Verifying Delivered QoS in Multi-hop Wireless Networks

Suresh Singh, *Member, IEEE,* and Tom Shrimpton, *Member, IEEE*

***Abstract*— The question we consider here is the following: "How can a source verify the QoS (Quality of Service) experienced by its packet(s) at each hop to the destination in a multi-hop wireless network?" For example, if Bob needs to forward packets within some maximum delay of δ_B, how can the source verify that Bob did in fact forward the packets within this bound? Answering this question will enable innovations in multi-hop wireless network deployments where nodes may receive payment not only for forwarding packets but also for meeting some QoS guarantees. In this paper we present protocols that enable verification of delivered QoS for individual packets as well as verification of statistical QoS for groups of packets. The protocols are proved to be cheat-proof. We also provide expressions for the minimum verifiable delay.**

***Index Terms*— C.2.8.a Mobile computing algorithm/protocol design and analysis, C.2.1.k Wireless communications**

I. Introduction

The problem of dealing with non-forwarders in multi-hop wireless networks has received a great deal of attention recently with researchers exploring various ways to encourage and/or force users to participate fully in such networks. Proposals have ranged from penalizing non-forwarders to compensating users for the cost of forwarding [11], [2], [5], [13]. Other work has addressed the issue of cheating and payment in such schemes [17]. All these solutions, however, *cannot enforce or verify timely packet delivery*, they only guarantee eventual delivery. This has problems not only for real-time applications but also for non real-time applications where response time is affected. We believe that timely delivery of packets is important and *timely delivery ought to be linked with payment*. For example, if Bob promises Alice that he will forward packets within some maximum time δ_B, then there needs to be some reliable mechanism whereby Alice can verify that Bob kept his word before she pays him. Likewise, Alice should also be held to her word to pay Bob if he forwarded her packet within δ_B. A question that may be raised here is what would motivate Bob to delay sending Alice's packet in the first place (and hence why worry about verifiability)? The answer is simply that Bob may want to maximize his revenue by agreeing to forward packets for many connections. Then, due to the resultant congestion, it takes a long time for Bob to forward them. In this case, clearly, Bob should not be paid. However, Bob can claim that nodes further downstream delayed the packets and he is being unfairly penalized. Thus, there is a need to have a reliable mechanism to verify the delivered delay and other QoS *per-hop* so as to pay or penalize nodes appropriately. In the above example, if we have a way to verify per-hop QoS, Bob would be penalized for delaying packets and this in turn would alter Bob's behavior.

The authors are with the Department of Computer Science, Portland State University, Portland, OR 97207.

In this paper we present three protocols to verify delivered QoS. As a first step, we prove an important impossibility result which says that without certain system-level assumptions it is impossible to verify per-hop QoS. Based on the insight provided by this proof, we make appropriate system assumptions and present two protocols that can verify per-hop QoS. The first protocol (section VI) gives us per-hop per-packet delay for each packet in a flow (computing jitter and loss is then a trivial matter). The second protocol (section VII) gives per-hop average and maximum delay for a flow. This second protocol is more message efficient and thus more attractive. In section VIII we remove all system assumptions and present a third protocol, which relies instead on a particular form of payment to force honest node behavior. In comparing the protocols we note that the first two protocols are attractive since payment is in no way tied to correct protocol function but they require some special conditions to work (most notably, the presence of a trusted computing resource at each node). The third protocol works without any system assumptions but restricts us to a specific form of payment. The other main results in the paper include proofs of cheat proofness (section VI-E) and computation of the minimum verifiable delay per hop (sections VI-F and VII-A).

A. Related Work

Recently, several authors have started looking at the problem of pricing in ad hoc networks to motivate users to forward packets for other users. The problem addressed by previous papers deals with the twin challenges of convincing users to forward packets for others and to ensure that users do so honestly. [6] proposes using a virtual currency called *nuglets* as payments for packet forwarding. The sender loads some nuglets in a packet before sending it. Intermediate nodes acquire some nuglets from the packet before forwarding it. If the packet runs out of nuglets, it is dropped. There are obvious frauds that can be perpetrated and therefore the requirement is that each node have tamper-proof hardware. Other papers [10] by the same research project called Terminodes study this approach and some variations.

[5] takes a different approach to the problem. Here, each node has a *reputation* which is a numeric value. A node with a poor reputation (negative value) will have its packets rejected while nodes with a positive reputation forward each others packets. A node gains a negative reputation when it

fails to forward packets (this is easy to determine if nodes can overhear forwarding transmissions). A node with a negative reputation can slowly build up its reputation by forwarding packets. However, reputation is lost quickly and is built up slowly. Some problems with these systems are noted in [17] including vulnerability to attacks where several selfish nodes collude. Furthermore, the system depends on the ability of nodes to overhear transmissions.

[17] describes a credit-based system for implementing cooperation between nodes. Unlike the nuglets idea, [17] does not require tamper-proof hardware. The system requires a Credit Clearing Service (CCS) in the internet (much like our CA). Nodes collect receipts for packets they have forwarded and claim the money whenever they get connected to the CCS. The sender pays for the transmission of the packet using credit that it has previously purchased from the CCS. Each node on the path derives a receipt based on message contents. When nodes submit their receipts to CCS for reimbursement, the CCS can easily determine if a particular message made it all the way to a destination. The paper describes mechanisms whereby colluding nodes can cheat (including a cheating sender who does not want to pay) and then describes an appropriate payment strategy for each hop which is such that the optimal strategy for each node is to not cheat. [14] describes a similar mechanism to stimulate collaboration in multi-hop cellular networks where a base station cannot cover a region and relies on multi-hop ad hoc networks to extend coverage.

[2] presents a routing mechanism for ad hoc networks with selfish agents. In the model, each node i has a cost c_i for forwarding packets (this may be a function of remaining battery power, for instance). The cost of a link (i, j) is then $c_i P^{min}_{i,j}$ where $P^{min}_{i,j}$ is the minimum power needed to transmit from i to j. The goal of the ad hoc VCG (Vickery, Clarke, Groves) routing protocol is to obtain a lowest cost path from source to destination. The problem lies in a node i misreporting c_i and/or $P^{min}_{k,i}$. The paper employs a form of the Vickery auction to ensure that the dominant strategy for nodes is to be truthful about c_i and $P^{min}_{k,i}$.

Unlike the previous approaches, [15] employs a cryptographic approach. The routing protocol AD-MIX presented is such that nodes are forced to route all packets for fear of losing packets destined to them. The idea is to select ≥ 2 intermediate nodes and successively encrypt packets with the public key of each of these nodes (called poles). When a packet passes through a pole, it is decrypted by the private key of that pole. Now, by ensuring that some percentage of packets meant for destination D are routed via D to a pole before being routed back to D will force D to truthfully forward all packets because it does not know if a particular (encrypted) packet it needs to forward is meant for it or not.

Our work here differs from this prior work in a fundamental way – our protocols provide verifiability of delivered QoS and can thus be used as the basis for developing novel pricing models for stimulating cooperation in wireless networks.

II. Model and Assumptions

In this paper we assume that all nodes in the system, including the senders and receivers of all connections, are untrustworthy. Specifically we assume that all nodes are *greedy* but *rational* — i.e., the nodes will not do anything that reduces their own revenue (these terms are formalized in section VI-E). Given this, our goal is to develop protocols whereby the QoS provided for a connection can be *verified*. Since the focus in this paper is only on verifying that negotiated QoS parameters were met, we do not develop protocols either for negotiating QoS or for implementing QoS routing. Specifically, we assume that for a sender/receiver pair to set up a connection, there exists a protocol whereby QoS is negotiated and unforgeable contracts signed by all involved nodes. Creating such unforgeable contracts can be done by using one of many different methods. One simple way is for all messages exchanged during the negotiation to include the sender's signature (see section IV). The set of these messages then forms the contract. For instance, say Alice sends a message containing QoS requirements for her connection to Bob (signed by Alice). Bob responds with an Ack (signed by Bob). These two messages now form the contract negotiated between Alice and Bob. Finally, we assume that there exist routing and other resource allocation protocols that enable nodes to provide QoS. The overall model now is as follows:

1) There exists a central authority (CA) that verifies if the negotiated QoS was met (as in, for example, [17]). All nodes involved in a connection create some form of *receipts* during the duration of the connection and make these available to the CA at some later time. These receipts provide enough information to the CA to determine the per-hop QoS. The CA may not always be reachable but is always eventually reachable. Of particular note is that *the CA does not participate in the forwarding protocol*. It only verifies QoS after the fact. The CA may be one or multiple entities. Note also that since nodes may forward their receipts to the CA at any time, congestion at the CA is not an issue. Indeed, one can imagine a user contacting the CA at the end of the day when she docks her laptop. Therefore, concerns such as the overhead of transmitting receipts to the CA or the flood of receipts at the CA are not significant.
2) A sender/receiver pair negotiates QoS for their connection using some protocol that results in unforgeable contracts. These are provided to the CA as part of the verification process.

For the purpose of discussion in this paper, we use the models shown in Figures 1(a) and (b). Here we look at three consecutive nodes on a path from some sender to some receiver. In Figure 1(a) δ_i denotes the negotiated per-packet delay at node i and in Figure 1(b), $\{\delta_i^{\text{avg}}, \delta_i^{\text{max}}\}$ denotes the average delay for the flow at node i and the maximum acceptable delay for any packet at node i. We assume that the transmission, propagation, and any other network protocol delay is included in the δ_i's.

Note that if we can verify the per-hop delay per-packet (Figure 1(a)), then it is an easy exercise to derive other QoS parameters such as loss and jitter. Since all nodes present their per-packet receipts to the CA, the CA can determine which packets were lost and at which node (because lost packets will

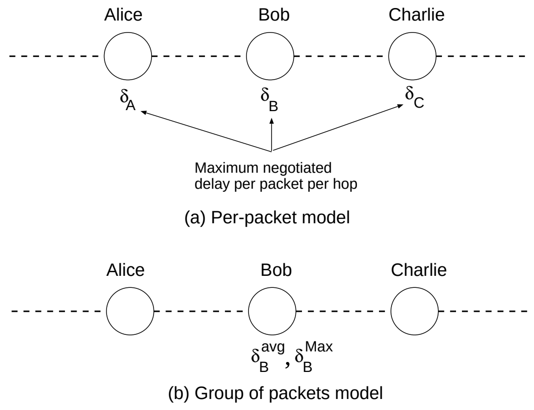

Alice
Bob
Charlie
δ_A
δ_B
δ_C
Maximum negotiated delay per packet per hop
(a) Per-packet model
Alice
Bob
Charlie
$\delta_B^{avg}, \delta_B^{Max}$
(b) Group of packets model

Fig. 1. QoS specification models.

not have receipts from nodes they did not visit). Likewise, per-hop jitter can also be computed using the per-hop verified delay.

III. Overview of the main approaches

Let us consider the general case where each node has its own time clock (unsynchronized with respect to all other clocks). Each node reports the time it received and transmitted each packet based on its local clock. In this scheme the times reported by each node can be completely arbitrary since we assume that nodes are greedy but rational cheaters. Thus, a node may arbitrarily delay forwarding packets but will report a very short delay. The challenge therefore is in (1) making nodes honestly report times and (2) making nodes forward packets in a timely fashion. We examine two approaches for achieving this goal. In the first, we utilize a global nonce source and a trusted computing resource at each node that allows us to get per hop per packet delays accurately. In the second approach we make no assumption of a global nonce source or the presence of a trusted computing resource. However, we use a specific form of payment schemes that makes it unprofitable to cheat.

While this latter approach is attractive in not requiring any special system level support, the fact that only specific payment schemes can be used is a big drawback. Ideally, the payment scheme should be left up to the discretion of the users of the wireless network and not tied to protocol behavior (except in so far as delivered QoS is concerned). This separation enables users and providers of wireless networks complete flexibility in building viable pricing models from a business perspective. In sections VI and VII we develop protocols conforming to the first approach and in section VIII we develop a protocol that requires special pricing schemes to work correctly.

IV. Notation

Our protocols require cryptographic operations to ensure correct behavior. This section provides a summary of the notation used in the remainder of this paper. Let $\mathcal{K}$ be a finite set, let $\mathcal{D} \subseteq \{0,1\}^*$ be a set of strings and let $n > 0$ be an integer. Let $\mathcal{H} = \{h_k : \mathcal{D} \rightarrow \{0,1\}^n\}_{k \in \mathcal{K}}$ be a hash function family that is collision-resistant [12].

A symmetric encryption scheme $\mathcal{SE} = (\mathsf{Key}, \mathsf{Enc}, \mathsf{Dec})$ is a triple of algorithms. The probabilistic algorithm Key is the (symmetric) key generation algorithm. The probabilistic or stateful algorithm Enc is the encryption algorithm; it takes a key K generated by Key and a plaintext string X to produce a ciphertext C. We denote this $C \leftarrow \mathsf{Enc}(K, X)$. The deterministic algorithm Dec is the decryption algorithm; it takes a key K generated by Key and a string to produce a string X that is a plaintext or distinguished symbol $\perp$, used to denote that the ciphertext is invalid for that key; we write $X \leftarrow \mathsf{Dec}(K, C)$. For proper operation we require that $M \leftarrow \mathsf{Dec}(K, \mathsf{Enc}(K, M))$.

An asymmetric (or public-key) encryption scheme $\mathcal{AE} = (\overline{\mathsf{Key}}, \overline{\mathsf{Enc}}, \overline{\mathsf{Dec}})$ is also a triple of algorithms. The probabilistic algorithm $\overline{\mathsf{Key}}$ takes a security parameter s (typically the key length) and produces a public-key/secret-key pair (pk, sk). The probabilistic encryption algorithm $\overline{\mathsf{Enc}}$ takes a public key pk and a plaintext string X to produce a ciphertext; we write $C \leftarrow \overline{\mathsf{Enc}}(pk, X)$. The deterministic algorithm $\overline{\mathsf{Dec}}$ takes a secret key sk to produce a plaintext or $\perp$; we denote this $M \leftarrow \overline{\mathsf{Dec}}(sk, C)$. For proper operation we require that $M \leftarrow \overline{\mathsf{Dec}}(sk, \overline{\mathsf{Enc}}(pk, M))$ for all M in the message space associated to public key pk. We assume that both $\mathcal{SE}$ and $\mathcal{AE}$ are semantically secure [8], [3], [4].

A signature scheme $\mathcal{SIG} = (\mathsf{Key}, \mathsf{Sign}, \mathsf{Verify})$ consists of key generation algorithm, for producing public-key/secret-key pairs, a signing algorithm Sign and a signature verification algorithm Verify. Sign takes a secret key sk and a string M to produce a signature, and we write this $\sigma \leftarrow \mathsf{Sign}(sk, M)$. The verification algorithm Verify takes a public key, a string and a purported signature for that string; if the signature is valid for that key the verification algorithm returns 1; otherwise it returns 0. We assume that $\mathcal{SIG}$ is unforgeable under a chosen-message attack [9].

V. Impossibility Result

As noted in section III, if nodes use their own local clocks to report receive/transmit times, there are very obvious frauds that can be perpetrated. Therefore, a reasonable question to ask is, if there is a global clock, can nodes still cheat? Unfortunately, the nodes can still report arbitrary old or future times for various events. Indeed, there is no difference between having a global clock or using local clocks.

Let us now make a stronger assumption. Say there is a global clock but nodes are unable to predict future time stamp values. Under this assumption does there exist a cheat-proof protocol for verifying per-hop QoS? We call this form of clock a *global nonce source* (NS) which can be heard by all nodes. It generates a sequence of unforgeable nonces that are unpredictable, one per clock tick. We call these values time stamps. The *resolution* of NS is assumed to be Δ. If $t_1 < t_2$ are two time stamps (i.e., nonces) then $f(t_2, t_1)$ denotes the number of clock ticks between these two values. These types of time stamps are generated as hash chains where, given a later value t_2 the previous value t_1 can be obtained by

$f(t_2, t_1)$ repeated applications of a hash function (see [16] for an example of where hash chains are used for payment). This mechanism makes it impossible (in reasonable time) for a node to generate future nonces (time stamps) though it is very efficient to compute f.

We note that such an external global nonce source already exists in most areas today – the cellular network. Devices such as those provided by [1] use the existing CDMA cellular network to provide highly accurate microsecond time resolution. Thus, if the mobile devices carried by users have this technology incorporated, then the assumption we make above is reasonable. The only additional requirement we have is that rather than transmitting time, the NS should transmit nonces as described.

In order to motivate the impossibility result, it is instructive to begin with a protocol that does not work. Using the example in Figure 1(a), say that Alice has a packet P that needs to be forwarded to Bob. Alice computes $h(P)$,

Alice $\longrightarrow$ Bob	$\mathsf{Sign}(sk_A, T_{AB} \,\|\, h(P)) \,\|\, P$
Bob $\longrightarrow$ Charlie	$\mathsf{Sign}(sk_B, T_{BC} \,\|\, h(P)) \,\|\, P$
Bob $\longrightarrow$ CA	$\mathsf{Sign}(sk_A, T_{AB} \,\|\, h(P))$ $\|\, \mathsf{Sign}(sk_B, T_{BC} \,\|\, h(P))$
CA computes $\bar{\delta}_B = f(T_{BC}, T_{AB})$	
$\bar{\delta}$ denotes the delay computed by CA; $\|$ denotes string concatenation	

TABLE I
INCORRECT PROTOCOL.

where h is a random element from a collision-resistant hash function family $\mathcal{H}$, and signs the concatenation of T_{AB} and $h(P)$. T_{AB} is a time stamp from the NS that Alice *claims* is the time when she forwarded the packet to Bob. Alice forwards this string along with the packet P to Bob. After Bob gets the packet from Alice, he performs the same operation to forward it to Charlie. Bob generates a receipt $\mathsf{Sign}(sk_A, T_{AB} \,\|\, h(P)) \,\|\, \mathsf{Sign}(sk_B, T_{BC} \,\|\, h(P))$ which he sends to the CA some time in the future. After CA obtains such receipts from all nodes along the path, it determines the delay experienced at each hop, as shown in the table. Note that each receipt contains duplicate information – Bob's receipt as well as Alice's receipt contain $\mathsf{Sign}(sk_A, T_{AB} \,\|\, h(P))$. The reason to do this is to ensure that nodes cannot lie about the time stamps. For instance, we want to prevent cases where Alice sends T_{AB} to Bob and T'_{AB} to CA.

To see that this protocol fails to correctly monitor QoS, it is sufficient to consider two cheating scenarios that are indistinguishable from each other. In the first scenario, the source and destination are honest but all the other nodes along the path collude. Thus, the first hop from the sender creates a receipt with the correct (i.e., latest) time stamp prior to forwarding it. However, after that all the nodes on the path arbitrarily delay the packet but use old time stamps to show that they did forward the packet within their deadlines. Thus, the destination gets the packet late and reports a correct but late time stamp to CA. In the second scenario, the source and destination collude and the other nodes are honest. Here, the destination simply claims to have received the packet really late even if it did arrive on time (all it has to do is wait to create the receipt until a long time has passed). The CA cannot distinguish between these two cases and thus the protocol is incorrect.

One fix to the protocol to prevent the destination from cheating is to have every node i encrypt the packet using a symmetric key K_i prior to forwarding it. When the destination D receives the packet, it then needs to create a time stamp T_D and send $\mathsf{Sign}(sk_D, T_D \,\|\, h(P))$ back upstream all the way to the source. On receiving $\mathsf{Sign}(sk_D, T_D \,\|\, h(P))$, each node along the path forwards its key K_i to the destination thus allowing the destination to read the packet. All the nodes include $\mathsf{Sign}(sk_D, T_D \,\|\, h(P))$ in their receipts sent to CA. The drawback of this scheme is the increased delay from when the packet is sent to when it can be read at the destination. Furthermore, despite this overhead, there is still the possibility of incorrect operation. Thus, after intermediate nodes receive $\mathsf{Sign}(sk_D, T_D \,\|\, h(P))$, they can arbitrarily delay sending K_i to the destination D which causes the actual QoS to be violated (since the application relying on the packets cannot decrypt it in time) but all the nodes get payment.

The discussion above indicates the difficulty in verifying the actual per-hop delay experienced by a packet as well as the end-to-end delay. Indeed, we have the following result.

Theorem V.1. (Impossibility) *Assume a 3-node path where Alice is the source and Charlie is the destination with Bob the forwarding node. If we assume that nodes are untrustworthy and the time stamps they report can be arbitrarily unbounded*[1], *then there does not exist any cheat-proof protocol.*

Proof: To prove the theorem, it suffices to generate two cases of cheating that are indistinguishable from one another at the CA, for *any protocol*. Consider Figure 2 where we identify events that are common to all protocols. t_1 is the actual time when Bob can read P following a transmission from Alice. t_2 is the time when Charlie could first read P following a transmission from Bob. Note that there may be one or more transmissions between Alice and Bob (and between Bob and Charlie, etc.). However t_1 is the earliest when Bob can read the packet (e.g., Alice may send P encrypted at first and later send the key – thus t_1 would represent the time the key is received by Bob). Now, we note that these two times t_1 and t_2 indicate events that have to occur in all protocols.

Let t'_1 denote the time when Alice claims Bob should be able to read the packet and t'_2 is the time when Bob claims that Charlie can read P. Charlie claims that he can read P at t'_3 only. Since CA is only sent t'_1, t'_2, t'_3, etc., it has to determine per-hop delays based only on these values. Let $\delta'_B = f(t_2, t_1)$ be the actual delay experienced at Bob. We have two cases:

Case 1: δ'_B is much greater than the maximum acceptable delay δ_B. Bob claims a delay of $f(t'_2, t'_1) < \delta_B$ while

[1]Nodes can cache and use arbitrarily old time stamps. However, nodes cannot use future time stamps because they cannot be predicted (due to the way in which NS creates the time stamps).

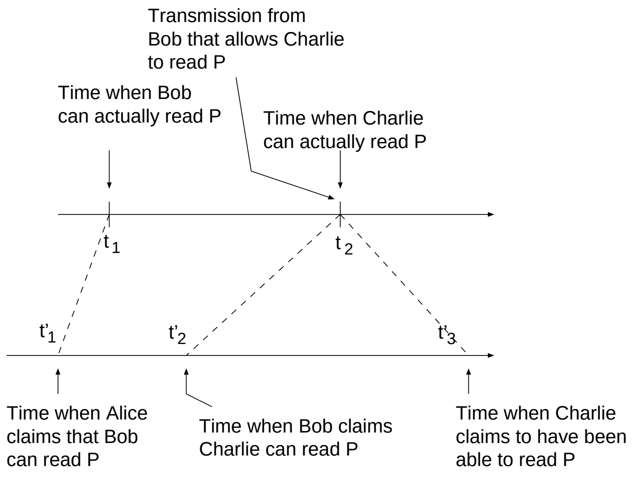

Transmission from Bob that allows Charlie to read P
Time when Bob can actually read P
Time when Charlie can actually read P
t1
t2
t'1
t'2
t'3
Time when Alice claims that Bob can read P
Time when Bob claims Charlie can read P
Time when Charlie claims to have been able to read P

Fig. 2. Impossibility proof.

according to Charlie's time stamp t'_3, Bob's delay is in fact $f(t'_3, t'_1) > \delta_B$.

Case 2: δ'_B is less than the negotiated delay δ_B. As above Bob claims a delay of $f(t'_2, t'_1)$ and according to Charlie's time stamp the delay at Bob is $f(t'_3, t'_1)$.

In both cases Bob claims a lower delay than δ_B and according to Charlie's time stamp Bob's delay is greater than δ_B. However, in one case Charlie gets to read P within Bob's deadline while in the other case it does not. The CA cannot distinguish between these two cases and thus any protocol that only uses these unbounded time stamps will be incorrect. □

The primary insight the above proof provides is that the protocols do not work because $t'_2 \neq t'_3$ (assuming zero propagation and transmission delay) indeed $f(t'_3, t'_2)$ may be unbounded and the CA cannot trust any one node's version of events.

VI. Solution 1: Using Angels to Verify per Packet QoS

The discussion in the previous section makes it clear that it is not possible to deliver verifiable QoS in the absence of some trustworthy component at each node which can at least bound the extent of lying about time stamps that nodes can indulge in. Following this insight, in this section we introduce the concept of an *angel*, and use this concept to develop a protocol for verifying per-hop QoS per-packet (Figure 1(a)).

A. Overview of Angels

An *angel* is a computing component that is always honest, in the sense that it will always faithfully execute the operations asked of it; it is not rational. Such an object might be instantiated by a piece of protected code, or trusted hardware. *We require that all parties to the protocol have an angel*. Each party runs its angel voluntarily during the protocol, that is angels do not provide oversight or enforce proper behavior per se. It is perfectly acceptable from the point of view of the protocol that a party never uses its angel (although the structure of the protocol will be such that not invoking the angel will ensure non-payment). *Our angels will be assumed to have been initialized with certain secrets (e.g., a secret key from a public-key/secret-key pair), and be able to provide certain cryptographic functionality to its host, such as producing signatures and decrypting ciphertexts, and* **that is all**. No further assumptions about the angels are required for our protocols.

Let us stress that an angel is not the same as a trusted third party as each angel is local to its host. Moreover, each angel is unaware of the existence of other angels and so does not coordinate with them; thus the angels as a group are not the same as a distributed trusted third party. This means that the angels do not coordinate packet transmissions among the nodes nor can they force specific node behaviors. Notice, too, that trusting an angel is quite different from trusting another node, as the angel is being trusted to carry out cryptographic operations on behalf of its host. We certainly would not trust another node in this way.

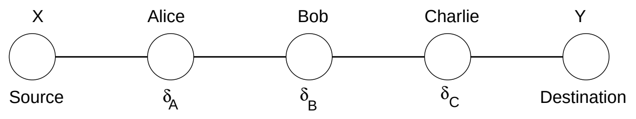
B. Receipt-generation protocol

Let δ denote the negotiated delay at each hop (which typically depends on congestion on that hop). The protocol we present here attempts to *estimate* the actual delay experienced (denoted as $\bar{\delta}$) by using time stamps signed by angels at each node. Table II presents the protocol for the case when Bob sends packet P to Charlie. For the sake of clarity, we assume that X is the source and Y the destination of this flow. In step (1), Bob generates a symmetric key K_B and uses this to encrypt the packet P. The key is in turn encrypted (along with some other information) using the public key of Charlie's angel. The reason we do this is that when Charlie wants to read the packet P, it will need to ask its angel to provide K_B (steps (3) and (5)). When Charlie makes this request, it includes a time stamp T_{CB}. This time stamp T_{CB} then denotes the time stamp given by Charlie to its angel when it wants to read P. Note that Charlie can select any value for T_{CB} but, as our discussion will show, due to the structure of the protocol, it is in Charlie's best interest to be truthful about T_{CB}.

In step (1), Bob includes $h(P)$ in R. The reason to do this is simply to ensure that the receipt generated by Charlie's angel σ_C ties together time stamp T_{CB} and packet P. When CA gets receipts σ from all nodes, it uses $h(P)$ to ensure that all receipts correspond to the same packet before determining $\bar{\delta}$'s. The estimated delay $\bar{\delta}_i$ at each hop i is computed using the expression in step (8). Thus, the delay for Bob $\bar{\delta}_B$ is computed as the number of clock ticks between when Charlie reads the packet and when Alice read the packet. The reason for using this expression is to force the greedy but rational nodes to behave in a deterministic way. We return to a discussion of this expression later in section VI-C. Another interesting feature of this is that even if some receipts are not delivered to the CA, the CA can still compute a subset of $\bar{\delta}$ values since each $\bar{\delta}$ only depends on two timestamps.

The other quantities included in R at step (1) are the time stamps T_{AX}, T_{BA} and the negotiated delay δ_B. The reason for including this information in R is to prevent a specific type of cheating as illustrated in Figure 3.

Let us assume that the end-to-end delay requirement for the packet is $\tau = \delta_A + \delta_C$ (this equation is also explained in



Bob transmits packet P to Charlie	
(1) Bob	Run Key to produce K_B; $Q \leftarrow \mathsf{Enc}(K_B, P)$; $R \leftarrow \overline{\mathsf{Enc}}(pk_C, K_B \,\|\, T_{BA} \,\|\, T_{AX} \,\|\, \delta_B \,\|\, h(P))$;
(2) Bob $\longrightarrow$ Charlie	$Q \,\|\, R$
(3) Charlie $\longrightarrow$ Angel	$R \,\|\, T_{CB}$

(4) Angel	$K_B \,\|\, T_{BA} \,\|\, T_{AX} \,\|\, \delta_B \,\|\, h(P) \leftarrow \overline{\mathsf{Dec}}(sk_C, R)$
	if $f(T_{CB}, T_{AX}) > \delta_B$ then *invalid*
	else $\sigma_C \leftarrow \mathsf{Sign}(sk_C, T_{CB} \,\|\, h(P))$

(5) Angel $\longrightarrow$ Charlie	$\sigma_C \,\|\, K_B$
(6) Charlie	$P = \mathsf{Dec}(K_B, Q)$
(7) Charlie sends σ_C to CA	
(8) CA computes $\bar{\delta}_B = f(T_{CB}, T_{AX})$	

TABLE II

PROTOCOL FOR PACKET TRANSMISSIONS. THE HASH FUNCTION h IS A RANDOM ELEMENT FROM THE FAMILY $\mathcal{H}$. T_{AX} IS THE TIME STAMP FOR WHEN ALICE READ THE PACKET SENT BY ITS PREVIOUS HOP X. T_{BA} IS THE TIME STAMP GIVEN BY BOB TO ITS ANGEL WHEN IT READ P. T_{CB} IS THE TIME STAMP GIVEN BY CHARLIE TO ITS ANGEL WHEN IT WANTS TO READ P. $f(T_2, T_1)$ COMPUTES THE NUMBER OF CLOCK TICKS THAT HAVE ELAPSED BETWEEN T_1 AND T_2.

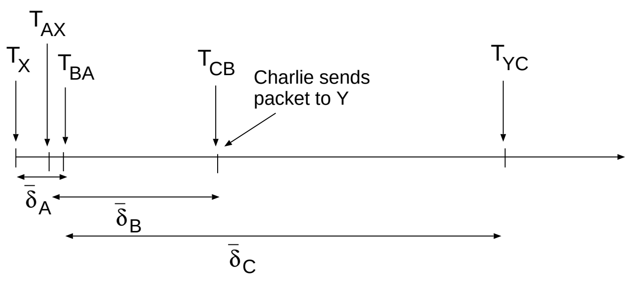


Fig. 3. Colluding nodes X, Y, Alice and Bob can cheat Charlie out of payment while meeting the end-to-end delay requirements.

section VI-C). The estimated end-to-end delay of the packet is then $\bar{\tau} = \bar{\delta}_A + \bar{\delta}_C$. Assume that Alice and Bob both forward the packet very quickly so that $\bar{\delta}_A << \delta_A$. Then, $\tau - \bar{\tau} = (\delta_A - \bar{\delta}_A) + (\delta_C - \bar{\delta}_C)$. Since $t_A = (\delta_A - \bar{\delta}_A) > 0$, Y can delay reading the packet for long enough to make $(\delta_C - \bar{\delta}_C) = -t_A$. Thus, the packet still meets the end-to-end bound but Charlie is not paid since according to Y's time stamp T_{YC}, Charlie violated its negotiated delay.

In order to guard against this form of cheating (numerous similar scenarios can be easily constructed), we introduce the check (if-then-else) in step (4). In Table II, Charlie's angel computes $\bar{\delta}_B$ using T_{AX} and T_{CB}. If this value is greater than δ_B, the key K_B is not provided to Charlie. In the example of Figure 3, the angel at Y will note that $\bar{\delta}_C > \delta_C$ and will not release key K_C to Y. Thus Y cannot read the packet which is incentive enough not to cheat in this way.

C. Computing Estimated Delays at CA

In Figure 4 we indicate the times when each node claims to have read the packet as $T_X, T_{AX}, T_{BA}, T_{CB}, T_{YC}$ (T_{ij} denotes the time when i claims to have read the packet from its upstream neighbor j). T_X is the time when source X claims to have read or generated the packet. Now, depending on the way in which CA computes $\bar{\delta}$ (i.e., assume expressions other than that given in step (8) of Table II), nodes can play different games to maximize their own revenue at the expense of other nodes. Let us consider two different such expressions. Say the CA uses $f(T_{AX}, T_X)$ as the value for $\bar{\delta}_A$ then Alice will decrypt the packet as soon as she receives it but can wait for a long time before transmitting it on to Bob. She may do this because she is forwarding packets for several connections and this behavior may maximize her revenue. This will cause $\bar{\delta}_A$ to be very small since Alice is pushing her delay onto Bob (whose delay is computed as $f(T_{BA}, T_{AX})$). So in this case, the packet is delayed by Alice but the delay is ascribed to Bob. Consider a different expression for $\bar{\delta}_A$ where we use $f(T_{BA}, T_{AX})$. In this case, after Bob receives the packet, he may wait for a long time to decrypt it. Thus, the delay experienced at Bob is ascribed to Alice.

Consider now the expression we use (step (8)). Here, $\bar{\delta}_B = f(T_{CB}, T_{AX})$. Thus, after Bob gets the packet from Alice he reads it and sends it on to Charlie and the delay ascribed to Bob is equal to the delay between when Alice read the packet and when Charlie reads the packet. Thus, it is in Bob's best interest to both read the packet as well as send it to Charlie as soon as he can in order to minimize $\bar{\delta}_B$. If he reads the packet and then waits for a long time to send it, this will increase T_{CB}. Likewise, if he waits for a long time to read the packet, this increases T_{BA} which in turn also increases T_{CB}.

One drawback with this method of assigning delays at each node is that we ascribe delay experienced at two hops to each node. In Figure 4 we indicate the delay between each time the packet is read as $x_1, \cdots, x_4$. Thus, according to our expression in step (8) (Table II) $\bar{\delta}_A = x_1 + x_2$, $\bar{\delta}_B = x_2 + x_3$, and so on. As we can see, the delay x_2 is being counted towards both Alice's as well as Bob's delay. We have to do this in order to force Alice to forward the packet as soon as she can. If we use either x_1 or x_2 individually for Alice's delay then the two forms of cheating discussed previously become possible.

There are two consequences of this form of ascribing delays. First, if the end-to-end delay requirement for our example path is τ then $\tau \geq \delta_A + \delta_C = (x_1 + x_2) + (x_3 + x_4)$. In general if we have an odd number of nodes between X and Y, then τ is the sum of the δ values for every other node along the path. A second consequence is the manner in which nodes now negotiate their delays δ. Say the actual (honest) delay that a packet will experience at Alice and Charlie is 1 whereas it is 100 at Bob. If Alice negotiates $\delta_A = 1$ then her estimated delay is always going to be greater than δ_A because x_2 is a large value ≤ 100. Therefore, Alice needs to negotiate a delay that accounts for the possible delay that is experienced at Bob as well. For instance, it would make sense for Alice to set $\delta_A = 101$ to account for forwarding the packet to Bob and for Bob to read the packet prior to forwarding it to Charlie (Alice has to assume the worst case that Bob will read the packet just before forwarding it and thus the packet will incur a delay of 100). Likewise, Bob sets $\delta_B = 101$ to account for forwarding the packet to Charlie and for Charlie to read it. Charlie sets $\delta_C = 1$ to account for its own delay. Thus, $\tau \geq \delta_A + \delta_C = 101 + 1 = 102$ which is the correct value for end-to-end delay given that the individual delays at Alice, Bob, and Charlie are 1, 100, and 1, respectively.

The algorithm used by each node to compute its negotiated delay δ is best illustrated using Figure 5. In the figure, d_A denotes Alice's estimate of the time between when the packet will be *transmitted by X and the time when Alice can then begin transmitting the packet*. This delay thus includes Alice's estimate of the the queueing delay at Alice. Note also that Alice reads the packet prior to transmitting it and therefore T_{AX} is always less than d_A. The delay d_B is Bob's estimate of the delay between when Alice commences transmission and when Bob expects to start transmitting, and so on. Thus, Alice sets $\delta_A = d_A + d_B$, Bob sets $\delta_B = d_B + d_C$ and Charlie sets $\delta_C = d_C + d_Y$. Delay d_Y is an estimate by Charlie of when the destination will read the packet. This can either be equal to the communication delay or some other quantity provided by X and Y during the time the contract is negotiated.

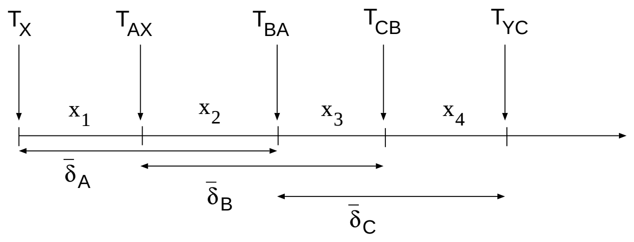


Fig. 4. Computing $\bar{\delta}_i$.

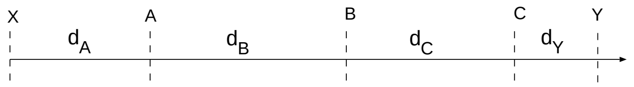


Fig. 5. Computing δ_i.

D. Properties of the Protocol

The assumption we make about nodes is that they are greedy but rational which means that they only seek to maximize their own revenue. Thus, they will attempt to minimize $\bar{\delta}$ while forwarding the packet as lazily as possible. Forwarding packets lazily means that the node can participate in many different connections to maximize its overall revenue. Under this model of node behavior, we can prove certain properties about the protocol described previously.

Figure 6 shows the timing relationship between different events. T_{ij} denotes the time stamp reported by node i when it reads the packet from node j. These are the only time stamps that are reported to CA and thus the estimated delays $\bar{\delta}$ are based on these values alone. t_{ij} denotes the *actual* time when i sends the packet to j (these times are not known to CA). For the purpose of the discussion here, we assume zero delay between when a packet is sent and when it is received. Finally, d'_i denotes the *actual* delay experienced by the packet at node i. Recall that d_i (in Figure 5) denotes the delay that node i used when negotiating δ_i.

Let us consider the transmission from Alice to Bob for different possible timings. Table III lists all the possible outcomes for different selections of T and t. There are four possibilities for Alice based on the relationship between d'_A, d_A and between T_{AX}, t_{AB}. If $d'_A \leq d_A$ this means that Alice forwards P within the time delay it used when negotiating δ_A. On the other hand, there is nothing stopping Alice from forwarding the packet late, i.e., $d'_A > d_A$. For each of these two cases, Alice may report a time stamp that is earlier than the actual time it sent the packet (i.e., $T_{AX} < t_{AB}$) or it will report the correct time. Alice *cannot* report a later time (i.e., $T_{AX} > t_{AB}$) since Alice cannot create future time stamps. Similar to Alice, Bob also has four possible behaviors giving us a total of sixteen combinations. If $d'_A \leq d_A$ and $d'_B \leq d_B$ then Bob always forwards the packet on to Charlie because $\bar{\delta}_A = f(T_{BA}, T_X) \leq d'_A + d'_B \leq d_A + d_B = \delta_A$. Of the remaining twelve cases, there are two instances in which Bob will drop the packet. This happens when both $d'_A > d_A$ and $d'_B > d_B$ and when $T_{BA} = t_{BC}$. This happens as follows: Bob's angel computes $\bar{\delta}_A = f(T_{BA}, T_X) = d'_A + d'_B > d_A + d_B = \delta_A$. Here, $f(T_{BA}, T_X)$ is exactly equal to $d'_A + d'_B$

Alice	Bob			
	$d'_B \leq d_B$		$d'_B > d_B$	
	$T_{BA} < t_{BC}$	$T_{BA} = t_{BC}$	$T_{BA} < t_{BC}$	$T_{BA} = t_{BC}$
$d'_A \leq d_A$				
$T_{AX} < t_{AB}$	$\bar{\delta}_A \leq \delta_A$ Bob forwards P	$\bar{\delta}_A \leq \delta_A$ Bob forwards P	?	?,+
$T_{AX} = t_{AB}$	$\bar{\delta}_A \leq \delta_A$ Bob forwards P	$\bar{\delta}_A \leq \delta_A$ Bob forwards P	?	?,+
$d'_A > d_A$				
$T_{AX} < t_{AB}$	?	?,*	?,♣	Bob drops P
$T_{AX} = t_{AB}$	?	?,*	?, ♣	Bob drops P
? $\triangleq$ if $(f(T_{BA}, T_{AX}) > \delta_A$ Bob drops P				

TABLE III
ALL POSSIBLE BEHAVIORS OF ALICE AND BOB

because this computation does not depend on T_{AX} at all and since Bob reports the correct time stamp, $T_{BA} = t_{BC}$. The remaining ten cases are classified as ? in the table because the fate of the packet depends on T_{BA}. Indeed, the packet is only forwarded if $f(T_{BA}, T_X) \leq \delta_A$. Of these ten cases there are two cases (indicated by *) where Bob truthfully reports T_{BA}. In these two cases, the packet is only forwarded if Bob compensates for the delay added on by Alice by forwarding the packet at least $d'_A - d_A$ early. In the two cases where $d'_A > d_A$ and $d'_B > d_B$ and where Bob reports $T_{BA} < t_{BC}$ (indicated by ♣), the packet will be forwarded if the criteria in ? is satisfied but there is a good chance that Charlie will drop the packet unless it can compensate for the delays added by Alice and Bob (equal to $(d'_A - d_A) + (d'_B - d_B)$). In the two cases indicated by a +, Bob delays sending the packet $d'_B > d_B$. If it does send the packet (because $d'_B - d_B \leq d_A - d'_A$), there is a possibility that the packet will be dropped by Charlie unless Charlie is willing to compensate for the additional delays of $f(t_{BC}, T_{AX}) - d_B = f(T_{AX}, t_{AB}) + d'_B - d_B$ that are because of delay added by Bob and delay added due to Alice misreporting T_{AX}. The remaining four cases are similar to the ones we have described above and are omitted.

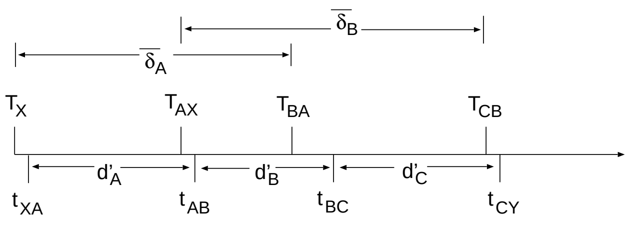


Fig. 6. Relative location of different times.

1) Summary: If we were to summarize the description above simply we can say that:

- If Alice forwards the packet to Bob early and if Bob exploits this to forward the packet late, there is a risk that Charlie will drop the packet. This is because Charlie only measures time from when Alice reportedly read the packet to when Charlie reads the packet. The extra delay that Bob adds on is therefore included in $\bar{\delta}_B$ by Charlie but the fact that Alice forwarded the packet early is not included in this computation.
- If Alice forwards the packet late, then there is a risk that Bob will drop the packet unless Bob can sacrifice some of its delay and forward the packet early.

E. Cheating Behaviors

Definition VI.1. Honesty *is defined as the case when $T = t$ and $d' \leq d$. In other words, (1) the node reads the packet and immediately forwards it while reporting the correct time stamp and (2) it meets its negotiated delay bound.*

Definition VI.2. Cheating *behavior is defined as the complement of honesty and includes three cases: $\{T = t \wedge d' > d\} \vee \{T < t \wedge d' \leq d\} \vee \{T < t \wedge d' > d\}$.*

Since we are dealing with wireless networks, there is always the chance that the packet will be lost either due to error, or because it did not meet the delay bound somewhere along the path. Given this presence of uncertainity, we can also define:

Definition VI.3. Rational *behavior is defined as one that* does not decrease *the probability that the packet reaches the destination in time.*

The motivation for nodes to behave rationally derives from their desire to get paid. This is because *we assume that unless the packet meets its end-to-end delay bound τ, no node gets paid*. Given these definitions, we can now state specific results relating to cheating nodes. For clarity, we continue to use Alice, Bob, and Charlie as the intermediate nodes. However, the results hold true for longer paths as well.

Lemma VI.1. (Single Cheater) *Cheating by a single node is not rational behavior.*

Proof: Consider the case when Alice cheats alone. Referring to Figure 6 we can write the following expressions:

$$\bar{\delta}_A = f(t_{XA}, T_X) + f(T_{AX}, t_{XA}) \\ +f(t_{AB}, T_{AX}) + f(T_{BA}, t_{AB})$$

and,

$$\bar{\delta}_B = f(t_{AB}, T_{AX}) + f(T_{BA}, t_{AB}) + f(t_{BC}, T_{BA}) + f(T_{CB}, t_{BC})$$

Referring to Table III, we can identify the following cases for Alice (assume that Bob is honest):

- $d'_A \leq d_A \wedge T_{AX} < t_{AB}$: If $d'_B \leq d_B$ Bob will forward the packet but there is an increased chance that Charlie will drop it. This is because this form of cheating by Alice increases the first term (i.e., $f(t_{AB}, T_{AX})$) in the expression for $\bar{\delta}_B$ above.
- $d'_A > d_A \wedge T_{AX} = t_{AB}$: In this case the second term $f(T_{AX}, t_{XA})$ in the expression for $\bar{\delta}_A$ computed by Bob will increase thus increasing the chance that Bob will drop the packet.
- $d'_A > d_A \wedge T_{AX} < t_{AB}$: Here, both the values for $\bar{\delta}_A$ and $\bar{\delta}_B$ will increase (as discussed for the first two cases) resulting in an increased chance of either Bob or Charlie dropping the packet.

Since all the cheating behaviors result in a greater chance of the packet being dropped, none is rational. □

Lemma VI.2. (Sequential Cheaters) *Cheating when a sequence of nodes colludes along the path is not rational.*

Proof: Say $A_1, \cdots, A_k$ are a sequence of nodes that collude together. Say Bob is the next hop from A_k and is the first honest node. Depending on how A_k cheats, either Bob or Charlie or both will have an increased chance of dropping the packet (using the argument from Lemma VI.1). This is thus not rational behavior. □

Lemma VI.3. *It is not rational for the source X to set $T_X < t_{XA}$ (as illustrated in Figure 6).*

Proof: The expression for $\bar{\delta}_A$ in Lemma VI.1 has a term $f(t_{XA}, T_X)$. By reducing this term to zero, X increases the chance that Bob will forward the packet from Alice. Hence it is not rational to have $T_X < t_{XA}$. □

Lemma VI.4. *The source and destination cannot cheat in a way to deny payment to any node.*

Proof: Cheating by the source and destination translates to the packet meeting its end-to-end delay bound τ but not paying one or more intermediate nodes. This is impossible because the angel at a node only provides the key if the previous node's estimated delay meets the delay bound, $\bar{\delta} < \delta$. Thus, the packet would not have reached the destination if any of the intermediate node's did not meet the delay requirement. □

We note that the destination can delay generating T_{YC} such that $\bar{\delta}_C = \delta_C$.

Lemma VI.5. *No arbitrary collection of nodes along the path can cheat.*

Proof: We can partition the set of nodes into subsets where each subset represents a sequence of nodes along the path. Then by Lemma VI.2 cheating is irrational for each such subset. Hence it is irrational for the collection of cheaters as well. □

Theorem VI.1. *The protocol is cheat-proof.*

Proof: The proof follows from Lemmas VI.1 to VI.5. □

F. Verifiable Resolution and Δ

Given that the nonce source (NS) transmits a time stamp every Δ seconds, it is interesting to examine the achievable time resolution of our protocol. In other words, what is the minimum δ value that can be verified? In general, it is easy to see that δ should be of the form $k\Delta, k = 1, 2, \ldots$. Is this resolution sufficient for present day wireless networks? If we consider a typical 802.11b data rate of 5.5Mbps, the time to transmit one 1500 byte packet is approximately 2 ms. For a higher rate radio like 802.11a this falls to 0.2 ms. Assuming a queueing delay at each node of the order of 1 ms as well, a clock resolution of $\Delta = 1ms$ works reasonably well. This has two implications: first, NS needs to send time stamps every millisecond and second, for a n hop path, the minimum end-to-end verifiable delay is $(n + 1)$ ms.

Unfortunately, if the NS has a coarser granularity, for example $\Delta \sim 100ms$, then we have a problem – the verifiable end-to-end delay will be of the order of hundreds of milliseconds which is useless for many real-time applications. The only solution we have is to negotiate $\delta = 0$ for all the nodes on the path. Then, if the sender synchronizes transmission with a clock tick, the destination needs to receive the packet before the next clock tick. This ensures that the end-to-end delay is at most Δ.

We note that the problem discussed here is somewhat artificial because in this section we only look at verifying the delay *per packet* transmission. In the next section we look at average delays for a flow and that gives us additional flexibility in improving the verifiable resolution to less than Δ.

VII. Solution 2: Verifying Average Delay Per Hop

Consider the model where the delay negotiated is per-flow rather than per-packet, Figure 1(b). While it is still possible to use the protocol described in Table II to verify average delay (the CA simply uses delay per packet to compute the average at the end of the connection), it is overkill because each node will have to send one receipt per packet forwarded. In this section we develop a variation that is more efficient in that each intermediate node only sends one receipt to the CA at the end of the connection and the receipts contain the average and maximum delay for that hop.

The protocol illustrated in Table IV works in much the same way as the protocol in Table II with some changes. Using the transmission from Bob to Charlie for illustration, the key idea in this protocol is for Charlie's angel to keep a running tally of the average delay per packet as well as the maximum delay seen. For each packet P^i:

- Charlie's angel keeps a running tally of the average delay for Bob. avg^{i-1} denotes the average delay of packets $P^1, \ldots, P^{i-1}$. If the average delay avg^i with packet P^i is greater than δ_B, the packet P^i is rejected (i.e., the angel returns *invalid*). The angel also keeps track of the maximum delay max^i seen thus far.
- Charlie's angel does not maintain any state. Rather, in addition to providing the key to decrypt packet P^i, it gives Charlie a receipt σ^i that contains: i which is the sequence number last seen, avg^i and max^i that denote

Bob transmits packet P^i to Charlie	
(1) Bob	Run Key to produce K_B^i; $Q^i \leftarrow \mathsf{Enc}(K_B^i, P^i)$; $R^i \leftarrow \overline{\mathsf{Enc}}(pk_C, K_B^i \,\|\, i \,\|\, T_{AX}^i \,\|\, T_{BA}^i \,\|\, \delta_B)$; $H^i = h(P^i \,\|\, H^{i-1})$
(2) Bob $\longrightarrow$ Charlie	$Q^i \,\|\, R^i \,\|\, H^i$
(3) Charlie $\longrightarrow$ Angel	$H^i \,\|\, R^i \,\|\, T_{CB}^i \,\|\, \sigma_C^{i-1}$
(4) Angel	Extract avg^{i-1}, max^{i-1}, $i-1$ from σ_C^{i-1}; avg$^i = ((i-1)$ avg$^{i-1} + f(T_{CB}^i, T_{AX}^i))/i$; max$^i = $ max(max$^{i-1}, f(T_{CB}^i, T_{AX}^i))$; if avg$^i > \delta_B$ *invalid* else $\sigma_C^i \leftarrow \mathsf{Sign}(sk_C, i \,\|\, H^i \,\|$ avg$^i \,\|$ max$^i)$
(5) Angel $\longrightarrow$ Charlie	$\sigma_C^i \,\|\, K_B^i$
(6) Charlie	$P^i = \mathsf{Dec}(Q^i, K_B^i)$
(7) At the end of the connection (n packets), Charlie sends σ_C^n to CA	

TABLE IV
PROTOCOL FOR MULTIPLE PACKET TRANSMISSIONS.

the average and maximum delays seen thus far, as well as a cumulative hash $H^i = h(P^i \,\|\, H^{i-1})$. Since σ^i is signed by Charlie's angel, Charlie cannot tamper with it. When Charlie presents σ^{i-1} along with R^i to its angel, the angel can check to see if the packet sequence numbers are in order. This ensures that Charlie cannot cheat by giving some old receipt $\sigma^j, j < i-1$ to the angel. Likewise, including H^i in σ^i ensures that the receipt does correspond to the sequence of packets upto and including P^i.

- We note that step (4) of the protocol can be changed to monitor traffic statistics differently. For instance, instead of maintaining a running average, the angel could have Charlie store $f(T_{CB}^i, T_{AX}^i)$ for *all* packets P^i. This is accomplished by including these values in σ^i that is kept by Charlie. Thus, instead of performing the check avg$^i > \delta_B$ in step (4), the angel could perform more sophisticated checks, for instance, "do 95% of all packets have a delay less than δ_B?". Furthermore, we note that by including all values of $f(T_{CB}^i, T_{AX}^i))$ in the receipt σ^i, the CA can compute arbitrary statistics (such as jitter for real-time applications) per-hop as well as end-to-end for the flow.

We omit proofs of cheat-proofness here since they follow arguments similar to those presented in section VI-E.

A. Improving Verifiable Resolution Given Δ

Recall our discussion in section VI-F where we concluded that for single packets, the best end-to-end delay time resolution we could hope to obtain is Δ. Fortunately, however, if we have a connection where n packets are sent from X to Y, we can do much better. Let δ denote the per-hop delay resolution we are interested in and $\Delta > \delta$ be the clock resolution of the global nonce source. Now consider the situation where CA estimates $\bar{\delta}$ for some node. Figure 7 shows the cases when

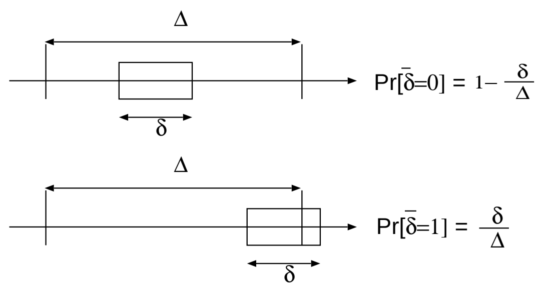


Fig. 7. Two cases showing CA computed delays.

$\bar{\delta} = 0$ and when $\bar{\delta} = 1$. The former case happens when the time stamps reported to CA are both the same (e.g., Alice and Charlie read the packet within the same clock tick, so $\bar{\delta}_B = 0$). The latter case happens when the time stamps differ by one clock tick. The probability that $\bar{\delta} = 0$ is given by $1 - \frac{\delta}{\Delta}$. To see this, consider the experiment where we have a line segment of length δ that is thrown onto a line segment of length Δ. Then, we look at the probability that the smaller line segment is wholly contained within the larger one. This only happens if the left edge of the shorter line segment lies between $[0, \Delta - \delta]$, otherwise it will cross the right boundary of the longer line segment. Assuming random uniform selection for the left edge of the short line segment, we get a probability as indicated above. The probability that $\bar{\delta} = 1$ can then be seen to equal $\frac{\delta}{\Delta}$.

Given that several packets are sent from X to Y, we can see that the expected value $E[\bar{\delta}] = 0 \times (1 - \frac{\delta}{\Delta}) + 1 \times \frac{\delta}{\Delta} =$

$\frac{\delta}{\Delta}$. However, in reality, if we only send a limited number of packets, the average value computed may be different from $\frac{\delta}{\Delta}$. It is therefore important to bound the difference between the average value (sample mean) and $\frac{\delta}{\Delta}$ (population mean) since this shows the best achievable resolution.

Let $\bar{\delta}_1, \bar{\delta}_2, \cdots, \bar{\delta}_n \in \{0, 1\}$ be the sequence of estimated delays for the n packets. Let us assume that these delays are identically distributed independent random variables. Let S_n denote the sum of these n delays. Then,

$$S_n = \sum_{i=1}^{n} \bar{\delta}_i$$

and the variance is,

$$\sigma^2 = \frac{\delta}{\Delta} - \left(\frac{\delta}{\Delta}\right)^2$$

Using the Central Limit Theorem ([7]), we can write,

$$P\left[|S_n - \frac{\delta}{\Delta}n| < \alpha\sigma\sqrt{n}\right] \approx \mathcal{N}(\alpha) - \mathcal{N}(-\alpha) \qquad (1)$$

Here $\mathcal{N}(x)$ is the standard normal probability distribution function. In words, the probability that the sample mean S_n/n differs from the population mean δ/Δ by less than $\pm\alpha\sigma/\sqrt{n}$ is given by $\mathcal{N}(\alpha) - \mathcal{N}(-\alpha)$.

Let us say that we would like the sample mean S_n/n to differ from δ/Δ by no more than $\gamma(\delta/\Delta), \gamma < 1$ with a probability of $\mathcal{N}(\alpha) - \mathcal{N}(-\alpha)$. In other words,

$$P\left[\left|\frac{S_n - \frac{\delta}{\Delta}n}{n}\right| < \frac{\delta}{\Delta}\gamma\right] \geq \mathcal{N}(\alpha) - \mathcal{N}(-\alpha)$$

Substituting these values back in equation 1 we get,

$$n\frac{\delta}{\Delta}\gamma \geq \alpha\sigma\sqrt{n}$$

This simplifies to,

$$n \geq \frac{\alpha^2}{\gamma^2}\left(\frac{\Delta}{\delta} - 1\right)$$

Let us say that $\mathcal{N}(\alpha) - \mathcal{N}(-\alpha) = 0.95$ (i.e., the probability that S_n/n differs from δ/Δ by less than $\pm\gamma(\delta/\Delta)$ is 95%), we get $\alpha = 1.96$. Substituting above, we get,

$$n \geq \frac{3.84}{\gamma^2}\left(\frac{\Delta}{\delta} - 1\right)$$

If we want to improve the clock resolution by a factor of 10 (i.e., $\delta/\Delta = 0.1$) such that the sample mean is within 10% of δ/Δ (i.e., $\gamma = 0.1$ which means that with a probability of 95% the sample mean will be in the range $[0.09, 0.11]$) then we need at least 3456 packets in the flow. On the other hand, if we only want to improve clock resolution by 50% (i.e., $\delta/\Delta = 0.5$) with $\gamma = 0.1$, then $n \geq 384$ packets. As another example, if we allow $\gamma = 0.25$, then the number of packets we need for $\delta/\Delta = 0.1$ is 552 (down from 3456) and if $\delta/\Delta = 0.5$ the number of packets we need is 61.

In summary, we see that if we consider a flow of packets between a source and destination, it is possible to improve the resolution of our verification protocol. However, to get to finer and finer resolutions (i.e., smaller δ/Δ) with a greater accuracy (smaller γ) we need a larger number of packets.

VIII. Solution 3: No Global Nonce Source, No Angels

Let us now consider the case when we have no NS and no angel. Consider a simple two-hop path as illustrated in Figure 8 from a source *s* to the destination *d* via node *a*. For every packet i, *s* reports the time that it transmitted the packet t_i^s. This time is based on *s*'s local clock only. We assume that r_i^s is the time that the packet was delivered by the application to the node for onward transmission. Thus $t_i^s - r_i^s$ is the buffering delay encountered at the source node *s*. At the destination, packet i is reported received at time r_i^d and is delivered to the application at time t_i^d. At node *a*, we similarly have times r_i^a and t_i^a denoting the time the packet was received from *s* and the time it was forwarded to *d*. Thus, $t_i^a - r_i^a$ is the delay experienced by packet i at node *a*.

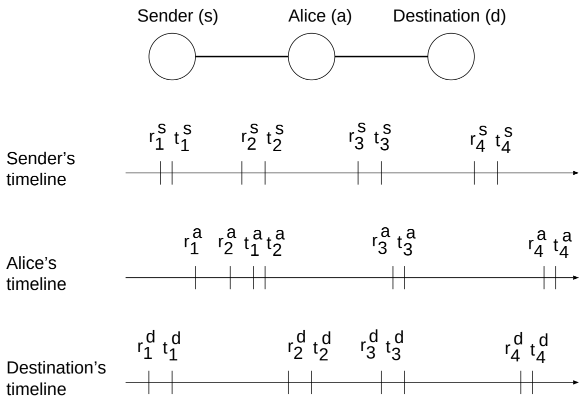

Sender (s)
Alice (a)
Destination (d)
r_1^s t_1^s r_2^s t_2^s r_3^s t_3^s r_4^s t_4^s
Sender's timeline
r_1^a r_2^a t_1^a t_2^a r_3^a t_3^a r_4^a t_4^a
Alice's timeline
r_1^d t_1^d r_2^d t_2^d r_3^d t_3^d r_4^d t_4^d
Destination's timeline

Fig. 8. Example of reported times.

Since nodes use only their own clocks to report the times r_i, t_i, there exist numerous ways of cheating (as noted in sections III,V). To force them to behave honestly, the protocol presented here is intimately linked to a specific form of payment where nodes delaying packets are hit with a large *penalty* whereas nodes who deliver the negotiated QoS get a fixed payment. We will show that this form of payment, if carefully crafted, will make it irrational for nodes to cheat since they end up owing money.

Let the path be $s \rightarrow a_1 \rightarrow a_2 \rightarrow \cdots \rightarrow a_k \rightarrow d$. Let $r_i^{a_j}$ and $t_i^{a_j}$ be the times that node a_j reports that it received and transmitted packet i. Table V describes the forwarding protocol used. Node a_j computes the hash of the concatenation of its receive and transmit times for the packet, along with the packet itself and appends this to the header. The header and the packet P^i are then forwarded to the next node a_{j+1}. Node a_j separately sends its receive and transmit times for all packets to the CA. The reason for including the hash in the header is to ensure that nodes do not change the reported times at a later date. Further, since receive and transmit times are not readable from the header, nodes further downstream cannot exploit this knowledge for their own benefit.

Once the CA receives the receive and transmit times for all packets P^i from all nodes a_j it runs the following algorithm to ascribe per-hop per-packet delays.

a_j transmits a packet P^i to a_{j+1}	
(1) a_j	$< hdr > \leftarrow < hdr > \|\| h(r_i^{a_j} \|\| t_i^{a_j} \|\| P^i)$
(2) $a_j \rightarrow a_{j+1}$	$< hdr > \|\| P^i$
(3) a_j sends $(r_i^{a_j}, t_i^{a_j})$ to CA	

TABLE V
PROTOCOL FOR SOLUTION 3.

1) CA first synchronizes times reported by all nodes with respect to the transmission and reception of the first packet in the stream. Thus, the transmit time reported by node a_{j-1} for the packet is set to the receive time reported by node a_j, and so on. Node a_1's receive time for the first packet is set to the transmission time from s and similarly node d's receive time for the first packet is set to the transmit time of that packet by a_k. Let us call this process **aligning** the times of individual nodes. After we align the times of all nodes with respect to the first packet, the transmit and receive times for all subsequent packets are appropriately offset for each node. Indeed, we claim that if nodes are honest and if the clock rates do not change for the duration of the connection, then all of the aligned times are mutually comparable and can be thought of being derived from a common global clock[2]. Let us denote these new aligned receive and transmit times for packet i at node a_j times as $R_i^{a_j}$ **and** $T_i^{a_j}$.
2) *If* nodes are honest, then, based on aligning times w.r.t. the first packet, we can write,
$$\bar{\delta}_i^{a_j} = T_i^{a_j} - T_i^{a_{j-1}} = R_i^{a_{j+1}} - T_i^{a_{j-1}} = R_i^{a_{j+1}} - R_i^{a_j}$$
3) However, nodes may not be honest. Therefore, we set (excluding s, d) the delays for individual packets i as follows:
$$\bar{\delta}_i^{a_j} = \max\left(|T_i^{a_j} - T_i^{a_{j-1}}|, |R_i^{a_{j+1}} - T_i^{a_{j-1}}|, |R_i^{a_{j+1}} - R_i^{a_j}|\right) \tag{2}$$
4) For the source s and destination d, the delays are computed as,
$$\begin{aligned}\bar{\delta}_i^s &= |R_i^{a_1} - R_i^s| \\ \bar{\delta}_i^d &= |T_i^d - T_i^{a_k}|\end{aligned} \tag{3}$$
respectively.

In order to understand the motivation for equations 2, 3 consider the simple example in Figure 8 again but with specific values as follows. For s, the reported receive and transmit times (r_i^s, t_i^s) for three consecutive packets are $\{(0,0),(1,1),(2,2)\}$. For node a the reported times (r_i^a, t_i^a) are $\{(0,1),(1,2),(2,3)\}$ (in other words each packet suffers a delay of 1). Finally, say d's reported times are $\{(1,1),(2.5,2.5),(3,3)\}$. Let us now align the times using packet 1 as reference. In this example, after alignment, we get the exact same times for all events yielding $(R_i, T_i) = (r_i, t_i)$ for all three nodes. Using equations 2, 3 we obtain the following per-packet delays at the three nodes:
$$\bar{\delta}_1^s = \bar{\delta}_2^s = \bar{\delta}_3^s = 0$$
$$\bar{\delta}_1^d = 1 - 1 = 0, \bar{\delta}_2^d = 2.5 - 2 = 0.5, \bar{\delta}_3^d = 3 - 3 = 0$$
and,
$$\bar{\delta}_1^a = 1 = \bar{\delta}_3^a, \text{ while, } \bar{\delta}_2^a = \max\{(2-1),(2.5-1),(2.5-1)\} = 1.5$$
In this example we assumed that the negotiated delay at node a is 1 and at s, d is zero (i.e., as soon as the packet comes to s it should be sent and likewise as soon as d receives the packet it needs to send it to the application). Thus, both nodes a and d violate the delay requirement for packet 2.

Observe that in reality we can get the above situation in more than one way: node a may be the culprit causing packet 2 to be delayed for 1.5 time units, but reporting an earlier transmit time; or node d may be at fault where even though it received the packet at time 2 it only forwarded it at time 2.5, and to cover itself it reports receiving the packet at time 2.5. Our algorithm cannot distinguish between these two cases (and, in general, between several alternative possibilities) but it does identify a superset of nodes that violate the delay bound. This example shows the situation when nodes are being dishonest. If nodes are honest, however, only the node(s) that delayed the packet would be identified as the culprit. Indeed, we have the following lemma.

Lemma VIII.1. *If all the nodes are honest then,*
1) *If all nodes meet their delay bounds, the delays computed by the algorithm also meet the delay bounds.*
2) *If some set of nodes* **s** *nodes do not meet their delay bounds for a given packet, then the algorithm identifies exactly this set of nodes as having higher than negotiated delays.*

Proof: The first part of the lemma is easy to see because given honest nodes, aligning their times can be considered equivalent to having a global clock (since times reported are correct). Indeed, the three values in equation 2 will be equal and will be the exact delay at each node. This same observation explains to the second part of the lemma as well. Say nodes $b_1, b_2, \cdots, b_l$ delayed the packet then by equation 2 each of these l nodes will be assigned the correct (large) delay. Note that the only other nodes that may be affected by this are the node upstream of b_1 and the node downstream of b_l (no other nodes will be affected because in equation 2 we only use timing information from one node on either side of the node under consideration). We claim, however, that the delays ascribed to these two nodes are not affected by the added delays at nodes b_i. Consider the node upstream of b_1 (let us call this node b_0 and the node upstream of it b_{-1}). Then,
$$\bar{\delta}^{b_0} = \max(|T^{b_0} - T^{b_{-1}}|, |R^{b_1} - R^{b_{-1}}|, |R^{b_1} - R^{b_0}|)$$
In none of the three delays above do we see a term T^{b_1} which is what is affected by node b_1 delaying the packet for too long. Indeed, careful scrutiny of the above expression will show that the three delays will have the same value and the delay $\bar{\delta}^{b_0}$ will be unaffected by the fact that b_1 delayed the packet. Similar arguments can be used for the node downstream of

[2] In general clocks differ in offset as well as in rate of drift. However, given a short enough connection, we can ignore the rate of drift and work simply with the offset.

b_l. Consider now the set **s** of nodes that delay the packet beyond the negotiated delay. We can break **s** into subsets of nodes that are contiguous. Then the above discussion applies independently for each such subset and hence to the whole set **s**. □

Definition VIII.1. *Define a node to be* **dishonest** *with respect to packet p if (1) p has a delay at the node greater than the negotiated delay and (2) the receive/transmit times the node reports for one or more packets are different from the actual times as per its clock.*

The intuition here is that the dishonest node will attempt to lie about packet p's delay by adjusting the values for (r_i, t_i).

Lemma VIII.2. *If a node is dishonest with respect to one packet p then the algorithm ascribes a higher delay for this packet to this dishonest node in addition to one or more other nodes.*

Rather than giving a proof (which is fairly simple), we draw the reader's attention to the example described previously which shows an instance of this type of protocol behavior.

The results thus far have shown the ability of the protocol to detect single cheaters. However, as the following lemma states, the protocol **does not** detect multiple colluding cheaters.

Lemma VIII.3. *If a sequence of nodes $b_1, b_2, \cdots, b_l$ on the path collude are dishonest, then the algorithm will only assign higher than negotiated delays for packets that are delayed at each b_k in the path to the last node b_l.*

Proof: Figure 9 shows the manner in which delays get

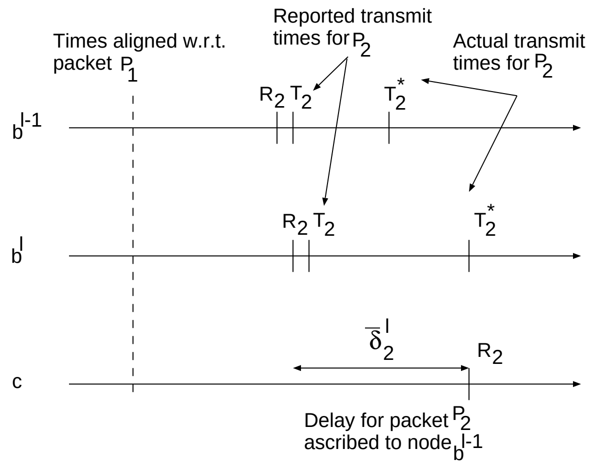

Times aligned w.r.t. packet P_1
Reported transmit times for P_2
Actual transmit times for P_2
b^{l-1}
R_2 T_2
T_2^*
b^l
R_2 T_2
T_2^*
c
$\bar{\delta}_2^l$
R_2
Delay for packet P_2 ascribed to node b^{l-1}

Fig. 9. Penalizing colluding nodes.

ascribed to node b^l for packet 2. In the figure, node c does not belong to the group of cheating nodes. Say node b^{l-1} delays packet 2 for a significant time and then transmits it at the time indicated by T_2^* but reports a transmission time of T_2. Likewise, node b^l also delays the packet but reports an earlier transmission time. According to equation 2, the delay ascribed to b^l for packet 2 will include the delay the packet experienced at node b^{l-1} as well as at b^l (the term $|R_2^c - T_2^{b_{l-1}}|$ will be the largest in the equation), as shown in the figure. Extending this to a series of cheaters $b^1, b^2, \cdots, b^l$, it is easy to see that node b^l will get blamed for a delay for packet 2 that is the sum of the additional delays suffered by the packet at each of the previous $l-1$ colluding nodes. □

For this form of cheating to be profitable, the total additional revenue generated by nodes $b^1, \cdots, b^{l-1}$ by cheating should exceed the loss of revenue at node b^l (who then needs to get compensated by $b^1, \cdots, b^{l-1}$). Therefore, to make this form of cheating unattractive, we need to enforce a specific form of pricing as follows.

Definition VIII.2. *If $\bar{\delta}_i^a \leq \delta_i^a$ (i.e., the delay ascribed for packet i to node a is less than the negotiated delay) then node a receives a fixed payment. On the other hand, if $\bar{\delta}_i^a > \delta_i^a$, then node a receives a* **negative** *payment $f(\bar{\delta}_i^a - \delta_i^a)$ where $f(x)$ is such that $|f(x)| >> |f(x_1)| + |f(x_2)|, \forall x = x_1 + x_2, x_1, x_2 > 0$*

This payment scheme works well in the case of colluding nodes as follows. Node b_l will be ascribed a total delay that includes the additional delay x_i (i.e., beyond the negotiated per-hop delay) the packet experienced at each of the nodes $b_1, \cdots, b_l$. If this additional delay is $x = \sum_{i=1}^{l} x_i$ then $|f(x)| >> \sum_{i=1}^{l} |f(x_i)|$. To see how this scheme helps, note that the reason nodes collude is to increase their profits. By delaying the packet for one connection, the colluding nodes may be able to forward packets for other connections faster thus making a greater profit. However, since b_l is being penalized, it needs to be compensated. Indeed, b_l needs to be compensated by at least $|f(x)|$ to make it worthwhile for it to participate in the deception. The cost to each node in the colluding set is thus $|f(x)|/(l-1) >> \sum_i |f(x_i)|$ Therefore, there is a net cost to colluding which will encourage nodes to not participate in this form of cheating.

Theorem VIII.1. *Solution 3 in combination with the payment scheme above ensures that nodes behave honestly.*

The proof follows from applying the above results. However, space limitations force us to omit the formal proof here.

IX. Applicability in practice

While the results presented in this paper are theoretical, we note that the proposed protocols can be deployed in practice relatively easily. The two main requirements we have are a need for a global time source and angels (for the first two protocols). There are two choices for time sources – GPS (Global Positioning Satellite) and the cellular phone infrastructure. However, these time sources need to generate unforgeable time stamps. Angels in our protocols are simple stateless computational pieces of software. Thus, we believe that if any user wishes to participate in the network, they ought to be required to download the angel software in addition to other software that they have to download anyway (e.g., routing software). Assuming these two pieces are in place, then given the existence of a trusted CA, our system is ready to use. However, as we noted previously, the time resolution of the global time source will place a restriction on the minimum verifiable delay per hop.

Several cities and companies are in the process of deploying wireless networks widely using different technologies such as

WiMAX or multi-hop 802.11a/b or combinations. In any of these networks, we can see a need for verifying delivered QoS since many of these networks will be charging for service. The feasibility of utilizing our solution is easy to see because the software needs are small and receipts are delivered offline to the CA. Indeed, since these networks will maintain extensive billing information for each user anyway (like today's cellular providers), the addition of a component for verifying and compensating users for forwarding packets is easy to accomplish.

X. Conclusions

In this paper we examine the problem of verifying the per-hop delay experienced by packets in wireless networks. The starting point is an impossibility theorem which mandates the need for a global nonce source and the presence of a trusted computing element at each node in order to do per-hop QoS monitoring. The protocols developed are cheat proof and we have expressions that bound the accuracy of the monitored delay. A third protocol does away with the need for the trusted computing element as well as the global nonce source but requires a particular form of payment to force honesty.

These protocols are the first attempt in the literature to look at verifying delivered QoS and as such opens up the intriguing possibility of building novel pricing schemes to encourage collaboration in wireless networks. These schemes may be modeled after a free market economy where nodes are free to make profit by forwarding and where payment is based on delivered performance. This appears to be an exciting research area particularly in the context of mixed cellular/ad hoc networks.

Acknowledgements

This work was funded by the NSF under a Medium ITR grant ANIR-0325014.

References

[1] http://www.endruntechnologies.com/index.htm, 2005.

[2] L. Anderegg and S. Eidenbenz. Ad hoc vcg: a truthful and cost-effecient routing protocol for mobile ad hoc networks with selfish agents. In *ACM Mobicom*, San Diego, CA, September 2003.

[3] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption: analysis of the DES modes of operation. In *Proceedings of 38th Annual Symposium on Foundations of Computer Science (FOCS 97)*, 1997.

[4] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. In H. Krawczyk, editor, *Advances in Cryptology – CRYPTO '98*, volume 1462, pages 232–249, 1998.

[5] S. Buchegger and J. Le Boudec. Performance analysis of the confidant protocol: Cooperation of nodes — fairness in dynamic ad-hoc networks. In *Proceedings of IEEE/ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHOC), Lausanne, CH*, June 2002.

[6] L. Buttyan and J. P. Hubaux. Stimulating cooperation in self-organizing mobile ad hoc networks. *ACM Journal; for Mobile Networks (MONET)*, 2002.

[7] William Feller. *An introduction to probability theory and its applications*, volume I. Wiley, 1968.

[8] S. Goldwasser and S. Micali. "Probabilistic encryption". *J. of Computer and System Sciences*, 28, April 1984.

[9] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attack. *SIAM Journal on Computing*, 17:281–308, 1988.

[10] J. P. Hubaux, L. Buttyan, and S. Capkun. The quest for security in mobile ad hoc networks. In *MOBIHOC*, Long Beach, CA, October 2001.

[11] O. Ileri, S-C. Mau, and N. B. Mandayam. Pricing for enabling forwarding in self-configuring ad hoc networks. In *IEEE WCNC*, Atlanta, GA, March 2004.

[12] P. Rogaway and T. Shrimpton. Cryptographic hash-function basics: Definitions, implications and separations for preimage resistance, second-preimage resistance, and collision resistance. In *Fast Software Encryption (FSE 2004)*, Lecture Notes in Computer Science, 2004.

[13] N. Salem, L. Buttyan, J. Hubaux, and M. Jakobsson. A charging and rewarding scheme for packet forwarding in multi-hop cellular networks, 2003.

[14] N. B. Salem, L. Buttyan, J. P. Hubaux, and M. Jakobsson. A charging and rewarding scheme for packet forwarding in multi-hop cellular networks. In *MOBIHOC*, Annapolis, MD, June 1 – 3 2003.

[15] S. Sundaramurthy and E. M. Belding-Royer. The ad-mix protocol for encouraging participation in in mobile ad hoc networks. In *IEEE ICNP*, Atlanta, GA, 2003.

[16] Hitesh Tewari and Donal O'Mahony. Multiparty micropayments for ad hoc networks. In *IEEE WCNC*, 2003.

[17] S. Zhong, Y. Yang, and J. Chen. Sprite: A simple, cheat-proof, credit-based system for mobile ad hoc networks. In *IEEE INFOCOM*, 2003.