Efficient and Secure Source Authentication for Multicast

Adrian Perrig[†*] Ran Canetti[‡] Dawn Song[†] J. D. Tygar[†]

[†]UC Berkeley, [*]Digital Fountain, [‡]IBM T.J. Watson

{perrig,dawnsong,tygar@cs.berkeley.edu, canetti@watson.ibm.com}

Abstract

One of the main challenges of securing multicast communication is source authentication, or enabling receivers of multicast data to verify that the received data originated with the claimed source and was not modified enroute. The problem becomes more complex in common settings where other receivers of the data are not trusted, and where lost packets are not retransmitted.

Several source authentication schemes for multicast have been suggested in the past, but none of these schemes is satisfactorily efficient in all prominent parameters. We recently proposed a very efficient scheme, TESLA, that is based on initial loose time synchronization between the sender and the receivers, followed by delayed release of keys by the sender.

This paper proposes several substantial modifications and improvements to TESLA. One modification allows receivers to authenticate most packets as soon as they arrive (whereas TESLA requires buffering packets at the receiver side, and provides delayed authentication only). Other modifications improve the scalability of the scheme, reduce the space overhead for multiple instances, increase its resistance to denial-of-service attacks, and more.

1 Introduction

With the growth and commercialization of the Internet, simultaneous transmission of data to multiple receivers becomes a prevalent mode of communication. Often the transmitted data is streamed and has considerable bandwidth. To avoid having to send the data separately to each receiver, several multicast routing protocols have been proposed and deployed, typically in the IP layer. (Examples include [12, 13, 23, 16, 6]). The underlying principle of multicast communication is that each data packet sent from the source reaches a number of receivers.

Securing multicast communication introduces a number of difficulties that are not encountered when trying to secure unicast communication. See [9] for a taxonomy of multicast security concerns and some solutions. A major concern is *source authentication*, or allowing a receiver to ensure that the received data is authentic (i.e., it originates with the source and was not modified on the way), even when none of the other receivers of the data is trusted. Providing source authentication for multicast communication is the focus of this work.

Simply deploying the standard point-to-point authentication mechanism (i.e appending a message authentication code to each packet, computed using a shared key) does not provide source authentication in the case of multicast. The problem is that any receiver that has the shared key can forge data and impersonate the sender. Consequently, it is natural to look for solutions based on asymmetric cryptography to prevent this attack, namely digital signature schemes. Indeed, signing each data packet provides good source authentication; however, it has high overhead, both in terms of time to sign and verify, and in terms of bandwidth. Several schemes were proposed that mitigate this overhead by amortizing a single signature over several packets, e.g. [14, 33, 29]. However, none of these schemes is fully satisfactory in terms of bandwidth and processing time, especially in a setting where the transmission is lossy and some data packets may never arrive. Even though some schemes amortize a digital signature over multiple data packets, a serious denial-of-service attack is usually possible where an attacker floods the receiver with bogus packets supposedly containing a strong signature. Since signature verification is computationally expensive, the receiver is overwhelmed verifying the signatures.

Another approach to providing source authentication uses only symmetric cryptography, more specifically on message authentication codes (MACs), and is based on delayed disclosure of keys by the sender. This technique was first used by Cheung [11] in the context of authenticating communication among routers. It was then used in the Guy Fawkes protocol [1] for interactive unicast communication. In the context of multicast streamed data it

was proposed by several authors [8, 4, 5, 25]. In particular, the TESLA scheme described in [25] was presented to the reliable multicast transport (RMT) working group [26] of the IETF and the secure multicast (SMuG) working group [30] of the IRTF and was favorably received. TESLA is particularly well suited to provide the source authentication functionality for the MESP header [10], or for the ALC protocol proposed by the RMT [19]. Consequently, an Internet-Draft describing the scheme was recently written [24].

The main idea of TESLA, is to have the sender attach to each packet a MAC computed using a key k known only to itself. The receiver buffers the received packet without being able to authenticate it. If the packet is received too late, it is discarded. A short while later, the sender discloses k and the receiver is able to authenticate the packet. Consequently, a single MAC per packet suffices to provide source authentication, provided that the receiver has synchronized its clock with the sender ahead of time.

This idea seems quite attractive at first. However, it has several shortcomings. This work points to these shortcomings and proposes methods to overcome them. Our description is based mostly on TESLA, although the improvements apply to the other schemes as well. We sketch some of these points:

1. In TESLA the receiver has to buffer packets, until the sender discloses the corresponding key, and until the receiver authenticates the packets. This may delay delivering the information to the application, may cause storage problems, and also generates vulnerability to denial-of-service (DoS) attacks on the receiver (by flooding it with bogus packets). We propose a method that allows receivers to authenticate most packets immediately upon arrival, thus reducing the need for buffering at the receiver side and in particular reduces the susceptibility to this type of DoS attacks.

   This improvement comes at the price of one extra hash per packet, plus some buffering at the sender side. We believe that buffering at the sender side is often more reasonable and acceptable than buffering at the receiver side. In particular, it is not susceptible to this type of DoS attacks.

   We also propose other methods to alleviate this type of DoS attacks. These methods work even when the receiver buffers packets as in TESLA.

2. When operating in an environment with heterogenous network delay times for different receivers, TESLA authenticates each packet using multiple keys, where the different keys have different disclosure delay times. This results in larger overhead, both in processing time and in bandwidth. We propose a method for achieving the same functionality (i.e., different receivers can authenticate the packets at different delays) with a more moderate increase in the overhead per packet.

3. In TESLA the sender needs to perform authenticated time synchronization individually with each receiver. This may not scale well, especially in cases where many receivers wish to join the multicast group and synchronize with the sender at the same time. This is so, since each synchronization involves a costly public-key operation. We propose a method that uses only a single public-key operation per time-unit, regardless of the number of time synchronizations performed during this time unit. This reduces the cost of synchronizing with a receiver to practically the cost of setting up a simple, unauthenticated connection.

4. We also explore time synchronization issues in greater depth and describe direct and indirect time synchronization. For the former method, the receiver synchronizes its time directly with the sender, in the latter method both the sender and receiver synchronize their time with a time synchronization server.

   For both cases, we give a detailed analysis on how to choose the key disclosure delay, a crucial parameter for TESLA.

5. TESLA assumes that all members have joined the group and have synchronized with the sender before any transmission starts. In reality, receivers may wish to join after the transmission has started; furthermore, receivers may wish to receive the transmission immediately, and perform the time synchronization only later. We propose methods that enable both functionalities. That is, our methods allow a receiver to join in "on the fly" to an ongoing session; they also allow receivers to synchronize at a later time and authenticate packets only then.

Organization Section 2 reviews TESLA, providing further details than in [25]. Section 3 contains the improvements and extensions proposed in this paper. Section 4 provides further discussion on the security of the improved scheme, with emphasis on resistance to denial-of-service attacks.

2 An Overview of TESLA

The security property TESLA guarantees is that the receiver never accepts M_i as an authentic message unless M_i was actually sent by the sender. Note that TESLA does not provide non-repudiation, that is, the receiver cannot convince a third party that the stream arrived from the claimed source.

TESLA is efficient and has a low space overhead mainly because it is based on symmetric-key cryptography. Since source authentication is an inherently asymmetric property (all the receivers can verify the authenticity but they cannot produce an authentic data packet), we use a delayed disclosure of keys to achieve this property. Similarly, the data authentication is delayed as well. In practice, the authentication delay is on the order of one round-trip-time (RTT).

TESLA has the following properties. First, it has a low computation overhead, which is typically only one MAC function computation per packet, for both sender and receiver. TESLA also has a low per-packet communication overhead, which is about 20 bytes per packet. In addition, TESLA tolerates arbitrary packet loss. Each packet that is received in time can be authenticated. Except for an initial time synchronization, it has only unidirectional data flow from the sender to the receiver. No acknowledgments or other messages are necessary. This implies that the sender's stream authentication overhead is independent of the number of receivers, hence TESLA is very scalable. TESLA can be used both in the network layer or in the application layer. The delayed authentication, however, requires buffering of packets until authentication is completed.

For TESLA to be secure, the sender and the receiver need to be loosely time synchronized, which means that the synchronization does not need to be precise, but the receiver needs to know an upper bound on the sender's time.

2.1 Sender Setup

In our model, a sender distributes a stream of data composed of message chunks $\{M_i\}$. Generally, the sender sends each message chunk M_i in one network packet P_i. Many multicast distribution protocols do not retransmit lost packets. The goal is therefore that the receiver can authenticate each message chunk M_i separately.

For the purpose of TESLA, the sender splits the time into even intervals I_i. We denote the duration of each time interval with T_{int}, and the starting time of the interval I_i is T_i. Trivially, we have $T_i = T_0 + i * T_{int}$. In each interval, the sender may send zero or multiple packets.

Before sending the first message, the sender determines the sending duration (possibly infinite), the interval duration, and the number N of keys of the key chain. This key chain is analogous to the one-way chain introduced by Lamport [18], and the S/KEY authentication scheme [15]. The sender picks the last key K_N of the key chain randomly and pre-computes the entire key chain using a pseudo-random function F, which is by definition a one-way function. Each element of the chain is defined as $K_i = F(K_{i+1})$. Each key can be derived from K_N as $K_i = F^{N-i}(K_N)$, where $F^j(k) = F^{j-1}(F(k))$ and $F^0(k) = k$. Each key of the key chain corresponds to one interval, i.e., K_j is active in interval I_j.

Since we do not want to use the same key multiple times in different cryptographic operations, we use a second pseudo-random function F' to derive the key which is used to compute the MAC of messages in each interval (we will explain the algorithm in detail later). Hence, $K'_i = F'(K_i)$. Figure 1 depicts this key derivation. We propose to use HMAC in conjunction with a cryptographically secure hash function for the pseudo-random function [2]. For example, a possibility is to use the following: $F(x) = \text{HMAC}(x, 0)$ and $F'(x) = \text{HMAC}(x, 1)$, where 0 and 1 are 8-bit integers. Note that the first argument of the MAC function is the key and the second argument is the data.

2.2 Bootstrapping a new Receiver

TESLA requires an initially authenticated data packet to bootstrap a new receiver. This authentication is achieved with a digital signature scheme, such as RSA [28], or DSA [32].

We consider two options for synchronizing the time, direct and indirect synchronization. We improve the time synchronization from our original work and describe the details in section 3.3. Whichever time synchronization mechanism is used, the receiver only needs to know an upper bound on the sender time.

The initial authenticated packet contains the following information about the time intervals and key chain:

- The beginning time of a specific interval T_j, along with its id I_j
- The interval duration T_{int}
- Key disclosure delay d (unit is interval)
- A commitment to the key chain K_i ($i < j - d$ where j is the current interval index)

2.3 Sending Authenticated Packets

Each key of the key chain is used in one time interval. However many messages are sent in each interval, the key which corresponds to that interval is used to compute the MAC of all those messages. This allows the sender to send packets at any rate and to adapt the sending rate dynamically. The key remains secret for d-1 future intervals. Packets sent in interval I_j can hence disclose key K_{j-d}. As soon as the receivers receive that key, they can verify the authenticity of the packets sent in interval I_{j-d}.

The construction of packet P_j sent in interval I_i is: $\{M_j \mid \text{MAC}(K'_i, M_j) \mid K_{i-d}\}$.

Figure 1 shows the key chain construction and the MAC key derivation. If the disclosure delay is 2 intervals, the packet P_{j+4} sent in interval I_{i+2} discloses key K_i. From this key, the receiver can also recover K_{i-1} and verify the MAC of P_j, in case P_{j+3} is lost.

2.4 Receiver Tasks

Since the security of TESLA depends on keys that remain secret until a pre-determined time period, the receiver must verify for each packet that the key, which is used to compute the MAC of that packet, is not yet disclosed by the sender. Otherwise, an attacker could have changed the message data and re-computed the MAC. This motivates the security condition, which the receiver must verify for each packet it receives.

Security condition: A packet arrived *safely*, if the receiver is assured that the sender cannot yet be in the time interval in which the corresponding key is disclosed.

The intuition is that if a packet satisfies the security condition, then no attacker could have altered it in transit, because the corresponding MAC key is not yet disclosed. In case the security condition is not valid, the receiver must drop that packet, because the authenticity is not assured any more. We would like to emphasize that the security of this scheme does not rely on any assumptions on network propagation delay. The original paper sketches a security proof [25].

We now explain how the authentication with TESLA works with a concrete example. When the receiver receives packet P_j sent in interval I_i at local time t_c, it computes an upper bound on the sender's clock t_j (we describe in section 3.3 how to compute this). To evaluate the security condition, the receiver computes the highest interval x the sender could possibly be in, which is $x = \lfloor (t_j - T_0)/T_{int} \rfloor$. The receiver now verifies that $x < I_i + d$ (where I_i is the interval index), which means that the sender must not have been in the interval in which the key K_i is disclosed, hence no attacker can possibly know that key and spoof the message contents.

The receiver cannot, however, verify the authenticity of the message yet. Instead, it stores the triplet $(I_i, M_j, \text{MAC}(K'_i, M_j))$ to verify the authenticity later when it knows K'_i. Two possibilities exist on how to handle the unauthenticated message chunk M_j. The first possibility is to hand M_j to the application, and notify it through a callback mechanism as soon as M_j is verified. The second possibility is to buffer M_j until the authenticity can be checked and pass it to the application as soon as M_j is authenticated.

If the packet contains a disclosed key K_{i-d}, regardless of whether the security condition is verified or not, the receiver checks whether it can use K_{i-d} to authenticate previous packets. Clearly, if it has received K_{i-d} previously, it does not have any work to do. Otherwise, let us assume that the last key value in the reconstructed key chain is K_v. The receiver verifies if K_{i-d} is legitimate by verifying that $K_v = F^{i-d-v}(K_{i-d})$. If that condition is correct, the receiver updates the key chain. For each new key K_w, it computes $K'_w = F'(K_w)$ which might allow it to verify the authenticity of previously received packets.

It is clear that this system can tolerate arbitrary packet loss, because the receiver can verify the authenticity of all received packets that satisfy the security condition eventually.

3 Our Extensions

We extend TESLA in a number of ways to make it more efficient and practical. First, we present a new method to support *immediate authentication*, meaning that the receiver can authenticate packets as soon as they arrive.

Second, we propose optimizations concerning key chains. In particular, for applications that use multiple authentication chains with different disclosure delays, we present a new algorithm that reduces the communication overhead.

Finally, we give discussions on the time synchronization issues and derive a tight lower bound on the key disclosure delay, which makes the scheme much more practical. Next, we remove a scalability limitation of the simple time synchronization protocol. Furthermore, we discuss how a receiver can authenticate received packets even if it is not time synchronized at the moment in which it receives the packet.

3.1 Immediate Authentication

A drawback of the original TESLA protocol is that the receiver needs to buffer packets during one disclosure delay before it can authenticate them. This might not be practical for certain applications if the receivers cannot afford much buffer space and bursty traffic might cause the receivers to drop packets due to insufficient buffer space. Moreover, as we show later in section 4.2, the requirement of receiver buffering introduces a vulnerability to a denial-of-service attack. To solve these problems caused by receiver-buffering, we propose a new method to support *immediate authentication*, which allows the receiver to authenticate packets as soon as they arrive.

The basic observation of this method is that we can replace receiver buffering with sender buffering. If the sender can buffer packets during one disclosure delay, then it could store the hash value of the data of a later packet in an earlier packet and hence as soon as the earlier packet is authenticated, the data in the later packet is authenticated through the hash value as well.

In the new scheme, the sender buffers packets for the

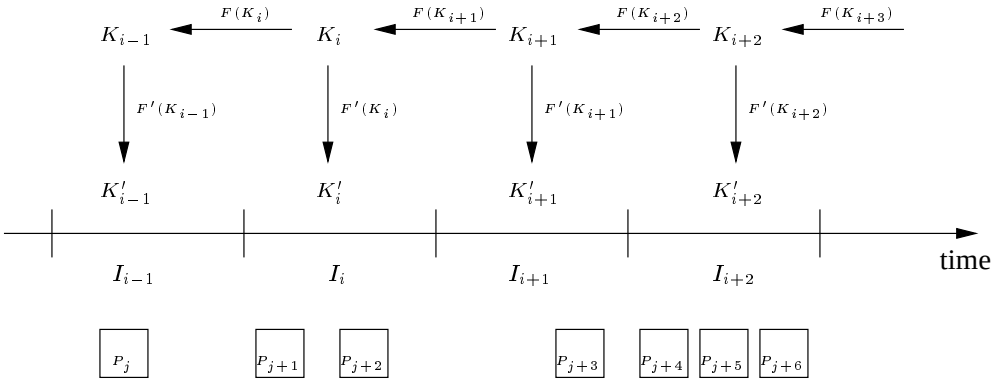

Figure 1: TESLA key chain and the derived MAC keys

duration of one disclosure delay. For simplicity of illustration, we assume that the sender sends out a constant number v of packets per time interval. To construct the packet for the message chunk M_j in time interval T_i, the sender appends the hash value of the message chunk M_{j+vd} to M_j and then computes the MAC value also over $H(M_{j+vd})$ with the key K_i. Figure 2 illustrates how the packet P_j is constructed by appending $H(M_{j+vd})$, MAC$(K_i, M_j \mid H(M_{j+vd}))$, along with the disclosed key K_{i-d}. (Note that the $\mid$ stands for message concatenation). When the packet P_{j+vd} arrives at the receiver which discloses the key K_i it allows authentication of packet P_j sent in interval I_i. P_j carries a hash of the data M_{j+vd} in P_{j+vd}. If P_j is authentic, $H(M_{j+vd})$ is also authentic and therefore the data M_{j+vd} is immediately authenticated. Also note that if P_j is lost or dropped due to violation of the security condition, P_{j+vd} will not be immediately authenticated and can still be authenticated later using the MAC value.

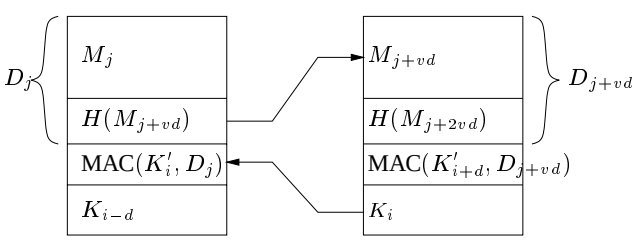

Figure 2: Immediate authentication packet example. $D_j = H(M_{j+vd}) \mid M_j$ and $D_{j+vd} = H(M_{j+2vd}) \mid M_{j+vd}$.

If each packet can only carry the hash of one other packet, it is clear that the sending rate needs to remain constant. Also it is clear that if a packet is lost, the corresponding packet cannot be immediately authenticated. To achieve flexibility for dynamic sending rate and robustness to packet loss, the sender can add the hash values of multiple future packets to a packet, similar to the EMSS scheme [25].

3.2 Concurrent TESLA instances

In this section, we present a space optimization technique in the case the sender uses multiple TESLA instances for one stream.

Choosing the disclosure delay involves a tradeoff. Receivers with a low network delay welcome short key disclosure delays because that translates into a short authentication delay. Unfortunately, receivers with a long network delay could not operate with a short disclosure delay because most of the packets will violate the security condition and hence cannot be authenticated. Conversely, a long disclosure delay would suit the long delay receivers, but causes unnecessarily long authentication delay for the receivers with short network delay. The solution is to use multiple instances of TESLA with different disclosure delays simultaneously, and each receiver can decide which disclosure delay, and hence, which instance to use. A simple approach to use concurrent TESLA instances is to treat each TESLA instance independently, with one key chain per instance. The problem for this approach is that each extra TESLA instance also causes extra space overhead in each packet. If each instance requires 20 bytes per packet (80 bit for key disclosure and 80 bit for the MAC value), using three instances results in 60 bytes space overhead per packet. We present a new optimization which reduces the space overhead of concurrent instances.

The main idea is that instead of using one independent key chain per TESLA instance, we could use the same key chain but a different key schedule for all instances. The basic scheme works as follows. All TESLA instances for a stream share the same time interval duration and the same key chain. Each key K_i in the key chain is associated with the corresponding time interval T_i, and K_i will

be disclosed in T_i.[1] Assume that the sender uses w instances of TESLA, which we denote with $\tau_1 \ldots \tau_w$. Each TESLA instance τ_u has a different disclosure delay d_u, and it will have a MAC key schedule derived from the key schedule shifted by d_u time intervals from the key disclosure schedule. Let $K^u_{i+d_u}$ denote the MAC key used by instance u in time interval T_i. We derive $K^u_{i+d_u}$ as $K^u_{i+d_u} = \text{HMAC}(K_{i+d_u}, u)$. Note that we use HMAC as a pseudo-random function, which is the same key derivation construction as we use in TESLA (see section 2.1 and figure 1). In fact, the keys of the first instance are derived with the same pseudo-random function as the TESLA protocol that uses only one instance. The reason for generating all different, independent keys for each instance is to prevent an attack where an attacker moves the MAC value of an instance to another instance, which might allow it to claim that data was sent in a different interval. Our approach of generating independent keys prevents this attack. Thus to compute the MAC value in packet P_j in time interval T_i, the sender computes one MAC value of the message chunk M_j per instance and append the MAC values to M_j. In particular, for the instance τ_u with disclosure delay d_u, the sender will now use the key $K^u_{i+d_u}$ as mentioned above for the MAC computation.

Figure 3 shows an example with two TESLA instances, one with a key disclosure time of two intervals and the other of four intervals. The lowest line of keys shows the key disclosure schedule, i.e. which key is disclosed in which time interval. The middle and top line of keys shows the key schedule of the first and second instance respectively, i.e. which key is used to compute the MAC for the packets in the given time interval for the given instance. Using this technique, the sender will only need to disclose one key chain no matter how many instances are used concurrently. If each disclosed key is 10 bytes long, then for a stream with m concurrent instances, this technique will save $10(m-1)$ bytes per packet, which is a drastic saving in particular for small packets.

3.3 Time Synchronization Issues

Loose time synchronization is an important component in TESLA. Although sophisticated time synchronization protocols exist, they usually require considerable management overhead. Furthermore, they generally have a high complexity and achieve properties that TESLA does not require. An example is the network time protocol (NTP) by Mills [21]. Bishop performs a detailed security analysis of NTP [7]. For these reasons, we outline a simple and secure time synchronization protocol that suffices the humble requirements of TESLA.

[1] Note that this key schedule is different from the previous schedule described in section 2.1, where key K_i was used to compute the MAC in interval I_i and was disclosed in interval I_{i+d}.

The time synchronization requirement that secures TESLA against an active attacker is that the receiver knows an upper bound of the difference between the sender's local time and the receiver's local time, Δ. For simplicity, we assume the clock drift of both sender and receiver are negligible, otherwise they will simply resynchronize periodically. We denote the real difference between the sender and the receiver's time with δ. Hence for loose synchronization, the receiver does not need know δ but only some Δ that is guaranteed to be greater or equal to δ. To compute Δ, we can use either a direct or an indirect time synchronization method. In the following, we first discuss a simple protocol for direct time synchronization, and next we discuss how to do indirect time synchronization.

Direct Time Synchronization

In direct time synchronization, the receiver performs an explicit time synchronization with the sender. This approach has the advantage that no extra infrastructure is needed to perform the time synchronization. We design a simple two-phase protocol that satisfies the TESLA requirements.

In the protocol, the receiver first records its local sending time t_R and sends a time synchronization request containing a nonce to the sender. Upon receiving the time synchronization request, the sender records its local receiving time t_S and sends the receiver a signed response packet containing t_S and the nonce.

$$R \rightarrow S : \mathsf{Nonce}$$
$$S \rightarrow R : \{\mathsf{Sender\ time}\ t_S, \mathsf{Nonce}\}_{K_S^{-1}}$$

Figure 4 shows a sample time synchronization between the receiver and the sender. Upon receiving the signed response, the receiver checks the validity of the signature and the matching of the nonce and computes $\Delta = t_S - t_R$. It is easy to see that the Δ computed this way satisfies the requirement that $\Delta \geq \delta$. Because $\Delta = t_S - t_R = (t_S - t_3) + (t_3 - t_R)$, $t_S - t_3 = \delta$, and $t_3 - t_R$ is the network delay for sending the request from the receiver to the sender which is greater or equal to 0, hence $\Delta \geq \delta$. An interesting point is that the network delay of the response packet and the delay caused by the computation of the digital signature do not influence Δ at all. Since only the initial timestamp matters, it is important that the sender immediately stores the arrival time t_S of the time synchronization request packet. The subsequent processing and propagation delay does not matter.

Because the digital signature operation is computationally expensive, we need to be careful about denial-of-service attacks where an attacker floods the sender with

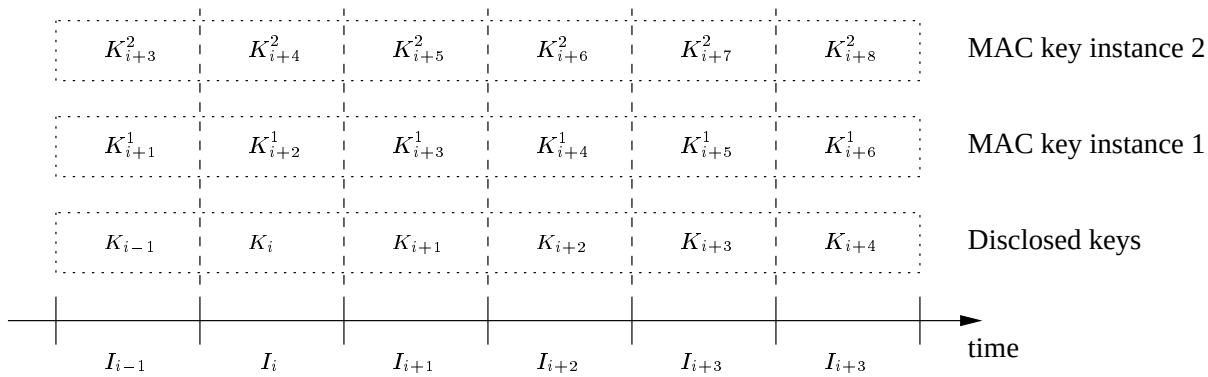

Figure 3: Multiple TESLA instances key chain optimization.

time synchronization requests. Section 4.1 addresses this issue.

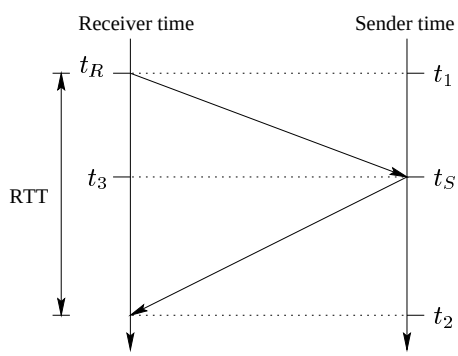

Figure 4: The receiver synchronizes its time with the sender.

Indirect Time Synchronization

In indirect time synchronization, both the sender and the receivers synchronize their time with a time reference and hence the sender and the receiver can reach implicit time synchronization. This approach is favorable especially in cases where the application needs time synchronization with a time reference anyhow. Let $\Delta_{SC} + |\epsilon_{SC}|$ denote the measured upper bound of the difference of the sender's time and the time reference's time with $|\epsilon_{SC}|$ as the maximum error, and let $\Delta_{CR} + |\epsilon_{CR}|$ denote the measured upper bound of the difference of the time reference's time and the receiver's time with $|\epsilon_{CR}|$ as the maximum error. Thus the receiver could reach an implicit time synchronization with the sender as $\Delta = \Delta_{SC} + \Delta_{CR} + |\epsilon_{SC}| + |\epsilon_{CR}|$ with $\epsilon = |\epsilon_{SC}| + |\epsilon_{CR}|$ as the maximum error.

In settings where the receiver is already time synchronized with the time reference, the receiver does not need to send any information to the sender. The sender just needs to periodically broadcast digitally signed packets that contain its time synchronization with the time reference, the time interval and key chain information outlined in section 2.2, along with the sender's maximum synchronization error ϵ_{SC}. A new receiver can start authenticating the data stream right after it receives one of the signed advertisements. This is particularly useful in the case of satellite broadcast.

Delayed Time Synchronization

Another interesting relaxation of the time synchronization requirement is that, if we assume that the receiver's clock drift is negligible during a period of time, then the receiver can receive the data stream from the sender before doing a time synchronization and authenticate the data later after a time synchronization. The receiver only needs to store the arrival time of each packet, so that it can evaluate the security condition after it performed the time synchronization. This is highly useful for many applications, for example a router can use TESLA to authenticate itrace messages [3], and the victim can authenticate the routers' IP markings afterwards when it wants to trace an attacker by performing an approximate time synchronization with the router [31].

3.4 Determining the Key Disclosure Delay

An important parameter to determine for TESLA is the key disclosure delay d. A short disclosure delay will cause packets to violate the security condition and cause packet drop, while a long disclosure delay causes a long authentication delay. Note that although the choice of the disclosure delay does not affect the security of the system, it is an important performance factor. We describe a new method on how to choose a good disclosure delay d. In particular, we show as follows that if RTT is a reasonable upper bound on the round trip time between the receiver and the sender, then in case of using direct time synchronization, we can choose $d = \lceil RTT/T_{int} \rceil + 1$, where T_{int}

is the interval duration. In case of indirect time synchronization, we can choose $d = \lceil (D_{SR}+\epsilon)/T_{int} \rceil + 1$, where ϵ is the sum of both the sender and receiver time synchronization error, and D_{SR} is a reasonable upper bound on the network delay of a packet traveling from the sender to the receiver.

Consider a packet P_i that is constructed using the MAC key K'_j in time interval I_j which will be disclosed d time intervals later. The packet P_i arrives at the receiver at its local time t_i^R. Hence the security condition is that

$$\lfloor \frac{t_i^R + \Delta - T_0}{T_{int}} \rfloor - I_j < d, \qquad (1)$$

where T_0 is the beginning time of the 0th time interval and T_{int} is the time interval duration. Assume packet P_i was sent at the sender's local time t_i^S. Hence $t_i^S < T_j + T_{int} = I_j \cdot T_{int} + T_0 + T_{int}$. We denote the average network delay time from the sender to the receiver with D_{SR} and the average network delay time from the receiver to the sender is D_{RS}, and hence $RTT = D_{RS} + D_{SR}$.

In case of a direct time synchronization, using the same notation as in section 3.3, $\Delta = \delta + (t_3 - t_R) \doteq \delta + D_{RS}, t_i^R + \delta - t_i^S \doteq D_{SR}$, and hence we can derive at the end that a tight bound for d to satisfy the equation 1 is $d = \lceil RTT/T_{int} \rceil + 1$, which allows most of packets to satisfy the security condition and still the receiver would not need to wait much extra longer than necessary to authenticate the packets. Similarly in case of an indirect time synchronization, we can derive that a good d is $d = \lceil (D_{SR} + \epsilon)/T_{int} \rceil + 1$.

4 Security Discussion and Robustness to DoS

Our original paper did not address denial-of-service (DoS) attacks on TESLA. In an IP multicast environment, however, DoS is a considerable threat and requires careful consideration. We discuss potential security problems in this section and show how to strengthen TESLA to thwart them. In particular, we show that there is no DoS attack on the sender if the receivers perform indirect time synchronization. In case of direct time synchronization, we show how to mitigate DoS attacks on the sender. Although there are some potential DoS attacks on the receiver side, we show that TESLA does not add any additional vulnerability to DoS attacks if the receiver has a reasonable amount of buffer space, otherwise we describe schemes that alleviate the exposure to DoS.

4.1 DoS Attack on the Sender

A DoS attack on the sender is not possible if TESLA is used with indirect time synchronization, because the sender does not keep per-receiver state or perform per-receiver operations. In the case of direct time synchronization, a DoS attack is possible, since the sender is required to digitally sign each nonce included in a time synchronization request. An attacker can perform a DoS by flooding the sender with requests.

This response packet needs to be authenticated with a digital signature scheme, such as RSA [28], or DSA [32]. Since public-key signature algorithms are computationally expensive, the signing of the response packet can become a performance bottleneck for the sender. A simple trick can alleviate this situation. The sender can aggregate multiple requests, compute and sign a Merkle hash tree that is generated from all the requester's nonces [20]. Figure 5 shows how such a hash tree is constructed. If N_h is the root of the hash tree, N_h would be included in the signed part of the response packet instead of the receiver's nonce N_r. To verify the digital signature of the response packet, each receiver would reconstruct the hash tree. Since it does not know the other receiver's nonces that are part of the hash tree, the sender would include the nodes of the tree necessary to reconstruct the root node. For the example in figure 5, the packet returned to receiver A would include N_b and H_{cd}. Receiver A can reconstruct the root node H_{ad} from these values and its own nonce N_a as follows: $H_{ad} = H(H(N_a, N_b), H_{cd})$. Note that the number of nodes returned in the response packet is logarithmic in the number of receivers whose request arrived in the same time interval. Assuming a 50 ms interval time (the sender would need to compute at most 20 signatures per second) and assuming that 1,000,000 receivers wanted to synchronize their time in that interval, the return packet would only need to contain 20 hash nodes or 200 bytes, assuming an 80 bit hash function. Any cryptographically secure hash function can be used for $H(x, y)$, for example MD5 [27], SHA-1 [17], or RIPEMD-160.

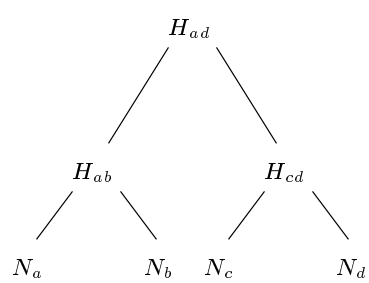

Figure 5: Hash tree over receiver nonces. Node $H_{ab} = H(N_a, N_b)$. $H_{ad} = H(H_{ab}, H_{cd})$.

4.2 DoS Attack on the Receiver

In this section, we discuss two DoS attacks on the client. Since we assume the attacker could have full control of the

network, some DoS attacks such as delay or drop packets are always possible. Delay packets could cause packets to violate the security condition and hence not to be authenticated. On the other hand, speeding up packets does not do anything at all. The receiver even benefits from this since she might be able to use a chain with a short disclosure delay that she could not use otherwise. We can show that replay packets cannot do much harm either. First, a duplicated packet is only accepted by the receiver within a short time period, since the security condition drops packets if they are replayed with a long delay. Second we can prevent the replay attack by adding a sequence number to each packet and by including the sequence number in the MAC. The TESLA protocol in the network layer or in the application layer will filter out duplicate packets.

In the rest of the subsection, we discuss some more complicated DoS attacks and show how to mitigate or prevent the attacks. First we discuss a flooding attack which fills up the receiver buffers. Second we discuss an attack that tries to waste the receiver's computation resources by unnecessarily re-computing the key chain.

DoS on the Packet Buffer

An powerful attack is to flood the multicast group with bogus traffic. This attack is serious because current multicast protocols do not enforce sending access control.[2] The solution we propose involves a weak but efficient and immediate authentication method that offers some protection against a flooding attack.

First if the receiver has a certain size buffer, we show that flooding cannot do much harm. Because the scheme only requires the receiver to buffer packets for the duration of one disclosure delay until the authenticity of the packets can be verified, hence the buffer size only needs to be the multiplication of the network bandwidth and the disclosure delay time. Assuming that the receiver has a 10Mbps network connection and a 500ms disclosure delay, the required buffer size is around 640kB, which should in general not be a major concern with today's workstations. Assuming 512byte network packets, the computation overhead to authenticate the packets is on the order of 1280 HMAC computations per second. Since the openssl HMAC-MD5 implementation processes on the order of $120,000$ 512-byte blocks per second on a 500MHz Pentium III Linux workstation, the estimated processor overhead for TESLA authentication is on the order of 1% of the CPU time.

Second if the receiver's buffer size is not large enough as computed above, flooding could result in a DoS attack because the receiver would drop packets due to a lack of buffer space.[3]

An obvious solution is to distribute a shared secret key to all receivers and to add a MAC to each packet with the shared secret key. This enables a receiver to quickly verify the packet, but it allows an attacker who knows the key to flood the clients anyhow.

Another approach is to use the key chain as a weak authentication method. Briscoe presents a related method for immediate authentication [8]. The receiver pre-authenticates the packet by verifying that the disclosed key really is part of the key chain. Based on the disclosed key, the receiver can also immediately derive the time interval of the packet and also immediately verify the security condition. Both checks are efficient and do not require any additional space overhead in the packet. An attacker would need to receive a packet from the sender, extract the disclosed key, and use that key to flood the receivers. Fortunately, the flooding time period of each key is limited to one interval duration.

Yet another solution is to use the immediate authentication we propose in section 3.1. In this case, the message does not need to be added to a queue if it is immediately authenticated.

In practice, the receiver allocates a queue for each time interval to buffer incoming packets until they can be authenticated. If the receiver has too little memory to buffer all incoming traffic during the disclosure delay, it needs to decide on a drop or replacement policy in case of a full buffer. Dropping all packets of a particular interval once the buffer is full is a poor policy, because an attacker might insert the spoofed traffic only early in each time interval, causing the receivers to buffer mostly spoofed packets. Ideally, the receiver uses a random replacement policy once the buffer is full. For each incoming packet, the receiver picks a packet within the buffer to replace.

DoS on the Key Chain

Another DoS attack is specific to how the TESLA receiver reconstructs the key chain. If an attacker could fool a receiver to believe that a packet was sent out far in the future, and the receiver would try to verify the key disclosed in the packet by applying the pseudo-random function until the last committed key chain value. This attack can be easily prevented by checking that the packet interval is less or equal the latest interval that the sender can possibly be in. For an incoming packet sent in interval I_j, the receiver can verify if the interval I_j is not in the future, i.e. if the sender can already be in that interval. The ver-

[2] Source-Specific Multicast (SSM) is a new multicast protocol, and a new IETF working group was formed in August 2000 [22]. SSM tends to address this problem by enforcing that only one legitimate sender can send to the multicast group.

[3] We do not consider the flooding attack from a network perspective (where flooding can cause link congestion and results in dropping legitimate traffic) because any network protocol is susceptible to this attack.

ification condition is that $I_j < \lfloor (t_i - T_0)/T_{int} \rfloor$, where t_i is an upper bound on the sender's time that the receiver computes at the arrival of the packet.

5 Related Work

Researchers have proposed signing data packets to achieve source authentication. Since a digital signature achieves non-repudiation, a signature is much stronger than just authentication. As we mentioned in the introduction, the communication and computation overhead of current signature schemes is more expensive than schemes that are based on symmetric cryptography. We will review only the schemes that provide source authentication and not the schemes providing non-repudiation, i.e. [14, 29, 33, 25].

The earliest related work is by Cheung [11]. He proposes a scheme akin to the basic TESLA protocol to authenticate link-state routing updates between routers. He assumes that all the routers in a network are time synchronized up to $\pm\epsilon$, and does not consider the case of heterogeneous receivers.

Anderson et al. [1] present the Guy Fawkes protocol which provides message authentication between two parties. Their protocol has the drawback that it cannot tolerate packet loss. They propose two methods to guarantee that the keys are not revealed too soon. The first method is that the sender and receiver are in lockstep, i.e. the receiver acknowledges every packet before the sender can send the next packet. This severely limits the sending rate and does not scale to a large number of receivers. The second method to secure their scheme is to time-stamp each packet at a time-stamping service, which introduces additional complexity and overhead.

Canetti et al. propose to use k different keys to authenticate every message with k different MAC's for sender authentication [9]. Every receiver knows m keys and can hence verify m MAC's. The keys are distributed in such a way that no coalition of w receivers can forge a packet for a specific receiver. The communication overhead for this scheme is considerable, since every message carries k MAC's. The server must also compute k MACs before a packet is sent, which makes it more expensive than the scheme we present in this paper. Furthermore, the security of their scheme depends on the assumption that at most a bounded number (which is on the order of k) of receivers collude.

Briscoe proposes the FLAMeS protocol that is similar to the Cheung [11] and part of the basic TESLA protocol. Bergadano, Cavalino, and Crispo present an authentication protocol for multicast [5]. Their protocol is similar to Cheung [11] and to parts of the basic TESLA protocol.

Bergadano, Cavagnino, and Crispo, propose a protocol similar to the Guy Fawkes protocol to individually authenticate data streams sent within a group [4]. Their scheme requires that the sender receives an acknowledgment packet from each receiver before it can send the next packet. This prevents scalability to a large group. The advantage is that their protocol does not rely on time synchronization.

Unfortunately, their protocol is vulnerable to a man-in-the-middle attack. To illustrate the attack, we briefly review the protocol for one sender and one receiver (adapted to use the same notation as we established in this paper):

$$
\begin{aligned}
&B \rightarrow A : KB_0, SN, \{KB_0, SN\}_{K_B^{-1}} \\
&A \rightarrow B : A_1, MAC(KA_1, A_1), KA_0, SN, \{KA_0, SN\}_{K_A^{-1}} \\
&B \rightarrow A : KB_1 \\
&A \rightarrow B : A_2, MAC(KA_2, A_2), KA_1
\end{aligned}
$$

In their scheme, both A (the sender) and B (the receiver) pre-compute a key chain, KA_i and KB_i, respectively. In the following attack, B intends to authenticate data from A, but we will show that the attacker I can forge all data. The attacker I captures all messages from B and it can pretend to B that all the messages come from A. To A, the attacker I just pretends to be itself.

$$
\begin{aligned}
B \rightarrow I(A) &: KB_0, SN, \{KB_0, SN\}_{K_B^{-1}} \\
I \rightarrow A &: KI_0, SN, \{KI_0, SN\}_{K_I^{-1}} \\
A \rightarrow I &: A_1, MAC(KA_1, A_1), KA_0, SN, \{KA_0, SN\}_{K_A^{-1}} \\
I \rightarrow A &: KI_1 \\
A \rightarrow I &: A_2, MAC(KA_2, A_2), KA_1 \\
I(A) \rightarrow B &: A_1', MAC(KA_1, A_1'), KA_0, SN, \{KA_0, SN\}_{K_A^{-1}}
\end{aligned}
$$

Note that the attacker I can forge the content of the message A_1 sent to B, because it knows the key KI_0. The attacker I can forge the entire subsequent message stream, without B noticing.

Another attack is that an eavesdropper that records a message exchange between A (sender) and B (receiver) can impersonate either A or B as a receiver to another sender C. This attack can be serious if the sender performs access control based on the initial signature packet and the revealed key chain. The attack is simple, the eavesdropper only needs to replay the initial signatures and all the disclosed keys collected.

6 Conclusions

In this paper, we have presented an extension to our TESLA scheme which provides a solution to the source authentication problem under the assumption that the

sender and receiver are loosely time synchronized. The basic TESLA protocol has the following salient properties:

- Low computation overhead. On the order of one MAC function computation per packet for both sender and receiver.
- Low communication overhead. Required is as little as one MAC value per packet. Periodically, the sender also needs to send out the secret keys.
- Perfect loss robustness. If a packet arrives in time, the receiver can verify its authenticity eventually (as long as it receives later packets).

The extensions we propose in this paper feature:

- The basic TESLA scheme provides delayed authentication. With additional information in a packet, we show in this paper how we can provide immediate authentication.
- We reduce the communication overhead when multiple TESLA instances with different authentication delays are used concurrently.
- We derive a tight lower bound on the disclosure delay.
- Harden the sender and the receiver against denial-of-service attacks.

7 Acknowledgments

We would like to thank Radia Perlman for the discussions on DoS attacks. We are also grateful to Bob Briscoe for helpful discussions and feedback.

References

[1] R. J. Anderson, F. Bergadano, B. Crispo, J.-H. Lee, C. Manifavas, and R. M. Needham. A new family of authentication protocols. *Operating Systems Review*, 32(4):9–20, October 1998.

[2] M. Bellare, R. Canetti, and H. Krawczyk. HMAC: Keyed-hashing for message authentication. Internet Request for Comment RFC 2104, Internet Engineering Task Force, Feb. 1997.

[3] S. Bellovin. The icmp traceback message. `http://www.research.att.com/~smb`, 2000.

[4] F. Bergadano, D. Cavagnino, and B. Crispo. Chained stream authentication. In *Selected Areas in Cryptography 2000*, Waterloo, Canada, August 2000. A talk describing this scheme was given at IBM Watson in August 1998.

[5] F. Bergadano, D. Cavalino, and B. Crispo. Individual single source authentication on the mbone. In *ICME 2000*, Aug 2000. A talk containing this work was given at IBM Watson, August 1998.

[6] N. Bhaskar and I. Kouvelas. Source-specific protocol independent multicast. Internet Draft, Internet Engineering Task Force, Mar. 2000. Work in progress.

[7] M. Bishop. A Security Analysis of the NTP Protocol Version 2. In *Sixth Annual Computer Security Applications Conference*, November 1990.

[8] B. Briscoe. FLAMeS: Fast, Loss-Tolerant Authentication of Multicast Streams. Technical report, BT Research, 2000. `http://www.labs.bt.com/people/briscorj/papers.html`.

[9] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas. Multicast security: A taxonomy and some efficient constructions. In *Infocom '99*, 1999.

[10] R. Canetti, P. Rohatgi, and P.-C. Cheng. Multicast data security transformations: Requirements, considerations, and prominent choices. Internet draft, Internet Engineering Task Force, 2000. draft-data-transforms.txt.

[11] S. Cheung. An efficient message authentication scheme for link state routing. In *13th Annual Computer Security Applications Conference*, 1997.

[12] S. E. Deering. Host extensions for IP multicasting. Request for Comments (Standard) 1112, Internet Engineering Task Force, Aug. 1989.

[13] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei. Protocol independent multicast-sparse mode (PIM-SM): protocol specification. Request for Comments (Experimental) 2362, Internet Engineering Task Force, June 1998.

[14] R. Gennaro and P. Rohatgi. How to Sign Digital Streams. Technical report, IBM T.J.Watson Research Center, 1997.

[15] N. Haller. The S/KEY one-time password system. Request for Comments (Informational) 1760, Internet Engineering Task Force, Feb. 1995.

[16] M. Handley, H. Holbrook, and I. Kouvelas. Protocol independent multicast - sparse mode (pim-sm): Protocol specification (revised). Internet Draft, Internet Engineering Task Force, July 2000. Work in progress.

[17] U. S. Laboratory). Secure hash standard. Federal Information Processing Standards Publication FIPS PUB 180-1. `http://csrc.nist.gov/fips/fip180-1.txt` (ascii), `http://csrc.nist.gov/fips/fip180-1.ps` (postscript), Apr. 1995.

[18] L. Lamport. Password authentication with insecure communication. *Commun. ACM*, 24(11), Nov. 1981.

[19] M. Luby, J. Gemmell, L. Vicisano, L. Rizzo, J. Crowcroft, and B. Lueckenhoff. Asynchronous layered coding. a massively scalable reliable multicast protocol. Internet draft, Internet Engineering Task Force, July 2000. draft-ietf-rmt-pi-alc-01.txt.

[20] R. Merkle. Protocols for public key cryptosystems. In *1980 IEEE Symposium on Security and Privacy*, 1980.

[21] D. L. Mills. Network Time Protocol (Version 3) Specification, Implementation and Analysis. Internet Request for Comments, March 1992. RFC 1305.

[22] S.-S. Multicast. `http://www.ietf.org/html.charters/ssm-charter.html`.

[23] R. Perlman, C. Lee, T. Ballardie, J. Crowcroft, Z. Wang, T. Maufer, C. Diot, J. Thoo, and M. Green. Simple multicast: A design for simple, low-overhead multicast. Internet

Draft, Internet Engineering Task Force, Mar. 1999. Work in progress.

[24] A. Perrig, R. Canetti, B. Briscoe, J. Tygar, and D. X. Song. TESLA: Multicast Source Authentication Transform. Internet Draft, Internet Engineering Task Force, July 2000. Work in progress.

[25] A. Perrig, R. Canetti, J. Tygar, and D. X. Song. Efficient authentication and signing of multicast streams over lossy channels. In *IEEE Symposium on Security and Privacy*, May 2000.

[26] Reliable Multicast Transport (RMT). `http://www.ietf.org/html.charters/rmt-charter.html`.

[27] R. L. Rivest. The MD5 message-digest algorithm. Internet Request for Comments, Apr. 1992. RFC 1321.

[28] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[29] P. Rohatgi. A compact and fast hybrid signature scheme for multicast packet authentication. In *6th ACM Conference on Computer and Communications Security*, November 1999.

[30] Secure Multicast Group (SMuG). `http://www.ipmulticast.com/community/smug/`.

[31] D. X. Song and A. Perrig. Advanced and authenticatd marking schemes for ip traceback. Technical Report UCB/CSD-00-1107., UC Berkeley, July 2000.

[32] U. S. National Institute of Standards and Technology (NIST). Digital Signature Standard (DSS), Federal Register 56. FIPS PUB 186, Aug. 1991.

[33] C. K. Wong and S. S. Lam. Digital signatures for flows and multicasts. In *Proc. IEEE ICNP '98*, 1998.