashes is done with symmetric keys. Each node maintains a single symmetric key with all authenticated nodes. We call this key a *neighborhood key*. Note that a node authenticates each node from which it receives a protocol message. This includes the current neighbors in the overlay, but also potential future neighbors. Whenever the set of (current or potential) authenticated neighbors changes, i.e., a new neighbor appears or an existing neighbor disappears, the node computes a new neighborhood key and sends this new key to all authenticated nodes. Neighborhood keys are securely exchanged in a *KeyUpdate* message, by encrypting the key with the public key of the receiver using the RSA algorithm. The public key is obtained from the certificate that was exchanged during the authentication.

The generation of a new neighborhood key and the transmission of a *KeyUpdate* message to authenticated neighbors is triggered when (1) a new authenticated neighbor has appeared; (2) an authenticated neighbor requests the neighborhood key; (3) an authenticated neighbor leaves the neighborhood or has not sent a message for a long time; (4) the node has reached the maximum sequence number;[3] or (5) the current neighborhood key has exceeded a specified maximum lifetime.

In Figure 7, the *Key Update* messages are sent immediately after the authentication is complete. That is, A sends a *KeyUpdate* immediately following the *CertReply* message, and A sends a *KeyUpdate* after it has verified the certificate contained in the *CertReply*. A *KeyRequest* message is transmitted when a integrity check fails on a message. Here, the node assumes that it does not have an updated neighborhood key. To prevent a malicious adversary from staging a DoS attack by sending forged messages that never pass an integrity test, the frequency of transmitted *KeyRequest* messages is limited.

If messages are encrypted or signed with neighborhood keys, only neighbors in the overlay network can decrypt or

[3]Every node maintains a sequence number for outgoing protocol and application messages, which is recorded at the receiver of a message. A receiver only accepts messages with increasing sequence numbers. The sequence number is reset when a new key is generated. When the sequence number wraps around, a new key must be generated.

verify transmitted messages. Since a note changes its neighborhood key each time a new neighbor appears or an existing neighbor departs, a newly joined node is unable to read messages sent before the node joined, and a departing node cannot read messages that are transmitted after it leaves. In this fashion, the neighborhood key method realizes backward and forward secrecy.

An alternative to a neighborhood key is a scheme where a node maintains a separate key for each neighbor. This, however, not only involves additional overhead for maintaining and storing the keys, it also requires that an outgoing message be encrypted separately for each neighbor.

In compassion to shared group keys where all nodes in the overlay network must update (re-key) the shared key whenever the membership in the overlay network changes, updating keys in the neighborhood method is a local operations, i.e., each node updates keys only with current neighbors in the overlay network. On the other hand, the workload due to updating neighborhood keys can be high. For example, when a new node joins the overlay network it may establish a neighborhood relationship with many other nodes before it converges to its final position in the overlay network. Since each change to the neighborhood requires that a node builds and distributes a new neighborhood key, the security features may delay the convergence of the overlay protocol. The problem is exacerbated during failures in the substrate network when the overlay topology must be reconstructed and many nodes join and leave the overlay network at the same time. When the time interval between changes to the neighborhood is smaller than the time required to update a neighborhood key, the overlay protocol may no longer converge to a stable topology. As a remedy, it is possible to relax the requirement of generating new keys each time the neighborhood of a node changes the overlay topology is unstable, at the cost of weakening forward and backward secrecy.

C. Data Confidentiality and Integrity

In the neighborhood key method, when an encrypted message is forwarded in the overlay network, the message must be decrypted and re-encrypted at each hop. Clearly, this is very time-consuming and not practical in large networks. To reduce the overhead incurred at each node we employ separate keys for each message. Here, when a node wants to transmit a message, it generates a new symmetric key for this message, called a *message key*, and encrypts the payload of the message with the message key. Then, the message key is encrypted with the neighborhood key and appended to the message. When a node receives an encrypted message it first decrypts the message key. (Recall that each node has the neighborhood keys of all authenticated neighbors.) If the message must be forwarded to another node, it re-encrypts the message key with its own neighborhood key.

In Figure 8(a) we show the encryption of a message that is transmitted by a node A with neighborhood key $NKey(A)$. The node generates a message key $MKey(M)$ for a message M, encrypts the message with the message key, encrypts

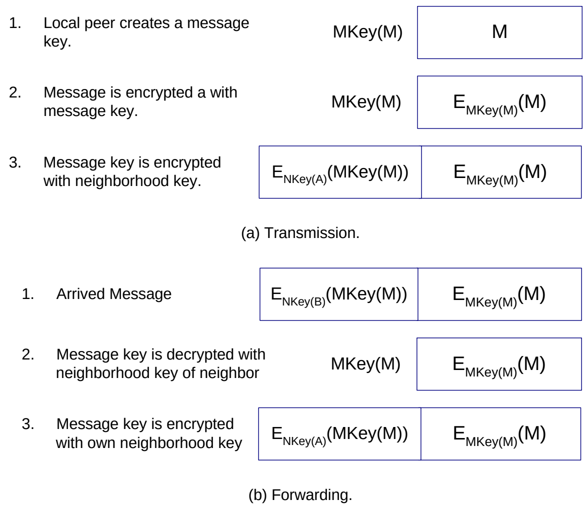
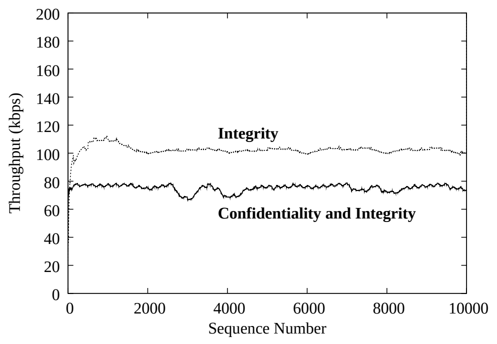

1. Local peer creates a message key.
MKey(M)
M
2. Message is encrypted a with message key.
MKey(M)
E_MKey(M)(M)
3. Message key is encrypted with neighborhood key.
E_NKey(A)(MKey(M))
E_MKey(M)(M)
(a) Transmission.
1. Arrived Message
E_NKey(B)(MKey(M))
E_MKey(M)(M)
2. Message key is decrypted with neighborhood key of neighbor
MKey(M)
E_MKey(M)(M)
3. Message key is encrypted with own neighborhood key
E_NKey(A)(MKey(M))
E_MKey(M)(M)
(b) Forwarding.

Fig. 8. Processing an encrypted application message (M is the message, $MKey(M)$ is the message key for message M, $NKey(A)$ and $NKey(B)$ are the neighborhood keys of nodes A and B, $E_{MKey(M)}(M)$ is the message encrypted with the message key, $E_{NKey(B)}(MKey(M))$ is the message key encrypted with the neighborhood key of B).

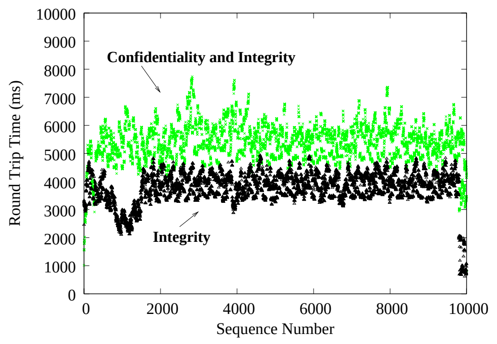
the message key with its neighborhood key, appends the encrypted message key to the message, and, finally forwards the message to a neighbor. In Figure 8(b) we show how node A forwards an encrypted message received from a neighbor B. First, A decrypts the message with B's neighborhood key, re-encrypts the message key with its own neighborhood key, and then forwards the message. Note that the encrypted message payload is not modified in this process. Merely, the encrypted message key must be processed. Since a message key is short (128 bits in our implementation, which reflects current best practices), the delay incurred by decrypting and re-encrypting the message key is limited.

The neighborhood key method is also involved in ensuring integrity of application messages and protocol messages. Both the neighborhood key and messages keys are involved in creating signed hashes, referred to as message authentication codes (MACs)[4]. There is a separate MAC for the message payload and the message header. First, a message key is used to compute the MAC of the message payload. The message key is added and encrypted with the neighborhood key, as described earlier. Then, the neighborhood key is used to compute a MAC for the message header. Both MACs, together with the encrypted message key, are transmitted as an extension header of the application message. Integrity is also provided for protocol messages. Here, the MAC is computed over the entire protocol message with the neighborhood key. Note that the MACs provides some level of route security in the sense of [11]. In our implementation, with the assumption that data confidentiality implies a desire for integrity, the MACs for the payload and header of application messages, and the MAC for protocol messages are always computed, when encryption of application payload is requested.

[4]Precisely, we use a keyed-hash message authentication code (HMAC), which involves a cryptographic hash function in combination with a secret key.



(a) Throughput.



(b) Round-trip time.

Fig. 9. Single-hop measurements.

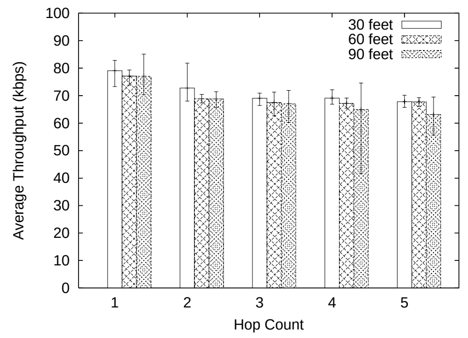


Fig. 10. Multi-hop throughput with neighborhood keys.

D. Neighborhood Key Method in Ad-Hoc Networks

We evaluate the neighborhood key method in single-hop indoor measurements and multi-hop outdoor measurements, in the same setup as discussed in Subsection II-B. We again evaluate the throughput and delay performance of a static application-layer ad-hoc overlay network. All required certificates are distributed before the measurements take place. (Due to space restrictions we cannot include our measurements of the authentication process and we refer to [2].)

We measure the performance of two following security settings. With *Integrity*, the software computes MACs for the message payload, the message header, and protocol messages, as described at the end of the previous subsection. With *Confidentiality and Integrity*, in addition, the message payload is encrypted as shown in Figure 8. A new message key is generated for each message. We only measure the performance of application messages.

The results of the throughput and (round-trip) delay measurements are shown in Figure 9. A comparison with the measurements without security from Section II shows that there is a noticeable performance penalty. The results also reflect that the operations performed for integrity and confidentiality are quite similar. The additional cost of data confidentiality is limited.

In Figure 10 we present throughput results of a multi-hop outdoor experiments, where PDAs are set up as shown in Figure 4. The values present the average throughput values over the duration of the experiment. The results show that the degradation of the throughput is small as the hop count is increased.

Our data lets us conclude that currently available PDAs can support our neighborhood key method at the application layer at data rates below 100 kbps or more. It remains open how these rates can be improved upon with an implementation that does not require a JVM.

IV. Spanning tree protocol

In this section, we describe a protocol that establishes and maintains a spanning tree overlay network topology. The protocol is referred to as *Spanning Tree Protocol* or *SPT* protocol. The SPT protocol assumes that the substrate network supports a broadcast operation for sending protocol messages. Our target environment for the SPT protocol is a mobile ad-hoc network, however, the protocol can also be used over non-wireless substrate networks.

The protocol has been implemented as an overlay protocol of the overlay socket in Figure 1. In our implementation, the overlay socket is configured with a node adapter that supports UDP multicast. Each node has a randomly assigned 32-bit long logical address that serves as unique identifier within an overlay network.

As other tree based approaches to ad-hoc routing, e.g., [22], the SPT protocol is inspired by Perlman's spanning tree algorithm for bridges [21] to a wireless environment. In the SPT protocol, all nodes agree on one node to be the root (*core*) of the spanning tree, and each node selects another node as its upstream neighbor (*ancestor*) in the rooted tree. Also, a node keeps track if it has downstream neighbors (*children*) in the rooted tree. The ancestor and the children of a node are referred to as its neighbors. A node exchanges application data only with its neighbors. Information about the neighbors is maintained in a *neighbor table*. A node also maintains a an *adjacency table* which contains a list of all nodes with which it can exchange messages.

Each node periodically broadcasts a *beacon message* in the substrate network. The information in the beacon messages is used (1) to accomplish a rendezvous of a new node with an existing spanning tree network, (2) to eliminate asymmetric links, (3) to select an ancestor in the tree, and (4) to repair partitions of the tree topology. Each node processes every beacon message that it receives.

The format of the beacon message is shown in Figure 11. The first field, the *overlay hash* contains a checksum computed over the overlay network identifier. Nodes use the hash to detect protocol messages from other overlays, which are then

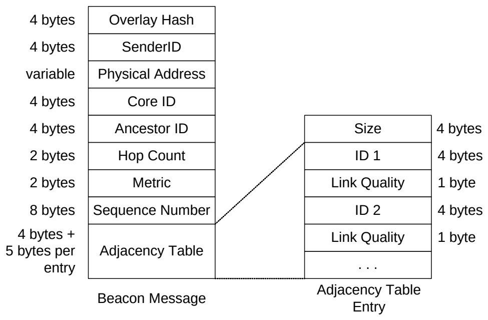


Fig. 11. Format of a beacon message.

discarded. The *SenderID* is the logical address of the sender of the message, and *Physical Address* is the address of the sender in the substrate network. If the TCP/IP suite is used as substrate network (which is the case in all our experiments), the physical address consists of an IP address and a port number. The *CoreID* is the logical address of a node, which is, according of the sender of this message, the root of the spanning tree. The *AncestorID* indicates the upstream neighbor of the sender of this message. The *HopCount* is the path length of the sender of this message to the core. The content of the *Metric* field plays a role when selecting an ancestor. Below we discuss two metrics and ancestor selection algorithms. The beacon message also contains the content of the adjacency table of the sender. Each entry of the table consists of an identifier and a link quality metric.

A. Measuring Link Quality

Many existing ad-hoc routing protocols attempt to minimize the number of hops of a route. While such protocols may exhibit good performance in simulation programs, they often look less favorably in actual networks, particularly IEEE 802.11b networks [3]. The reason is that minimizing the hop count tends to increase the geographical distance between nodes. This, however, leads to higher losses and possibly unstable links [5], [12], [17].

Alternative approaches, e.g., as proposed in [6], [7], [29], estimate the latency or the reachability between nodes when computing a path, and attempt to find routes that have low latency or good reachability. In the SPT protocol, we use an application-layer equivalent of these strategies. Specifically, we interpret the rate of successful beacon transmissions as a metric for the link quality between adjacent nodes. Using beacon messages to measure link quality does not introduce additional control traffic. We express the link quality of the unidirectional link between a node A and a node B, $LQ_B(A)$, as the ratio of beacon messages that B received from A, measured over a time period of N beacon transmission intervals (We set $N = 10$ in all of our experiments.). The quality of a bidirectional link quality is computed as

$$LQ(A, B) = min(LQ_B(A), LQ_A(B)) \ .$$

Each node A keeps track of the value $LQ_A(B)$ for all received beacon messages from adjacent nodes. When A sends a beacon message, it includes all values $LQ_A(q)$ for each node q in its adjacency table.[5] When A receives a beacon message from node B, the adjacency table in the message contains $LQ_B(A)$. With this information, A can compute $L(A, B)$. When $LQ(A, B)$ is below a threshold value (30% by default), then B is ineligible to become the ancestor of A.

When A receives a beacon message, and A is not listed in the adjacency list of the message, then A has discovered an asymmetric link, that is, A knows that B does not receive A's beacon messages. If an asymmetric link persists for a longer period of time, B is not eligible to be a neighbor of A.

B. Ancestor Selection

Each node uses the beacon messages received from adjacent nodes to select its ancestor in the spanning tree. In the SPT protocol, we have realized two ancestor selection algorithm. They both create a spanning tree, yet with different properties. The first seeks to minimize the number of hops to the core, the other seeks to maximize the quality of the path to the core. Both algorithms are extension of Perlman's spanning tree algorithm to a mobile ad-hoc environment. Due to space constraints, the following discussion is abbreviated, and we refer to [2] for details of the protocol.

The fields in the beacon message that play a role in the selection of the ancestor are the *SenderID*, *Core ID*, and the *Metric*. Initially, each node believes that it is the core and sends a beacon message with *SenderID = CoreID*. If a node receives a beacon that advertises a core with a smaller identifier, it will try to get connected to that tree and select the sender of the message as its ancestor. For messages with identical *CoreID*s, a node selects a node as new ancestor if it advertises a smaller metric value. To prevent oscillatory behavior, a new ancestor is selected only if the metric is improved by a threshold value.

Minimum Hop Count to Core. Here, each node selects an ancestor that minimizes the path length to the core. This results in a minimum-hop spanning tree. Each node sets the metric to the value of the *HopCount* field. A node changes its ancestor whenever it reduces the hop count by at least one.

Path Quality to Core. This metric tries to optimize the quality of the path to the core, by considering the link quality along the path to the core when selecting the ancestor. If $L = \{1, 2, \ldots, K\}$ denotes a set of links that form a path in the network and p_i $(0 \leq p_i \leq 1)$ is the bidirectional link quality of the i-th link, we calculate the path quality as $\prod_{i \in L} p_i$. By expressing the metric of a path as the product of the link metrics of the path, the metric can be related to the probability of a successful message transmission on the path (assuming independence of the link quality at each link). In our protocol we write the link metric as $\delta_i = -\log p_i$, which yields

$$\prod_{i \in L} p_i = e^{-\sum_{i \in L} \delta_i} \ .$$

When transmitting the path quality, we transmit the sum in the exponent $\sum_{i \in L} \delta_i$. A core node sets the value of the metric

[5]The actual value transmitted in the message is the number of received beacon messages, and not a ratio.

to 1. The values in the *Metric* field are scaled to yield good accuracy in the relevant range.

As most distributed shortest path or minimum spanning tree algorithms, the algorithm presented above is susceptible to creating transient loops and to the count-to-infinity problem. The latter problem occurs when the network state has changed and information about the old state is still propagated in the network. As suggested in the DSDV protocol [19], the problem is mitigated by adding a sequence number field to a message. In our context, a core node sets the sequence number in its beacon message and increments the number for each subsequent message. Other nodes do not change the sequence number. Every node stores the sequence number of each core, and takes action when the sequence number has not increased for a longer time period.

C. Forwarding of Data

The spanning tree established by the SPT protocol is suited for forwarding application messages to a specific destination (unicast) or to all nodes (multicast).
Multicast routing. At the source, an application transmits a new multicast message to each neighbor. At intermediate nodes, the message is forwarded to all neighbors except the neighbor form which the message was received. A message is forwarded to a neighbor with a unicast send operation over the substrate network. We also support broadcast operations in the substrate network (e.g., UDP multicast over 802.11b), where a single transmission can forward a message to all neighbors. We refer to [2] for the problems that can arise and how the SPT protocol deals with it.
Unicast routing. Unicast routing in a spanning tree poses a challenge since a node does not know if a particular destination is located upstream or downstream in the tree. Thus, as in the spanning tree algorithm for bridges [21], a node maintains a forwarding table for routing unicast messages. New entries to the forwarding table are added in two ways. The first method is a passive learning algorithm analogous to the learning algorithm for bridges [21]. The second method is an active search for a route that is triggered by sending a *RouteRequest* message. This is a control message that triggers the establishment of routing table entries. Nodes that receiver a *RouteRequest* either forward the request or respond to it by sending a *RouteReply*.

In the SPT protocol, if an address is not found in the forwarding table, the message is forwarded to all neighbors (with exception of the neighbor from which the message was received). As an alternative, the node could buffer messages until a *RouteReply* message delivers the desired entry. Such a modification would give the protocol the flavor of an on-demand protocol, such as DSR or AODV.

D. Evaluation of the SPT Protocol

We evaluate the spanning tree protocol in terms of simulation experiments using the Glomosim [30] simulator for ad-hoc networks. We evaluate how well the SPT protocol can maintain connectivity between nodes in a mobile ad-hoc network. The parameters of the simulations, shown in Table I, are typical for simulations in the published literature that evaluate ad-hoc routing protocols. There are 50 mobile nodes and 10 members transmit unicast packets to 10 receivers at a constant rate.

Parameter	Value
Simulated Area	1500 m × 500 m
Number of Nodes	50
Number of Sending Nodes	10
Wireless range	250 m
Simulation Time	900 sec
Data Rate	1 packet/sec
Message Size	512 Bytes
Max. speed	0 − 20 m/sec
Mobility Mode	Random Waypoint
Speed	*Uniform[0, max. speed]*
Pausing Time	20 sec
Transmission mode	unicast

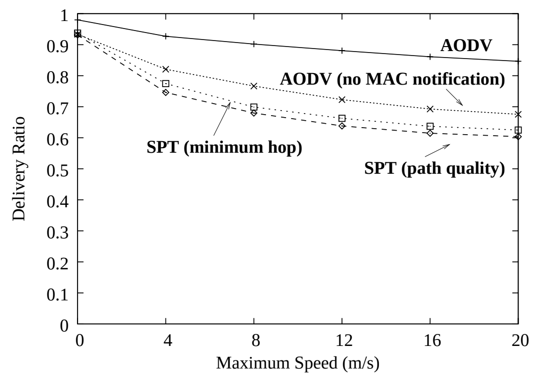
TABLE I
SIMULATION PARAMETERS.

We compare the two versions of the SPT protocol (hop count and path quality) to the AODV protocol [20], which is one of the main on-demand ad-hoc routing protocols. The term ‘on-demand’ refers to the fact that AODV builds a path to a destination only if there is data to be transmitted. The path established by AODV follow the shortest reverse path to the source. Data is buffered at the source until a path is established. AODV has a MAC layer notification mechanism that detects when a link of a path becomes unavailable, and repairs an interrupted path.

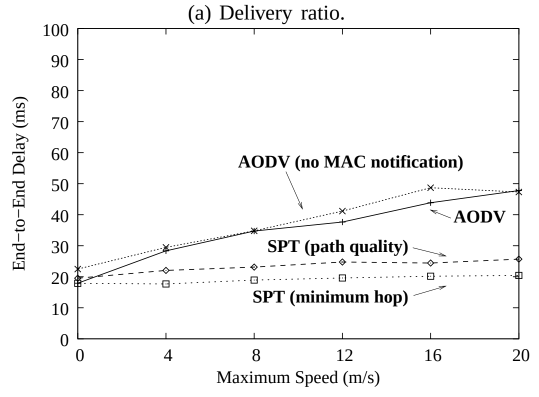
We consider the following performance metrics, which are frequently used to evaluate ad-hoc routing protocols. The *Delivery Ratio*, defined as the fraction of transmitted packets that are successfully delivered to the destination, measures how well a protocol finds routing paths in a network with mobile nodes. The *Average End-to-End Delay* is the time to deliver a packet to the destination averaged over all transmissions. The *Normalized Forwarding Overhead* is the ratio of the number of events when packet transmissions occur (either sent at the source or forwarded at intermediate nodes) and the number of events when a packet arrives at the receiver. The optimal ratio is achieved when packets are always forwarded on the shortest path to their destination. Inefficient routes, routes that are out of date because of mobile nodes, and inefficient forwarding, e.g., flooding of packets, increase the normalized forwarding overhead.

Figure 12 depicts the performance measures as a function of increasing mobility of nodes. Each data point represents the average of ten simulation runs, Figure 12(a) shows that the delivery ratio of the SPT protocol is lower than that of the AODV protocol, with a larger difference at higher speeds. We also consider a modification to AODV where we disable the MAC notification for unavailable links. The purpose of this is to show how an application-layer version of AODV may perform that does not have access from lower layers of the protocol stack.

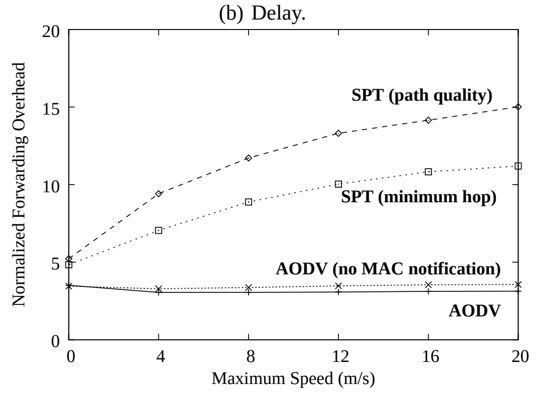
The graph in Figure 12(b) shows that the SPT protocol has



(a) Delivery ratio.



(b) Delay.



(c) Forwarding overhead.

Fig. 12. Simulation of mobile network.

lower delays. This is intuitive, since AODV buffers packets when it does not know how to forward a packet, whereas the SPT protocol floods a packet in such a situation. Figure 12(c) illustrates the cost of the SPT protocol in terms of forwarding overhead. The forwarding overhead is higher than in AODV since unicast routes in the SPT protocol do not minimize the path between senders and receivers. Also, when a route is not known a message is flooded to all neighbors resulting in multiple transmissions of the same packet. In all graphs, the minimum-hop version of SPT has a better performance than the path quality variant. In the next section, we will see that this is not true in our measurement experiments.

Overall, the simulation results show that the SPT protocol provides favorable results, which, in comparison to a popular ad-hoc protocol, improves delay performance at the cost of a lower delivery ratio and higher forwarding overhead.

V. SPT Protocol with Data Security

In this section we put together all protocols developed in this paper to evaluate how the neighborhood key method performs in an ad-hoc network with mobile nodes that self-organize using the SPT protocol.

The experiments are outdoor measurements with PDAs as shown in Figure 5. The setup of the PDAs is shown in Figure 13. Six PDAs (A, B, C, D, E, F) are placed in a line with a distance of approximately 90 ft between them. In addition, a person holding a PDA (labeled as M) walks parallel to the fixed PDAs at a distance of about 50 ft, covering a round-trip distance of about 1260 ft.

In the experiment, A transmits unicast messages with a 512 byte payload to the mobile PDA at a rate of 10 messages per second. All messages are transmitted using UDP unicast as substrate network. The spanning tree protocol is configured so that E is the core of the tree.

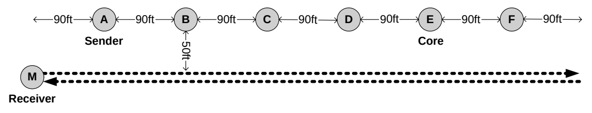
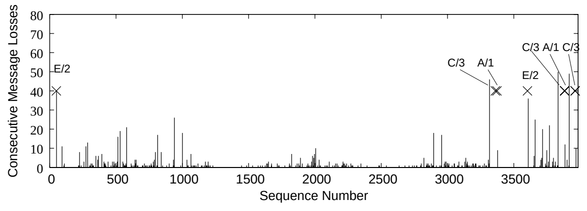
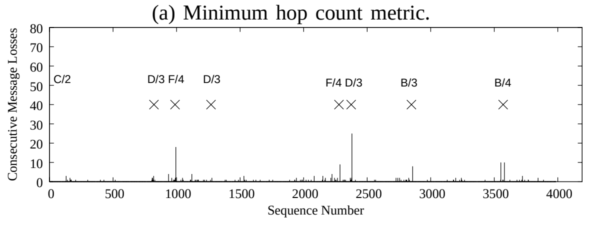


Fig. 13. Measurement scenario with a mobile receiver.

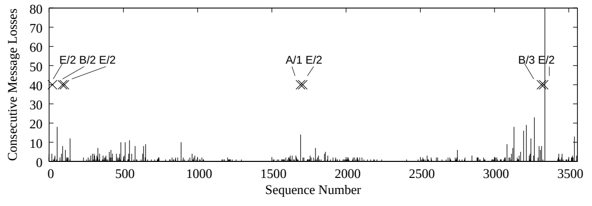


(a) Minimum hop count metric.

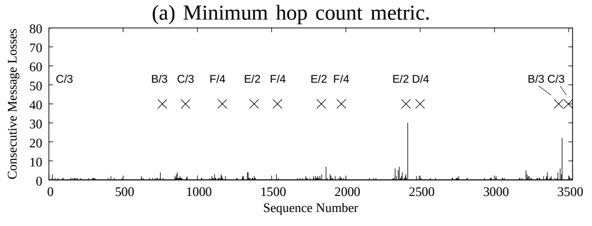


(b) Path quality metric.

Fig. 14. Measurements with mobile receiver (confidentiality not enabled).



(a) Minimum hop count metric.



(b) Path quality metric.

Fig. 15. Measurements with mobile receiver (confidentiality enabled).

We present results of two sets of measurements. For each set, we run the spanning tree protocol with the minimum hop metric and the path quality metric. We measure the degradation due to mobility by recording gaps of the transmission stream at the receiver. Specifically, we plot the consecutive number of lost messages at the receiver. We also record changes to the route between the sender and the receiver.

Figure 14 shows the outcome a measurement when security features are not enabled. The crosses ($\times$)) in the plots indicate a change of the spanning tree topology that affects the mobile PDA. We also include the length of the route form the sender to the receiver. For example, a label '*C/3*' indicates that C is the ancestor of the mobile PDA, and that the path from the sender (A) to the mobile receiver (M) is 3 hops long. A comparison of Figures 14(a) and 14(b) shows that the path quality metric has significantly fewer losses.

In Figure 15 we repeat the first measurement, but, in addition, provide data confidentiality and integrity with the neighborhood key method. Here, each time the mobile PDA exchanges a beacon message with one of the fixed PDAs for the first time, there is an exchange of certificates and neighborhood keys. For message encryption, we generate a new message key for each transmitted message. Recall that the message key is decrypted and re-encrypted at each hop between the sender and the receiver. A comparison of Figures 14 and 15 shows that the security mechanisms only cause a limited degradation of the recorded message losses.

VI. Conclusions

We presented overlay network protocols for ad-hoc networks that ensure forward and backward secrecy for application data. All routing and security functions were realized and evaluated in an operational application-layer overlay network system. Measurement experiments with PDAs shed light on the throughput and delay performance achievable with state-of-the-art handheld wireless devices. While throughput and delay performance of currently available PDAs limit their applicability to low-bandwidth scenarios, this paper has empirically demonstrated the efficacy of application layer ad-hoc networking with (and without) data security.

References

[1] Freepastry. http://freepastry.rice.edu.

[2] Hypercast design documents and materials. http://www.cs.virginia.edu/hypercast/material.html, 2005.

[3] D. Aguayo, J. Bicket, S. Biswas, G. Judd, and R. Morris. Link-level measurements from an 802.11b mesh network. In *Proc. of ACM Sigcomm'04*, Aug. 2004.

[4] K. Chen and K. Nahrstedt. Effective location-guided overlay multicast in mobile ad hoc networks, 2005. To appear.

[5] K.W. Chin, J. Judge, A. Williams, and R. kermode. Implementation experience with manet routing protocols. *ACM Sigcomm Computer Communications Review*, 32(5), Nov. 2002.

[6] D. S. J. De Couto, D. Aguayo, J. Bicket, and R. Morris. A high-throughput path metric for multi-hop wireless routing. In *Proc. of ACM MobiCom'03*, Sep. 2003.

[7] R. Draves, J. Padhye, and B. Zill. Comparison of routing metrics for static multi-hop wireless networks. In *Proc. of ACM Sigcomm'04*, Aug. 2004.

[8] M. Ge, S. V. Krishnamurthy, and M. Faloutsos. Overlay multicasting for ad hoc networks. In *Proc. of Third Mediteranean Ad Hoc Networking Workshop (Med-Hoc-Net), Bordum, Turkey*, 2004.

[9] C. Gui and P. Mohapatra. Efficient overlay multicast for mobile ad hoc networks. In *Proc. of IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1118–1123, March 2003.

[10] Y. C. Hu, S. M. Das, and H. Pucha. Exploiting the synergy between peer-to-peer and mobile ad hoc networks. In *Proc. of 9th Workshop on Hot Topics in Operating Systems (HotOS IX)*, May 2003.

[11] Y.-C. Hu and A. Perrig. A survey of secure wireless ad hoc routing. *IEEE Security and Privacy*, 2(3):28–39, May/June 2004.

[12] Y.C. Hu and D.B. Johnson. Design and demonstration of live audio and video over multihop wireless ad hoc networks. In *Proc. of IEEE Milcom'02*, 2002.

[13] C. Karlof and D. Wagner. Secure routing in wireless sensore networks: attacks and countermeasures. *Ad Hoc Networks (Elsevier)*, 1(3):293–315, 2003.

[14] J. Liebeherr and Tyler K. Bean. Hypercast: a protocol for maintaining multicast group members in a logical hypercube topology. *Proc. of First Workshop on Networked Group Communications (NGC)*, Jul 1999.

[15] 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.

[16] J. Liebeherr, J. Wang, and G. Zhang. Programming overlay networks with overlay sockets. In *Proc. 5th COST 264 Workshop on Networked Group Communications (NGC 2003), LNCS 2816*, pages 242–253, Sep. 2003.

[17] H. Lundgren, E. Nordstrom, and C. Tschudin. Coping with communication grayzones in IEEE 802.11b. In *Proc. of 5th ACM International Workshop on Wireless Mobile Multimedia (WoWMoM 2002)*, Sep. 2002.

[18] H. Luo, J. Kong, P. Zerfos, S. Lu, and L. Zhang. Ursa: Ubiquitous and robust access control for mobile ad hoc networks. *ACM/IEEE Transactions on Networking*, 2005. To appear.

[19] C. E. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance vector routing (DSDV). In *Proc. of ACM Sigcomm*, pages 234–244, Sep. 1994.

[20] C. E. Perkins and E. M. Royer. Ad-hoc on-demand distance vector routing. In *Proc. 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, Feb. 1999.

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

[22] S. Radhakrishnan and G. Racherla. Protocol for dynamic ad-hoc networks using distributed spanning trees. *Wireless Networks*, 9:673–686, 2003.

[23] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. of IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany*, Nov. 2001.

[24] R. Shollmeier, I. Gruber, and M. Finkenzeller. Routing in mobile ad-hoc and peer-to-peer networks: A comparison. In *Proc. of International Workshop on Peer-to-Peer Computing, Pisa, Italy*, May 2002.

[25] M. Steiner, G. Tsudik, and M. Waidner. Key agreement in dynamic peer groups. *IEEE Transactions on Parallel and Distributed Systems*, 11(8):769–780, Aug. 2000.

[26] D. Wallner, E. Harder, and R. Agee. Key management for multicast: Issues and architectures. IETF RFC 2627, June 1999.

[27] L. Xiao and et. al. Prioritized overlay multicast in mobile ad hoc environments. *IEEE Computer*, 37(2):67–74, Feb 2004.

[28] Y. R. Yang, X. S. Li, X. B. Zhang, and S. S. Lam. Reliable group rekeying: A performance analysis. In *Proc. of ACM Sigcomm*, pages 27–38, Aug. 2001.

[29] M. Yarvis, W. S. Conner, and L. Krishnamurthy. Real-world experience with an interactive ad hoc sensor network. In *Proc. of the International Workshop on Ad Hoc Networks*, Aug. 2002.

[30] X. Zeng, R. Bagrodia, and M Gerla. Glomosim: A library for parallel simulation of large-scale wireless networks. In *Proc. of Workshop on Parallel and Distributed Simulation*, pages 154–161, 1998.

[31] X. B. Zhang, S. S. Lam, and H. Liu. Efficient group rekeying using application-layer multicast. In *Proc. of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS 2005)*, Jun. 2005. To appear.

[32] L. Zhou and Z. J. Haas. Securing ad hoc networks. *IEEE Network*, 13(6):24–30, Nov./Dec. 1999.