

An Overlay Approach to Data Security in Ad-Hoc Networks ^{*}

Jörg Liebeherr[†]

Guangyu Dong^{††}

[†] Department of Electrical and Computer Engineering
University of Toronto
Toronto, Ontario, M5S 1L2

^{††} Free Peers, Inc.
Miami Beach, FL 33141

December 2005

Abstract

While it has been argued that application-layer overlay protocols can enhance services in mobile ad-hoc networks, hardly any empirical data is available on the throughput and delay performance achievable in this fashion. This paper presents an application-layer overlay approach to ensure integrity and confidentiality of application data in an ad-hoc environment. A key management and encryption scheme, called *neighborhood key method*, is presented where each node shares secrets only with authenticated neighbors in the ad-hoc network, thus avoiding global re-keying operations. All proposed solutions have been implemented and empirically evaluated in an existing software system for application-layer overlay networking. Results from indoor and outdoor measurement experiments with mobile handheld devices provide insight into the performance and overhead of overlay networking and application layer security services in ad-hoc networks.

1 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

^{*}The research in this report was done while the authors were with the University of Virginia. The research is supported in part by the National Science Foundation under grant ANI-0085955.

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, 20] or to realize a multicast service in an ad-hoc network [3, 8, 9, 23].

An advantage of building ad-hoc networks at the application layer is that they are easy to deploy, since there is no need for protocol compatibility at the OS or hardware level. 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 additional overhead for communication and computing and a reduced ability to interact with lower layers of the protocol stack.

This paper presents and evaluates an application-layer overlay solution that addresses data security requirements of applications on backward secrecy (a newly joined member cannot access data transmitted before the member joined) and forward secrecy (a departing member cannot access data that is transmitted after the member left). We present a key management and encryption method, called *neighborhood key method*, where each node shares a secret key only with its neighbors in the overlay network. The neighborhood key method avoids network wide re-keying operations, without requiring that payload data be re-encrypted at each hop.

The paper also considers an overlay routing protocol for ad-hoc networks that organizes nodes in a spanning tree topology and evaluates its performance with the proposed security scheme. Since mobile nodes exchange data only with neighbors in the spanning tree and since the neighborhood key method only requires security associations between neighbors in overlay topology (here: the spanning tree), a node can securely exchange data with all other nodes, while only maintaining security associations with its upstream and downstream neighbors in the spanning tree.

This paper is the first study that presents systematic empirical measurement data showing the performance of application-layer ad-hoc networking, with and without security, on commercially available portable wireless devices (PDAs). The remainder of the paper is structured as follows. In Section 2 we discuss an overlay software system that is the basis for the protocol implementation presented in this paper. In Section 3 we present the neighborhood key method. In Section 4 we present a tree-based ad-hoc routing protocol. In Section 5 we present experiments with mobile nodes that measure the performance of the routing protocol from Section 4 combined with the security mechanisms of Section 3. We provide brief conclusions in Section 6.

2 Overlay Performance in Ad-hoc Networks

In this section we present an empirical evaluation of the delay and throughput performance of overlay networks in a static-ad network of handheld wireless devices without security mechanisms. All routing and security protocols presented in this paper are realized in an open source software system for application-layer overlay networks called HyperCast [1, 14]. We provide a brief description of the HyperCast software architecture. Then we present the results of measurement experiments that give insights into the baseline performance of an application-layer approach in an ad-hoc environment.

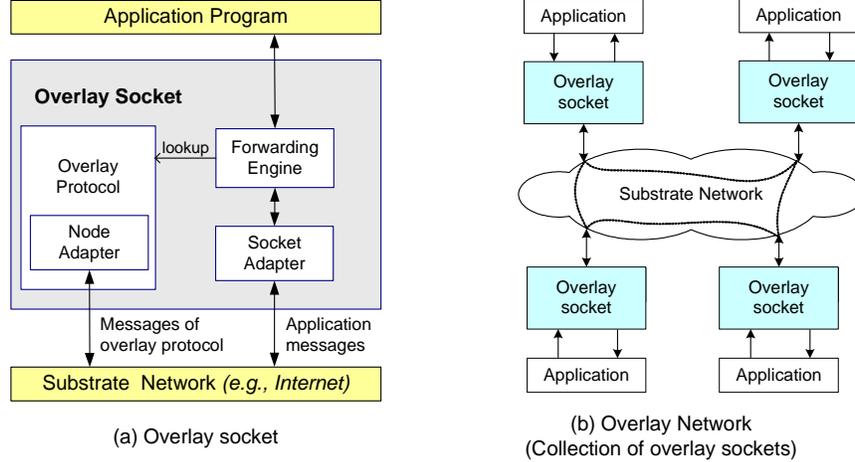


Figure 1: Components of overlay sockets.

2.1 The HyperCast Overlay Software System

The HyperCast software uses the concept of an overlay socket as an endpoint of communication in an overlay network. Application programs send and receive data through an overlay socket. An overlay socket attaches to an underlying substrate network that permits point-to-point or point-to-multipoint communication with other overlay sockets. An overlay network is a collection of overlay sockets that are organized in a logical topology and that are attached to a common substrate network. Each overlay socket performs functions that maintain its participation in the overlay topology without involvement of the application program. An overlay socket is configured with attributes that specify the name of the overlay network to be joined, the method to join the overlay network, 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.

In Figure 1(a) we show the main components of an overlay socket and their interactions. An application program uses the application programming interface (API) of the overlay socket for sending and receiving messages. The overlay socket has two interfaces to the substrate network, one for application data and one for control information.

Each overlay socket has an *overlay protocol* component responsible for establishing and maintaining the overlay network topology. The overlay protocol performs the functions of an application-layer routing protocol. It exchanges protocol messages with other overlay sockets for the purpose of building an overlay topology and setting up next-hop forwarding information for application messages. When overlay sockets are running over an infrastructure substrate network (e.g., the Internet), the overlay protocol is used to build a mesh overlay topology. In the context of ad-hoc networking, the overlay protocol executes an ad-hoc routing protocol. In this paper, we consider a routing protocol that establishes a spanning tree topology.

The *forwarding engine* component is responsible for sending, receiving, and forwarding formatted application messages in the overlay network. The forwarding engine performs the functions of an application-level router. When an application program transmits a message, the forwarding engine performs a lookup in the routing table that is managed by the overlay protocol to deter-

mine how to forward the message. When a message is received from the substrate network, the forwarding engine determines if the local socket is a destination of the message. If so, it passes the message to the application program. The forwarding engine also determines via a lookup if the message must be forwarded to another overlay socket.

The components used to access the substrate network are called adapters. If the substrate network is the Internet, then the adapters build and maintain associations via UDP or TCP. Each overlay socket has two adapters: a *node adapter* and a *socket adapter*. The node adapter handles messages of the overlay protocol. The socket adapter transmits and receives messages with application data. Having separate interfaces to the substrate network for application messages and protocol messages reflects a separation of the control plane and the data plane in the design of the overlay socket. Messages of the security protocol, e.g. for authentication and key exchange, to be discussed in the next section take advantage of this separation and will be transmitted on the control plane using the node adapter.

2.2 Measurements in a Static Ad-hoc Network

We now present measurement experiments that evaluate the performance of overlay networking with the HyperCast software 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. All PDAs run a single application program, that sends and receives messages over a single identically configured overlay socket. All experiments in this paper utilize this setup.

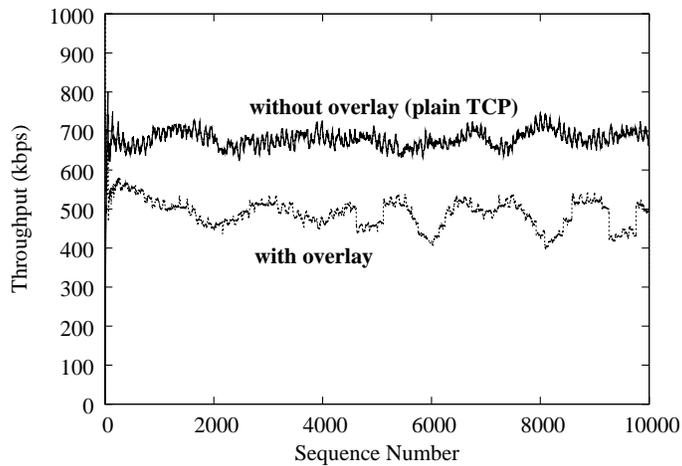
The experiments in this section evaluate the throughput and delay performance of a static ad-hoc overlay network built with off-the-shelf PDAs. The results provide a base-line scenario that can be compared to results that consider mobile nodes or security functions.

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

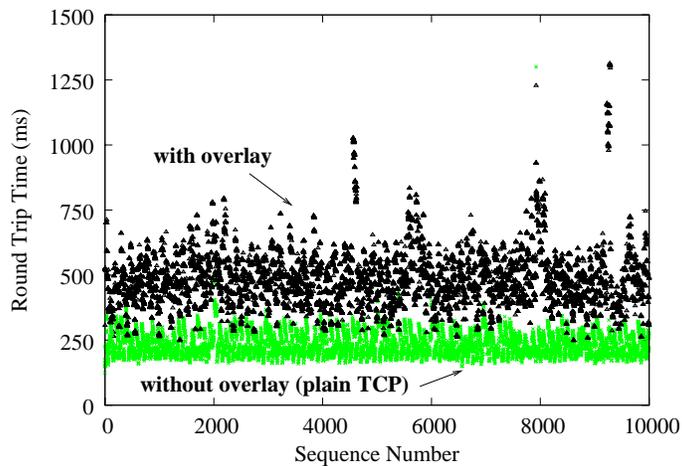
We use TCP connections over an IP network as substrate network (i.e., the overlay sockets are configured with socket adapters that establish TCP connections for transmitting application messages). With this choice, the flow and congestion control algorithms of TCP settle the transmission rate of the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

The results of this experiment are shown in Figure 2. Figure 2(a) depicts the achieved throughput of the data exchange calculated at the receiver by computing the number of messages received over a sliding window of 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.



(a) Throughput.



(b) Round-trip delay.

Figure 2: Single-hop measurements.

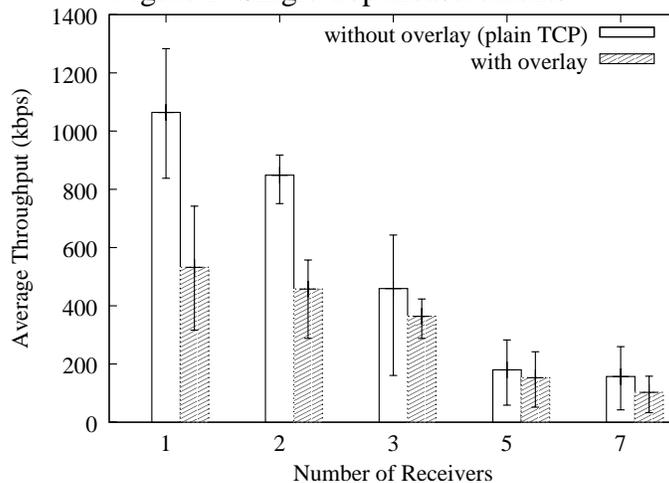


Figure 3: Single-hop measurements (Multicast).

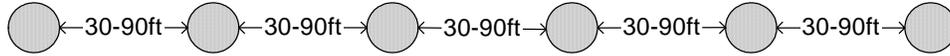
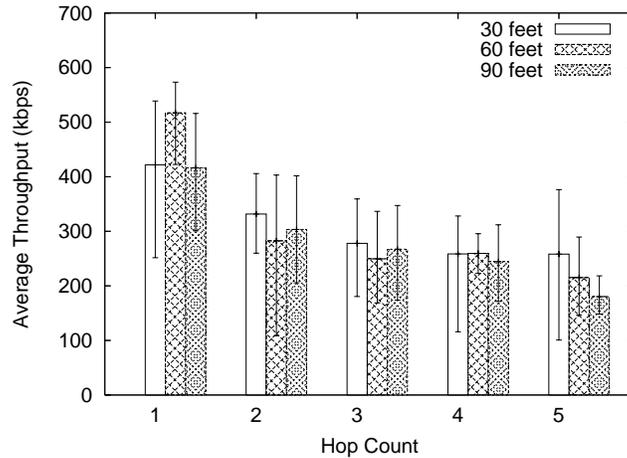


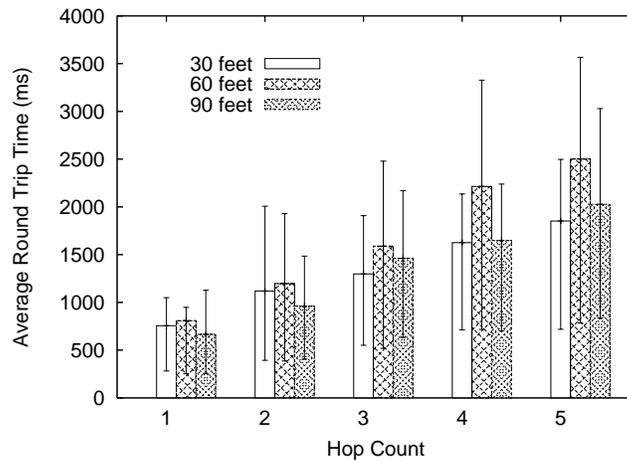
Figure 4: Six PDAs in a line topology.



Figure 5: Outdoor experimental setup with PDAs.



(a) Average Throughput.



(b) Average round-trip delay.

Figure 6: Multi-hop measurement results.

Single-Hop Measurements (Multicast). We next repeat the single-hop TCP experiment with multiple (up to seven) receivers. In this experiment, a single sender transmits application data using a multicast send operation of the overlay socket API. This send operation is translated by the overlay socket into multiple unicast TCP transmissions to each receiver. 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 (hence, roundtrip delay measurements cannot be presented).

Figure 3 shows the average throughput of all data transmissions, 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. Note that the improved results in Figure 3 for one receiver over those in Figure 2(a) are due to the fact that we do not send acknowledgments in this experiment.

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. 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. The leftmost PDA in Figure 4 is the sender. 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(a) depicts the average throughput values when increasing 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. We attribute the fact that the performance sometimes seems to improve when the distance is increased to the limitations of running wireless experiments. Since the battery lifetime of the PDAs limits the number of experiments that can be conducted at one time, the measurements were done over a period of several days. However, as has been observed elsewhere [2, 7], conducting identical wireless measurements at different times of the day or under different weather conditions may have widely varying outcomes. Figure 6(b) shows the results of the average end-to-end packet delays, with error bars indicating the range of delay measurements. The results show that delays increase linearly with the number of hops. Each added hop adds in most cases between 200-500 ms to the round-trip delay.

3 Neighborhood Key Method

This section contains the main contribution of this paper. It presents a new key management and encryption scheme, called the *neighborhood key method*, that ensures integrity and confidentiality of application data in overlay networks. Even though our evaluation will focus on ad-hoc networks, the method can be applied to overlays connected to a network infrastructure. We note 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 schemes presented here adapt and use widely established security mechanisms, such as public-private keys, signed certificates, and per-message keys, and are subject to the strengths and weaknesses of these schemes, to construct a practical and implementable security solution for overlay-based ad-hoc networks.

The security mechanisms discussed in this section are implemented as a wrapper for the adapters of the overlay socket, effectively adding a layer between the adapters and the overlay protocol (see Section 2). This has the benefit that the security mechanisms are independent of the overlay protocols. Details of implementation issues of the security system can be found in [1].

In most approaches to secure group communications in overlays or network-layer multicast, nodes share a single symmetric group key for encrypting and decrypting messages [22, 21]. The group key is globally updated and distributed each time the group membership changes [21, 24, 27]. This is referred to as group re-keying. Since group re-keying can be complex and often assumes access to a common server, our solution for mobile ad-hoc networks resorts to a different solution. In our approach, each node keeps and manages its own symmetric key, called a *neighborhood key*, that it shares only with its immediate neighbors in the overlay network. As a result, re-keying becomes a local operation that involves only the neighbors of a node in the overlay network. The drawback of a straightforward implementation of local keys is that each message must be decrypted and re-encrypted each time it is forwarded. The proposed scheme avoids this pitfall by encrypting each message at the source with a message-specific key. The source encrypts the message key with the neighborhood key and appends the encrypted key to the message. In this way, only the message key must be decrypted and re-encrypted when a message is forwarded.

3.1 Authentication

The difficulty of authentication and trust management without access to an infrastructure with trusted third parties or intermediaries is well-documented [11, 20, 28]. Many solutions have been proposed, including advance dissemination of private keys for all node pairs, threshold cryptography approaches [16, 28], 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 using X.509 Version 3 format. We assume that each node has its own certificate and that this certificate has been signed by a trusted third party. Also, each node stores the certificates of one or more trusted third party. Note that without online access to certificate authorities, it is not feasible to perform trust revocation. While, in principle, a distributed authentication protocol, e.g., [16], could be used, the practicality of (mostly threshold cryptography based) distributed authentication schemes is not established at this time.

In our scheme, each node performs authentication independent of and without coordination with other nodes. An exchange and verification of certificates between neighbors in the overlay occurs only when needed. When a node receives a 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. These secret keys are used to encrypt or sign messages. Each node accepts protocol and application messages only from authenticated nodes.

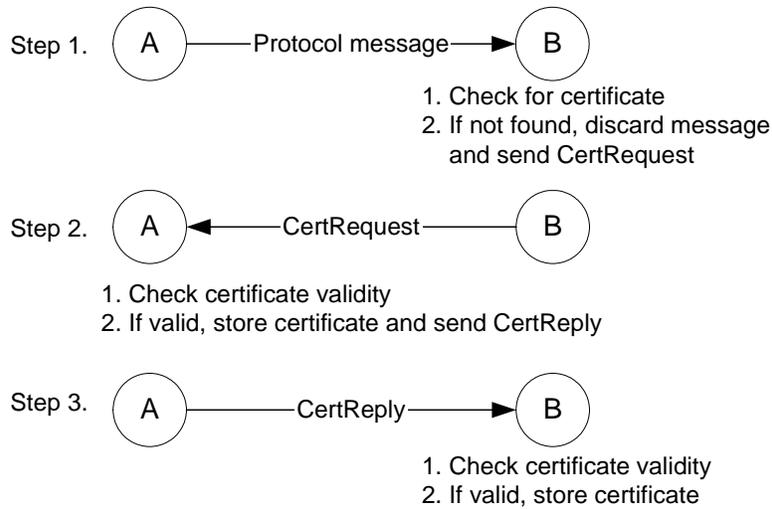


Figure 7: Authentication of nodes.

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 to B that includes 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 and application messages.

3.2 Exchange of Neighborhood Keys and Data Encryption

Encryption of data and the signing of hashes is done with symmetric keys. Each node maintains a single symmetric key, the *neighborhood key*, that it shares with its current neighbors in the overlay topology, and possibly also with potential future neighbors, and other nodes.

Whenever the set of (current or potential) authenticated neighbors changes, i.e., a new neighbor appears or an existing neighbor disappears, a node must compute a new neighborhood key and send this new key to all its 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 of the receiver is available due to the certificate that was exchanged during 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 se-

quence number;¹ or (5) the current neighborhood key has exceeded a specified maximum lifetime.

In Figure 7, *Key Update* messages would be exchanged immediately after the authentication is complete. That is, A sends a *KeyUpdate* immediately following the *CertReply* message (in Step 3), and B sends a *KeyUpdate* after it has verified the certificate contained in the *CertReply*. A node can also explicitly request the key from an authenticated neighbor, by sending a *KeyRequest* message. A *KeyRequest* message is transmitted whenever an 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 verify transmitted messages. Since a node 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 the neighborhood key method 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 each outgoing message (or, at the minimum, the message key for each outgoing message) be encrypted separately for each neighbor.

In comparison 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 at the cost of weakening forward and backward secrecy.

As an alternative, the HyperCast software system provides two variations of the neighborhood key method that can reduce the overhead due to key updates. These variations are discussed below.

In Figure 8, we illustrate two scenarios that will incur frequent key exchange operations in the neighborhood key method. In Figure 8(a) we depict an overlay network with three nodes (connected by lines), and a special node. Suppose that all nodes in periodically contact this node, e.g., to verify that the overlay network is not partitioned. If the the neighborhood key method is used in this scenario, the special node exchanges messages with all nodes in the overlay network. Thus, whenever the node membership changes, the special node needs to update and distribute

¹Every 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.

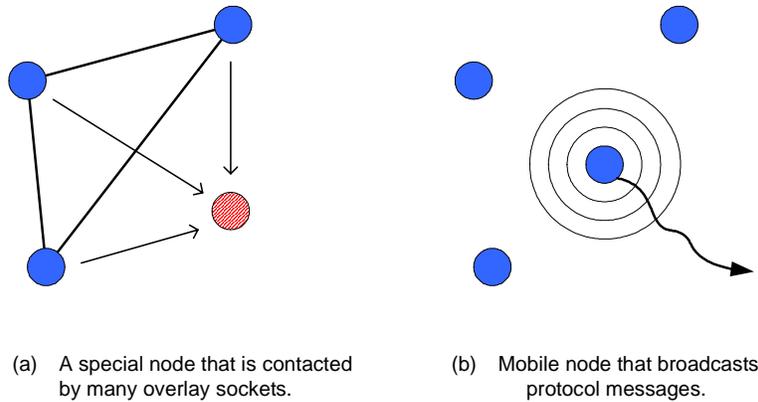


Figure 8: Scenarios with frequent updates of neighborhood keys.

its neighborhood key to all current members in the overlay network. This not only leads to a prohibitively high load at the special node. Also, each node in the overlay network must perform protocol operations when the membership of the overlay network changes.

In Figure 8(b) we show a scenario that may appear in a mobile ad hoc environment. The figure depicts a mobile node (the path is indicated by an arrow) that broadcasts protocol messages (indicated by circles around the transmitting node) that announce the presence of the mobile node to other nodes in its vicinity. With the neighborhood key method, whenever a node receives broadcast messages from a mobile node for the first time it must update its neighborhood key. When the number of mobile nodes is large and the mobility of nodes is high, the load due to updating keys in terms of traffic and computations may be significant.

Common to both scenarios in Figure 8 is that nodes may spend a lot of effort to update keys with its neighbors due to nodes that are not neighbors in the overlay network topology, e.g., the special node or nodes receiving broadcast messages. Since the primary role of the neighborhood key is the encryption or protection of application data, and only neighbors in the overlay network exchange application data, the effort spent with updating neighborhood keys may be reduced by distinguishing nodes that are neighbors in the overlay network from those that are not neighbors in the overlay topology and have separate key management for neighbors and non-neighbors. This leads to the following variations of the neighborhood key method.

- *Separate Neighborhood Keys for each Non-neighbor.* Here, each overlay socket has a neighborhood key for all neighbors in the overlay topology, and a separate key for each non-neighbor with which it communicates. The advantage of this method becomes evident in the scenario in Figure 8(a). An overlay socket does not need to communicate with the special node when its neighborhood changes in the overlay topology. The special node would need a separate key for each overlay socket in the overlay network. However, when an overlay socket joins or leaves, the special node need not update keys with any other node in the network. The drawback of this variation is that maintaining separate security associations with single overlay sockets precludes the use of broadcast operations in the substrate network.
- *Shared Group Key for all Non-neighbor:* In this method, each overlay socket has a neighborhood key for all neighbors in the overlay topology, and a single shared key for protocol

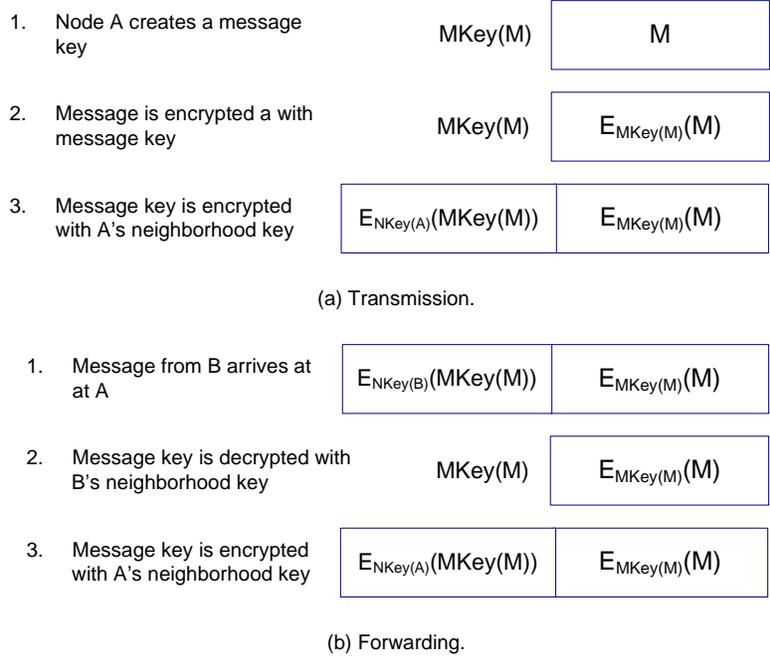


Figure 9: Processing an encrypted application message (M is the message, $MKey(M)$ is the message key for message M , $NKey(A)$ and $NKey(B)$ are the neighborhood keys of nodes A and B , $E_{MKey(M)}(M)$ is the message encrypted with the message key, $E_{NKey(B)}(MKey(M))$ is the message key encrypted with the neighborhood key of B).

messages sent to non-neighbors in the overlay topology. The shared key must be known to all overlay sockets in the overlay network. This method addresses the scenario of Figure 8(b) where protocol messages are transmitted in the substrate network with a broadcast transmission. When the broadcast message is signed with the shared key, the authenticity of the broadcast message can be verified without requiring the neighborhood key from the mobile node. The drawback of this method is that it has the same global re-keying requirements as the shared group key schemes discussed at the beginning of this section. With this variation of the neighborhood scheme, however, the shared key scheme is only extended to (some) protocol messages. Application payload is still protected with a neighborhood key, that is exchanged whenever the set of neighbors in the overlay network changes.

The above variations do not relax the requirement of a mutual authentication. Overlay sockets always drop messages received from unauthenticated sockets and initiate a certificate exchange. Also, the variations do not affect the transmission of overlay messages, since only neighbors exchange overlay messages and the variations do not change the security scheme for overlay messages.

3.3 Data Confidentiality and Integrity

In a straightforward implementation of 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.

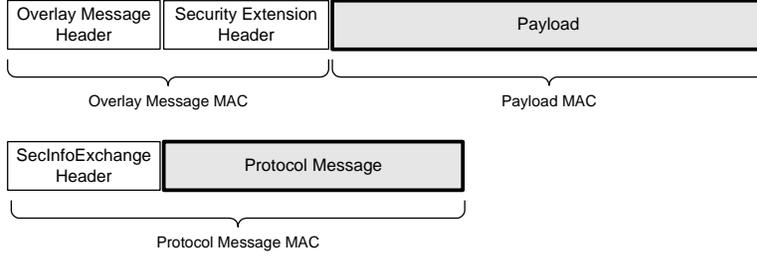


Figure 10: MAC Computation.

Clearly, this is very time-consuming and not practical in large networks. In the following, we present a solution that significantly reduces the encryption and decryption overhead incurred at each node. In our scheme, 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 this 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. This is feasible since each node holds the neighborhood keys of all its authenticated neighbors. If the message must be forwarded to another node, the node re-encrypts the message key with its own neighborhood key before transmission.

In Figure 9(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 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 9(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)². As shown in Figure 10 there are three different MACs. For application data, there is a MAC for the message payload (Payload MAC) and the message header (Overlay Message MAC). Both MACs are included in an extension header of the application message, called the Security Extension Header, which also includes the encrypted message key. The computation proceeds as follows. First, the payload MAC is computed using the message key, and added to the security extension header. Next, the message key is encrypted with the neighborhood key, as described earlier, and also added to the security extension header. Then, the neighborhood key is used to compute a MAC for the message header and the security extension header (not including the field for the overlay message MAC). Integrity is also provided for protocol messages. Here, a Protocol Message MAC is computed over the entire protocol message with the neighborhood key. Note that the MACs provides some level of route

²Precisely, we use a keyed-hash message authentication code (HMAC), which involves a cryptographic hash function in combination with a secret key.

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. Additional details on the computation on the security extension header and the MACs can be found in [1].

3.4 Evaluation of Security with Neighborhood Key Method

We evaluate the neighborhood key method in single-hop indoor measurements and multi-hop outdoor measurements, in the same setup as discussed in Subsection 2.2.

Latency of Authentication. We first show experiments that measure the latency incurred by the authentication process and the initial exchange of neighborhood keys, when new nodes join the overlay network. The measurements use the ‘plain’ neighborhood key method and does not take advantage of the optimizations discussed at the end of Subsection 3.2. Since the authentication process is driven by the operations of the overlay protocol, the delay of the authentication is dependent on the employed overlay protocol. The following experiments use the spanning tree protocol described in the next section. In the protocol, each node periodically broadcasts a single message, called Beacon message. The spanning tree protocol has a configuration parameter, called the Beacon time, that determines the frequency at which each overlay socket transmits a Beacon message. The frequency of the Beacon message determines the overall progress of the protocol. Increasing the Beacon times reduces the overhead of the protocol, while shorter Beacon times increase the ability of the protocol to react to changes of the network topology. The default value of the Beacon time is set to 1000 ms.

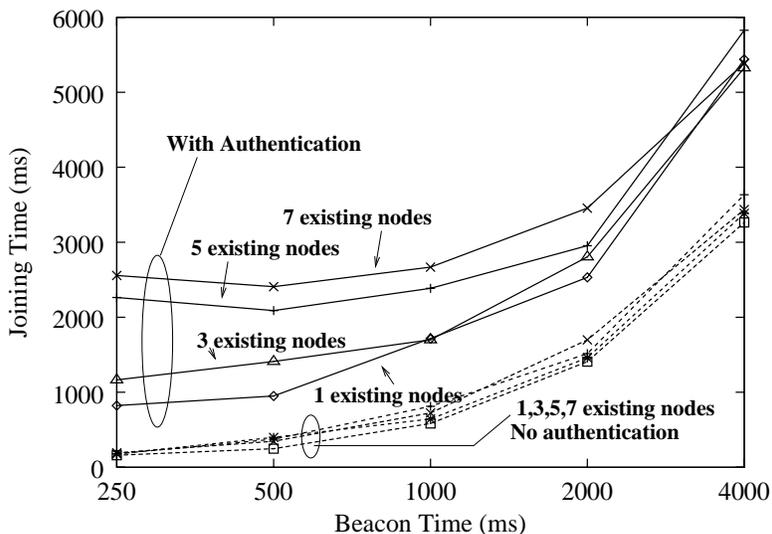


Figure 11: Delay of authentication for new nodes.

The experiment is conducted indoors with up to 7 PDAs located in a room. An experiment consists of simultaneously starting overlay sockets on 1, 3, 5, or 7 PDAs and measure the elapsed time until each PDA has found a neighbor in the topology. This time delay is called the Joining

time. Figure 11 shows the results of the experiment. Each data point depicts the average maximum Joining time for 5 repetitions of an experiment, where the Beacon time is varied between 250 and 4000 ms. (Note the log scale on the x-axis). The dashed lines show the results without the security scheme (labeled *No authentication*) and the solid lines show the results with the security scheme enabled (labeled *With authentication*). First, without the security scheme, the joining time is not very sensitive to the number of nodes that enter the overlay at one time. Also, note that the time to join is approximately equal to the duration of the Beacon time. With security enabled, the joining nodes exchange and validate certificates, and then generate and exchange neighborhood keys. Figure 11 shows that there is a noticeable dependence of the joining time on the number of existing overlay sockets when the beacon time is small. The difference diminishes as the Beacon time is increased. This is explained by the fact that the authentication process is triggered by receiving protocol messages (in this case, Beacon messages) from non-authenticated nodes.³ When the time between Beacon messages is small, an overlay socket receives many messages in a small amount of time. If initially, each message results in an exchange of certificates, the PDAs experience a higher load and longer delays when the number of existing nodes is large. When the Beacon time is increased, the certificate exchanges are spread out over longer periods of time, without overloading the PDAs. Consequently, the joining time is less dependent on the number of nodes already in the overlay network.

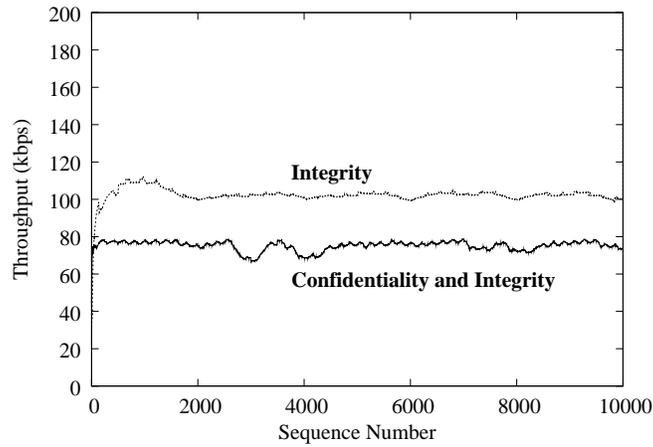
Single-Hop Measurement with Security. The experiment has the same setup as the ‘Single Hop (Unicast)’ experiment in Section 2. We again measure the throughput and delay performance of a static application-layer ad-hoc overlay network. We measure the performance of two security settings, labeled *Integrity* and *Confidentiality and Integrity*. 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 9. 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 12. A comparison with the measurements without security from Section 2 shows that there is a noticeable loss of performance due to encryption/decryption of messages and message keys. The results also reflect that the operations performed for integrity and confidentiality are quite similar. The additional cost of data confidentiality is limited. This should not be surprising since the only difference consists in encryption and decryption of the payload at the sender and the receiver, respectively. All other computations, such as the generation and encryption/decryption of message keys with the neighborhood keys, are already needed for the integrity mechanism.

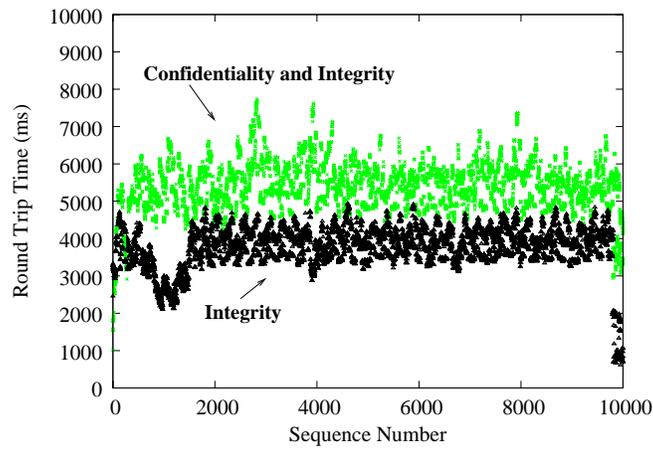
Multi-Hop Measurements with Security. In Figure 13 we present throughput results of a multi-hop outdoor experiments, where PDAs are set up as shown in Figure 4, but with security level set to *Confidentiality and Integrity*. The values present the average throughput values over the duration of the experiment. 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 with an implementation that does not require a Java virtual machine.

³Recall that using the second variations of the neighborhood key method would avoid this problem.



(a) Throughput.



(b) Round-trip time.

Figure 12: Single-hop measurements.

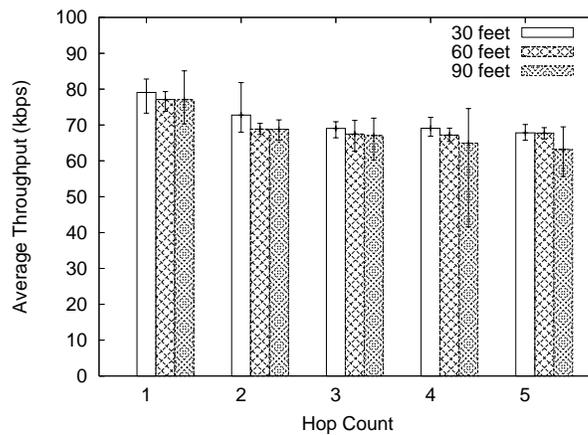


Figure 13: Multi-hop throughput with neighborhood keys (*Confidentiality and Integrity*).

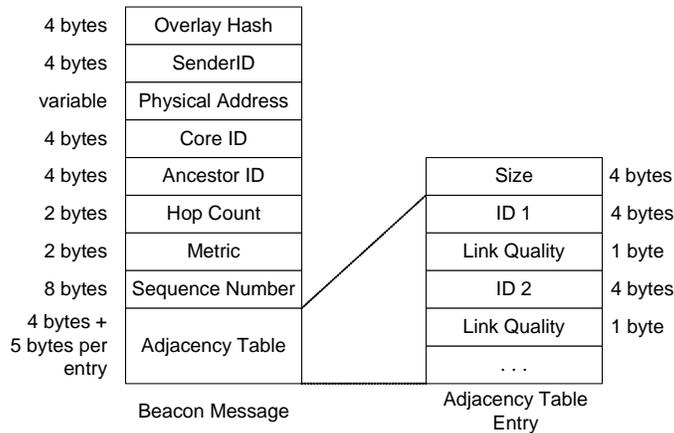


Figure 14: Format of a beacon message.

4 A Spanning Tree Protocol for Ad-hoc Networks

So far we have considered a static ad-hoc network environment. In this section, we describe a protocol that can establish and maintain an overlay network in an environment with mobile nodes. We consider a protocol that maintains a spanning tree topology, and refer to the protocol as *Spanning Tree Protocol* or *SPT* protocol. The protocol assumes that the substrate network supports a broadcast operation for sending protocol messages. The target environment for the SPT protocol is a mobile ad-hoc network, however, the protocol can also be used over non-wireless substrate networks.

Application data is forwarded between neighbors in the spanning tree topology. Recall that the neighborhood key method automatically establishes security associations with neighbors in the overlay network. Therefore, after the spanning tree is built, a node can transmit data securely without incurring delays due to establishing a security association with each next hop neighbor.

The protocol has been implemented as an overlay protocol component of the overlay socket in Figure 1. The overlay socket is configured with a node adapter that can send and receive UDP multicast messages. 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., [19], the SPT protocol adapts Perlman's spanning tree algorithm for bridges [18] to a wireless environment. 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. Each 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 an *adjacency table* which contains the list of all nodes with which it can exchange messages.

In the SPT protocol, each node periodically broadcasts a *beacon message* in the substrate network. The information in the beacon messages is used to (1) accomplish a rendezvous of a new node with an existing spanning tree network, (2) learn about the reachability of other nodes, (3) eliminate asymmetric links, (4) agree on a single node as the root of the tree, (5) select an ancestor in the tree, and (6) 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 14. The first field, the *overlay hash* contains a checksum computed over the overlay network identifier. Nodes use this hash to detect protocol messages from other overlays, which are then 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. The format and length of the physical address is determined by the type of substrate network. If the substrate network is an IP based 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.⁴ 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 a node selects an ancestor. In Section 4.2 we discuss two 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.

4.1 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 [2]. 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 [4, 12, 15].

Alternative approaches, e.g., as proposed in [5, 7, 25], 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 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.

⁴Note that the spanning tree protocol for bridges does not specify an ancestor, but its role corresponds to that of the root port of a bridge.

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

4.2 Ancestor Selection

Each SPT node uses the beacon messages received from adjacent nodes to select its ancestor in the spanning tree. In the SPT protocol, we have realized several variants of an ancestor selection algorithm. Each version of the algorithm creates a spanning tree, yet with different properties.

We use $A.Sender$, $A.Core$, $A.Ancestor$, $A.HopCount$, and $A.Metric$ to denote the contents of the corresponding fields in the beacon message sent by node A . The beacon messages by node A are also referred to as A 's beacon message. Initially, each node A believes that it is the core and sends a beacon message A with

$$\begin{aligned} A.Sender &= "A", \\ A.Core &= "A", \\ A.Ancestor &= invalid, \\ A.HopCount &= 0, \\ A.Metric &= see below . \end{aligned}$$

Since A is the core, it sets the address of its ancestor to an invalid address. In the SPT protocol, '0' is defined to be an invalid node identifier.

When node A receives a message M from some other node, A determines if the message advertises a better ancestor. If the core advertised in the message has a numerically smaller identifier than the core of the node, i.e., $M.Core < A.Core$, then $M.Sender$ is a better ancestor for A . When A changes its core, the fields of its beacon message are changed as follows:

$$\begin{aligned} A.Core &= M.Core, \\ A.Ancestor &= M.Sender, \\ A.HopCount &= M.HopCount + 1, \\ A.Metric &= see below . \end{aligned}$$

This rule ensures that each node selects an ancestor that advertises the core with the smallest identifier.

If the core in the received message is identical to A 's core ($M.Core = A.Core$), A compares $M.Metric$ and $A.Metric$ and determines if it can find a better ancestor. Each node tries to minimize the value of the metric. If the metric advertised in the messages improves upon A 's metric by at least a threshold value $Thold$, i.e., $M.Metric \leq A.Metric + Thold$, then A switches its ancestor to $M.Sender$. We refer to $Thold$ as the *jumping threshold*. With the jumping threshold, it is feasible to tune how quickly a node changes an ancestor. It is desirable to set the jumping threshold such that a node selects a new ancestor only if there is a significant improvement to the given metric. When A changes its ancestor, the fields of its beacon message are changed as follows:

$$\begin{aligned} A.Ancestor &= M.Sender, \\ A.HopCount &= M.HopCount + 1, \\ A.Metric &= see below . \end{aligned}$$

The value of the metric depends on the selection criteria discussed below. This will be discussed next.

Minimum Hop Count to Core. Here, each node selects an ancestor that minimizes the path length to the core. Each node sets the metric to the value of the HopCount field. A node that has

determined that it is a core sets the value of its metric to 0. A node A switches its ancestor if it receives a beacon message M such that $M.Metric + Thresh \geq A.Metric$. We generally set the jumping threshold to $Thresh = 2$, hence, a node changes ancestors whenever it reduces the hop count by at least one. In this case, node A sets the values for its own beacon messages to

$$A.Metric = M.Metric + 1 .$$

It is apparent that this procedure realizes a minimum path to the core.

Path Quality to Core. This metric a node in the network 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 $L = \{1, 2, \dots, K\}$ denotes a set of links that form a path in the network and is p_i ($0 \leq p_i \leq 1$) 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 (This introduces an assumption that link quality at links is independent). 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 to 1. Also, the values in the Metric field are scaled to have good accuracy in the relevant range. When a node A receives a message M that advertises a better ancestor ($M.Metric \leq A.Metric + Thold$), it sets the metric to

$$A.Metric = M.Metric - \log (LQ(A, M.Sender)) ,$$

where $LQ(A, M.Sender)$ is the bidirectional link quality between node A and the sender of the message. Since the link quality is in the range $0 \leq LQ(A, M.Sender) \leq 1$, its logarithm is negative. Due to the additional restriction that the link quality of a node to its parent must be above a specified threshold, issues with large values of the metric when the link quality is zero or close to zero, do not arise in praxis. puts a limit to the largest value of the metric.

It is apparent that the above steps maximize the product of the link qualities of the path to the core. With an additive metric, i.e., the metric of a path is the sum of the metrics of the links, and an objective to minimize the path metric, the algorithmic properties are identical to those in the minimum hop metric.

Note that our construction of routing metrics permits to combine hop count and path quality into a single metric, where each component is assigned a weight. Specifically, if H denotes the number of hops to the core and $\sum_{i \in L} \delta_i$ is the path quality to the core, we could define a metric $\alpha H + \beta \sum_{i \in L} \delta_i$, with two parameters $\alpha > 0$ and $\beta > 0$, which distributes the weight of the hop count and the path quality according to values of α and β .

The spanning tree established by the SPT protocol is suited for forwarding application messages to a specific destination (unicast) or to all nodes (multicast). We omit a discussion of additional mechanism provided for forwarding of data, including, avoiding the count-to-infinity problem, population of a forwarding table for unicast routing, etc, and refer to the protocol specification [6] for complete details.

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	<i>Uniform[0, max. speed]</i>
Pausing Time	20 sec
Transmission mode	unicast

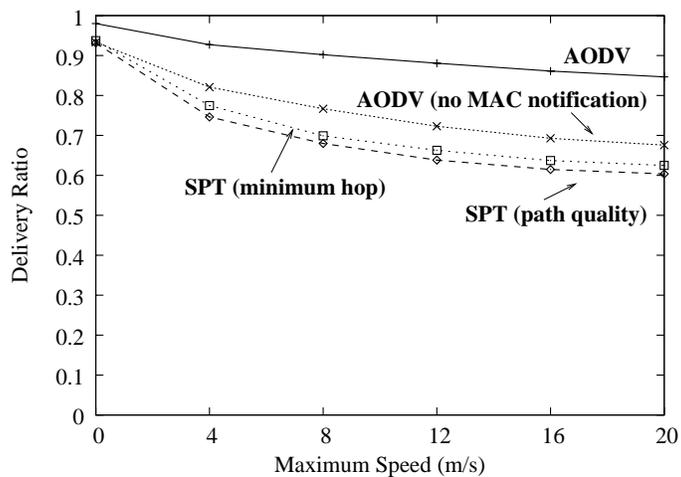
Table 1: Simulation parameters.

4.3 Evaluation of the SPT Protocol

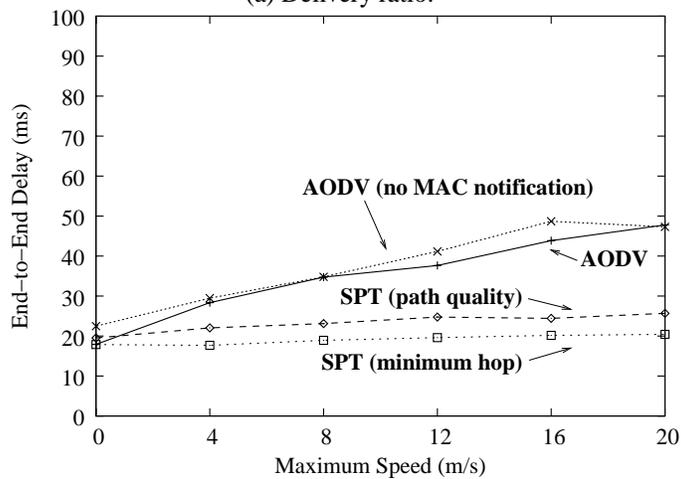
We evaluate the spanning tree protocol in simulation experiments with the Glomosim [26] simulator for ad-hoc networks. The parameters of the simulations, shown in Table 1, are typical for simulations in the published literature that evaluate ad-hoc routing protocols. There are 50 mobile nodes and 10 members transmit unicast packets to 10 receivers at a constant rate.

We compare the two versions of the SPT protocol (hop count and path quality) to the AODV protocol [17], which is one of the major 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 follows 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 also consider a modification to AODV where we disable the MAC notification for unavailable links. In this way, we can see how an application-layer version of AODV may perform that does not have access from lower layers of the protocol stack. The SPT protocol which proactively establishes a routing topology and the AODV protocol which builds routes in a reactive fashion (‘on demand’) can be seen as representing different ends of the design space for ad-hoc protocols.

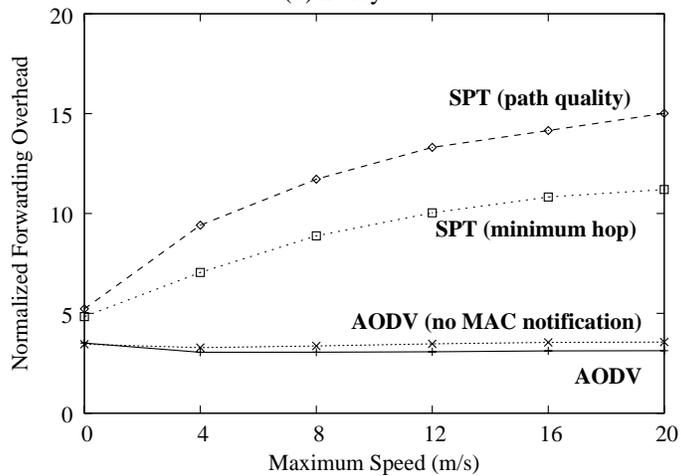
We consider the following performance metrics, which are frequently used to evaluate ad-hoc routing protocols. The *Delivery Ratio*, defined as the fraction of transmitted packets that are successfully delivered to the destination, measures how well a protocol finds routing paths in a network with mobile 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.



(a) Delivery ratio.



(b) Delay.



(c) Forwarding overhead.

Figure 15: Simulation of mobile network.

Figure 15 depicts the performance measures as a function of increasing mobility of nodes. Each data point represents the average of ten simulation runs, Figure 15(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.

The graph in Figure 15(b) shows that the SPT protocol has 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 15(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. As mentioned earlier, this observation is typical for simulation studies of ad-hoc routing protocols, and is generally not corroborated in empirical measurement studies. In the next section, we will see that this is confirmed by our own 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.

5 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 16. 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.

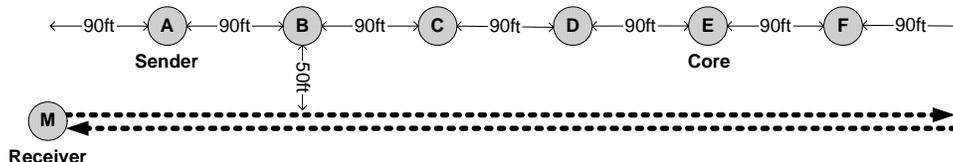
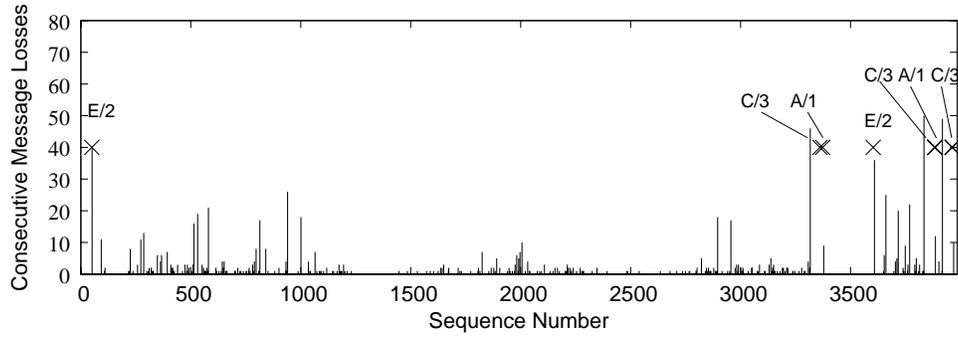
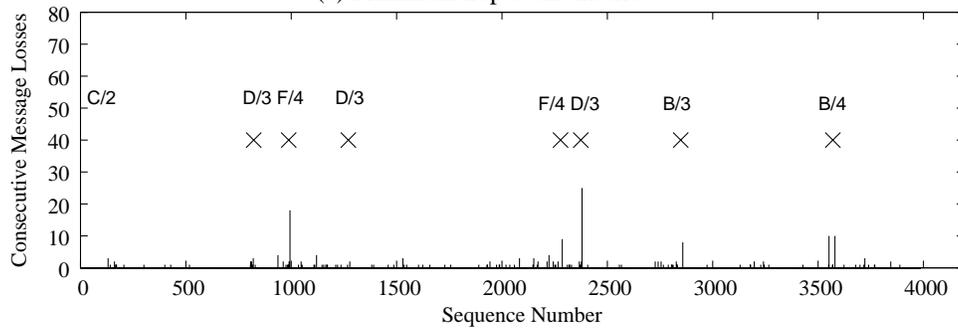


Figure 16: Measurement scenario with a mobile receiver.

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.

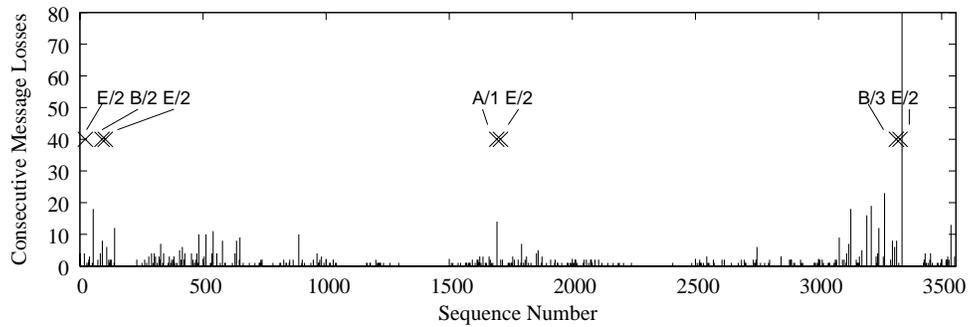


(a) Minimum hop count metric.

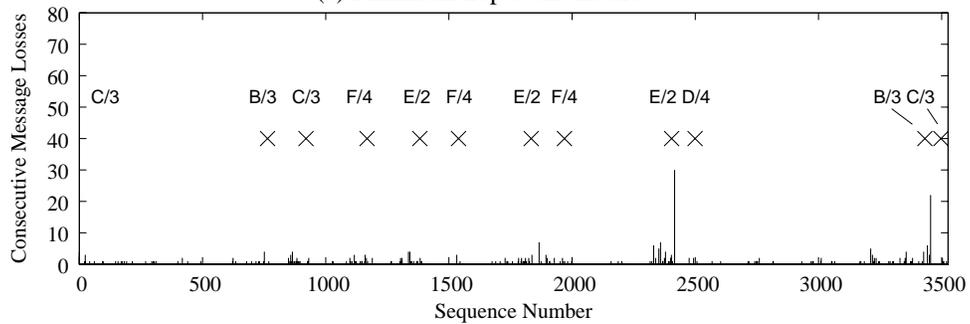


(b) Path quality metric.

Figure 17: Measurements with mobile receiver (with *Confidentiality and Integrity*).



(a) Minimum hop count metric.



(b) Path quality metric.

Figure 18: Measurements with mobile receiver (confidentiality enabled).

Figure 17 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 from 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 17(a) and 17(b) shows that the path quality metric has significantly fewer losses.

In Figure 18 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 17 and 18 shows that the security mechanisms only cause a limited degradation of the recorded message losses.

6 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] Hypercast design documents and materials. <http://www.comm.utoronto.ca/hypercast/material.html>, 2005.
- [2] 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.
- [3] K. Chen and K. Nahrstedt. Effective location-guided overlay multicast in mobile ad hoc networks. *International Journal of Wireless and Mobile Computing, Special Issue on Group Communications in Ad Hoc Networks*. To appear.
- [4] 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.
- [5] 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.
- [6] G. Dong and J. Liebeherr. Specification of the SPT protocol. <http://www.comm.utoronto.ca/hypercast/material.html>, 2005.

- [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, 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.
- [15] 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.
- [16] 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*, 13(6):1049–1063, Nov./Dec. 2004.
- [17] 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.
- [18] 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.
- [19] S. Radhakrishnan and G. Racherla. Protocol for dynamic ad-hoc networks using distributed spanning trees. *Wireless Networks*, 9:673–686, 2003.
- [20] 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.

- [21] 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.
- [22] D. Wallner, E. Harder, and R. Agee. Key management for multicast: Issues and architectures. IETF RFC 2627, June 1999.
- [23] L. Xiao and et. al. Prioritized overlay multicast in mobile ad hoc environments. *IEEE Computer*, 37(2):67–74, Feb 2004.
- [24] 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.
- [25] 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.
- [26] 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.
- [27] 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.
- [28] L. Zhou and Z. J. Haas. Securing ad hoc networks. *IEEE Network*, 13(6):24–30, Nov./Dec. 1999.