Stateless connections

Tuomas Aura, Pekka Nikander
Helsinki University of Technology, FIN-02015 HUT, Finland
Tuomas.Aura@hut.fi, Pekka.Nikander@hut.fi

Abstract We describe a secure transformation of stateful connections or parts of them into stateless ones by attaching the state information to the messages. Secret-key cryptography is used for protection of integrity and confidentiality of the state data and the connections. The stateless protocols created in this way are more robust against denial of service resulting from high loads and resource exhausting attacks than their stateful counterparts. In particular, stateless authentication resists attacks that leave connections in a half-open state.

1 Introduction

In the open networks, malicious denial-of-service attacks and resource exhaustion by unexpectedly high demand for a service have become increasingly serious threats. In this paper we show how stateless protocols can be used to make systems more robust against denial-of-service. Our goal is to limit the number of potential attackers and to make recovery after an attack easier. This is done by saving the server state in the client rather than in the server.

The paper is structured as follows. We first discuss resource exhaustion problems typical of stateful services in Sec. 2. Sec. 3 shows how to securely make protocols stateless and compares their behavior to stateful ones under denial-of-service attacks. Sec. 4 describes partially stateless protocols. Sec. 5 contains some examples of real-world protocols and Sec. 6 concludes the paper.

The closest resemblants to our ideas in the literature are the HTTP cookie mechanism [7] and the stateless Sprite file server [9] that distributed the storage of state information in order to recover from server crashes. Perrochon [8] suggests stateless front-ends to stateful services. The cost of saving the connection state has been considered as one aspect affecting scalability of authentication protocols, for example, in [6]. Most literature concentrates on application-specific design techniques and reactive countermeasures.

2 Denial of service in stateful protocols

In Sec. 2.1 we explain how storing connection state makes stateful protocols vulnerable to resource exhaustion by overload. Sec. 2.2 discussed malicious attacks.

2.1 Running out of connection table space

In stateful protocols, there is always an upper limit on the number of clients that can connect to a server simultaneously, even if the clients might still be satisfied

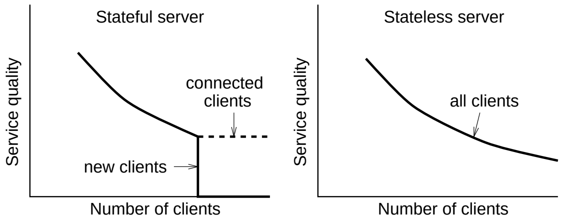

Figure1. Service quality as a function of load in stateless and stateful servers

with a thinner share of the server capacity. Eventually, the restriction is caused by the limited space that is available for storing connection state information. When more and more clients attempt to connect to the server, its storage space becomes exhausted and new connections must be refused. In the worst case, the connected clients do not consume the full capacity of the server, and the server remains partially idle while the refused clients are waiting to connect.

An unintentional mistake or a communication error may also leave a connection in an eternally open state. The client can forget to close the connection, lose its connection table data, or it may be unable to reach the server for the closing commands. In the end, the stale connections must be purged from the server table but it is difficult do this without sometimes closing valid connections.

One way to describe the problem with stateful servers is that their behavior under stress is unideal. Fig. 1 compares the quality of stateful and stateless service when the number of clients or connections to the server increases. The service quality in stateful server shows sharp fall at a certain threshold while in the stateless server it declines slowly.

2.2 Attacks exhausting connection limit

In a denial-of-service attack, an attacker can exhaust the connection limit of a server. If an open connection does not obligate the client to actively use the service, the malicious party needs to sacrifice only little of its capacity to continually block other clients' access to the service. Connection state maintenance becomes a critical resource, which it normally would not be. The attack is particularly disturbing because the attacker is not utilizing the service but merely consuming a secondary resource that is needed for the service access.

Of course, the attacker tries to keep the connections in a state where the number of simultaneous clients is as limited as possible, the effect of blocking maximally harmful, and its own efforts minimal. Furthermore, the attacker would prefer not to reveal its own identity to the server. Therefore, the danger of this type of attacks is usually greatest at the beginning of connections.

3 Making connections stateless

In Sec. 3.1, we show how to make protocols stateless by passing the state information between the protocol principals along the messages. Sec. 3.2 and 3.3 add integrity check to the state data and to the entire connection. Sec. 3.4 compares the behavior of stateful and stateless protocols under denial-of-service attacks.

3.1 Transformation from stateful into stateless

Assuming that the communication channels are reliable and flooding attacks the only concern, we can transform any stateful client/server protocol or communication protocol with initiator and responder into a stateless equivalent. This is done by sending state information from the server to the client with every message. Along the next message from the client, the state information is returned to the server. A stateful protocol and an equivalent stateless protocol:

1. $C \longrightarrow S : Msg_1$	S stores $State_{S1}$.
2. $S \longrightarrow C : Msg_2$	
3. $C \longrightarrow S : Msg_3$	S stores $State_{S2}$.
4. $S \longrightarrow C : Msg_4$	
$\quad\dots \qquad\quad \dots$	

1. $C \longrightarrow S : Msg_1$
2. $S \longrightarrow C : Msg_2, State_{S1}$
3. $C \longrightarrow S : Msg_3, State_{S1}$
4. $S \longrightarrow C : Msg_4, State_{S2}$
$\quad\dots \qquad\quad \dots$

Usually the server or the responding principal is the primary target of the denial-of-service threats and it is sufficient to make this principal stateless. In a symmetric protocol it is also possible to make both principals stateless by passing the states of both principals between them. Similar transformations are possible for multi-party protocols if the messages travel suitably. There one must take care that the state information is returned to the stateless principal in time.

The main reason for the stateless transformation is that it makes the system behavior more ideal. When there is no limit on the number of clients, the limit cannot be exploited by denial-of-service attacks. The ideal protocol properties also simplify quantitative analysis of system behavior under stress.

Moreover, the stateless protocol moves the responsibility of saving the state information from the server to the client. The client, who has requested the service, is better motivated to maintain the information and to recover from error conditions and data loss. The server does not have to reserve its resources for a single client for the indefinite time that may pass between protocol messages.

Another application for stateless protocols is information services that divide the server load between several identical machines. The servers can be geographically distributed or clustered in one place. When the servers are stateless, client requests can be routed to an arbitrary server without giving any consideration to where the previous messages were processed. Routing decisions and reply addresses can be changed dynamically in order to level the load on the servers and to minimize communication costs.

As a drawback, the stateless protocols require additional bandwidth for transferring the state data. If the states are large, the cost may be too high. File or document servers are therefore examples of promising applications for stateless protocols while intensely interactive sessions most often are not.

Although stateless protocols implemented in the above way resist denial-of-service attacks by server flooding, we still have to address other security issues. This is the topic of the next sections.

3.2 Integrity and confidentiality of the state data

When the state data is repeatedly transferred through insecure channels, its integrity and confidentiality become an important security concern. Since the state messages are sent and eventually received by the server itself, we can protect their integrity with message authentication codes that are relatively short and inexpensive to compute.

Also, the freshness of the state data should be checked in order to limit the number of times the data can be replayed. Timestamps can be applied liberally, because they are checked by their creator against the same clock that is used for the timestamping. Expired messages can be simply ignored. (The client is responsible for taking any corrective steps after such error conditions.) It is, however, necessary to allow long lifetimes for the states so that the data does not expire before the client wants to continue the message exchange and succeeds in sending its next request. Hence, the timestamp lifetime should be longer than the expected duration of a denial-of-service attack, usually on the order of several hours or a few days.

One consequence of timestamping is that distributed servers that accept state data packets created by each other must have synchronized clocks. Fortunately, the accuracy does not need to be very high if the timestamp lifetimes are long.

The improved transformation of a stateful service into a stateless one is illustrated by the protocol schema below. Every message leaving the server contains a timestamped state of the connection, authenticated with a key K_S^s known only by the server. The state is then returned to the server along the next message from the client.

$$\boxed{\begin{array}{l} 1.\ C \longrightarrow S : Msg_1 \\ 2.\ S \longrightarrow C : Msg_2,\ T_{S1}, State_{S1},\ MAC_{K_S^s}(T_{S1}, State_{S1}) \\ 3.\ C \longrightarrow S : Msg_3,\ T_{S1}, State_{S1},\ MAC_{K_S^s}(T_{S1}, State_{S1}) \\ 4.\ S \longrightarrow C : Msg_4,\ T_{S2}, State_{S2},\ MAC_{K_S^s}(T_{S2}, State_{S2}) \\ \quad \ldots \qquad\quad \ldots \end{array}}$$

There is often redundancy in the actual message and the state information. In that case, it is not necessary to repeat the redundant data. The MAC can be computed over all state information that is explicitly or implicitly returned to the client in the next step.

Another method for checking freshness is to change signature keys periodically. A few of the newest keys should be kept in the server's memory for accepting fresh messages. State information signed with older keys is then discarded as invalid. A key identifier should be added to the messages. The period of generating new signature keys becomes thus the resolution of message expiration times. The period of validity is the period of key generation multiplied by the number of newest keys accepted. Keys with different lifetimes can be used for different purposes. The computation required for maintaining the keys is independent of the number of clients or of the amount of traffic in the system.

Also, any secret state data is easily concealed by encrypting it with a secret key K_S^e known only by the server. The mechanism is illustrated below.

$$
\boxed{\begin{array}{rl}
i.\ C \longrightarrow S: & Msg_i,\ T_{S,i}, E_{K_S^e}(State_{S,i}),\ MAC_{K_S^s}(T_{S,i}, State_{S,i}). \\
i+1.\ S \longrightarrow C: & Msg_{i+1},\ T_{S,i}, E_{K_S^e}(State_{S,i}),\ MAC_{K_S^s}(T_{S,i}, State_{S,i})
\end{array}}
$$

3.3 Integrity and confidentiality of the connection

So far, the described stateless protocols have not addressed the integrity or confidentiality of the actual protocol messages in any way. In this section we will describe a technique for authenticating and encrypting stateless connections. We have an additional reason for the security enhancements. Namely, the statelessness opens a new line of attack against connection integrity: replay of connection states. The stateless principals have no means for detecting replays, because they cannot remember which messages have already been received and processed. An integrity check that links the state data to the actual messages will limit the ways in which third parties can utilize recorded server states in attacks.

In the following protocol schema, the client and the server have a shared secret key K_{CS} for signing and encrypting connection data. The server passes the key to the client along with all other state data.

$$
\boxed{\begin{array}{rl}
i.\ C \longrightarrow S: & Msg_i',\ T_{S,i}, E_{K_S^e}(K_{CS}), State_{S,i},\ MAC_i \\
i+1.\ S \longrightarrow C: & Msg_{i+1}',\ T_{S,i}, E_{K_S^e}(K_{CS}), State_{S,i},\ MAC_i
\end{array}}
$$

The message authentication code is $MAC_i = MAC_{K_S^s}(T_{S,i}, K_{CS}, State_{S,i})$ and the protected messages $Msg_k' = E_{K_{CS}}(Msg_k), MAC_{K_{CS}}(k, Msg_k, MAC_i)$ for $k = i, i+1$.

It is necessary to encrypt the messages only if their contents are secret. Authentication codes, on the other hand, should be used in in all systems where replay attacks are considered a threat, even on anonymous connections. Binding the state data together with the corresponding messages in this way effectively shields the system against third-party replay attacks that attempt to manipulate the logic of the protocol. (Replay flooding attacks will be discussed in Sec. 3.4.) Replays to the server still can result in multiple processing of requests and duplicate responses to the client. The protocol designer should ensure that the client is able to detect the duplicates or is not affected by them.

A dishonest client, on the other hand, cannot be stopped from replaying state data from earlier stages of the protocol run. By replaying old states, the client can return to any previous point in the protocol run. Consequently, the client can execute parts of the protocol several times, or go through several alternative branches of the protocol run. In some protocols, the possibility of collecting information from several alternative execution paths is catastrophic. For example, in many zero knowledge proofs, allowing two choices for the prover or verifier could result in a false proof or disclosure of secret knowledge, respectively. Also, stateless protocols cannot be used when accounting of service use is needed, for example, for billing the clients. Therefore, not all protocols can be securely made stateless. The stateless transformation is not suitable for protocols where the client is not entitled to the combined benefit from a small number of alternative protocol runs. Luckily, most communication protocols do not account resource usage and are deterministic enough so that the client will not gain any advantage by replaying the states.

3.4 Replays and denial of service

An attacker with access to the communication channel between the server and its clients might try to exhaust a stateless server by continuously sending replays of old messages. In this section, we compare the replay-flooding attack against stateless protocols to the connection-limit-exhaustion attack against stateful ones and show that the stateless protocol performs better.

We first consider the stateless server under a replay flooding attack. The best the attacker can do is to replay messages at the maximum throughput rate of the server so that no service capacity remains for honest clients. (This is the worst case scenario. In most communication systems, some legitimate messages will still get through.)

Another danger is that the legitimate connections start breaking because the time stamps on the state data expire while the attacker blocks the service. This can be avoided by making the timestamp lifetimes longer than it takes to detect the attack and to take countermeasures. After the attack, the clients can continue the connections without delay. This is the reason why the time stamps should last days rather than seconds or minutes. The timestamp lifetime, however, should not be infinite, because we want to limit the amount of replayable material in circulation.

Next, we consider the behavior of a stateful server when an attacker is creating new connections and leaving them open. If the attacker opens connections at the maximum rate C allowed by the server, no other clients can access the service. The stateful server must purge the idle connections from its memory after a certain time to make space for new ones. If the server has enough memory to save the state of M connections, each connection will remain in the memory at most time M/C. In a typical system, this time will be much shorter than the duration of an average attack. Thus, the attacker is able to break the existing connections.

Comparing the stateless and stateful protocols under their characteristic attacks, we observe that an equal rate of replays against the stateless server and connection openings against the stateful server have approximately the same effect on the service quality during the attack. Both servers can be clogged by the attacks. The stateless server, however, recovers automatically after an attack.

Another major advantage for the stateless server is that the described worst-case scenario is less likely to happen for it. We assumed that the attacker can record enough messages for the replay attacks. On a large network, most nodess never see any such messages. Hence, the number of parties that can mount the replay attack against the stateless server is very small in comparison to the group that can open false connections to the stateful server. For example, on the Internet, anyone in the world can open connections to almost any public server but very few hosts can record connections to a particular server.

We conclude that stateless protocols are, in general, more robust against denial-of-service attacks than their stateful counterparts. Stateless protocols make recovery after an attack easy and dramatically reduce the number of potential attackers.

4 Partially stateless protocols

Often the benefits of statelessness are biggest at certain specific parts of the protocol. Sections 4.1 and 4.2 discuss stateless connection opening and idle periods. In Sec. 4.3 we consider stateless layers in protocol stacks. Sec. 4.4 discusses optimizations based on state caching.

4.1 Stateless handshake

When abuse of the service is expected, the obvious solution is to authenticate the clients at the beginning of the connection. Attackers usually do not want to reveal their identity. Furthermore, authentication helps the server administration in resolving problems off-line. In this section, we show how statelessness improves robustness of the protocol at the beginning of the connection, before sufficient client authentication has taken place.

The level of authentication may vary from strong cryptographic identification to anonymous verification of access rights or electronic payment. In any case, the first few steps of the protocol before the client has been authenticated are vulnerable to the same kinds of denial-of-service attacks as the completely unauthenticated connections. The attacker may start the authentication procedure and then leave the server waiting at an intermediate state. Therefore, we suggest that authentication protocols should always remain stateless until the client authentication is complete or the client has in some other way clearly shown its commitment to honest use of the service. After the authentication, the server can change to stateful mode. Especially in pay-per-use services this is most natural because the server must save accounting information.

We now demonstrate the importance of authenticating the client before the server becomes stateful. In the three-way X.509 authentication protocol [5], an attacker can replay a large number of old copies of the first message. This could exhaust the space that the principal B has reserved for saving the state of the protocol between sending Message 2 and receiving Message 3. (Note that the three-way X.509 protocol uses nonces for verifying freshness of the messages. Hence, principal B only knows that Message 1 is fresh after receiving Message 3.)

$$
\boxed{
\begin{array}{l}
1.\ A \longrightarrow B : S_A(N_A, B, E_B(K_{AB})) \\
2.\ B \longrightarrow A : S_B(N_B, A, N_A, E_A(K_{BA})), \\
3.\ A \longrightarrow B : S_A(N_B, B)
\end{array}
}
$$

In the following modification of the X.509 protocol, the responding principal B does not need to save the state of the protocol until it has positively authenticated A.

$$
\boxed{
\begin{array}{l}
1.\ A \longrightarrow B : S_A(N_A, B, E_B(K_{AB})) \\
2.\ B \longrightarrow A : S_B(N_B, A, N_A, E_A(K_{BA})), \\
\qquad\qquad\quad T_B, E_{K_B^e}(K_{AB}, K_{BA}),\ MAC_{K_B^s}(T_B, N_B, A, K_{AB}, K_{BA}) \\
3.\ A \longrightarrow B : S_A(N_B, B), T_B, E_{K_B^e}(K_{AB}, K_{BA}), MAC_{K_B^s}(T_B, N_B, A, K_{AB}, K_{BA})
\end{array}
}
$$

The above protocol is deterministic in the sense that there is only one execution path. Its runs differ only in that fresh nonces and keys are generated every time. Therefore, an attacker could not possibly collect any interesting information by replaying old states and thus causing the principals to re-execute steps. Luckily, most cryptographic protocols have a similar deterministic nature. The biggest benefit attackers can sometimes gain from repeating protocol runs or steps is a little more material for cryptanalysis, but this should not endanger the security of strong cryptographic algorithms.

Most key exchange protocols aim at producing unique keys for every session and purpose. For the stateless principals, however, there is no difference between one and many sessions with the same parameters. Furthermore, the branching of the key exchange process can lead to the generation of several alternative end results. Thus, one-to-one correspondence between session and keys may be partially lost in the stateless protocols.

In anonymous services that are free to everyone, the return address of the client may be the only available identifier, but it is often enough. If the client responds to a message from the server, the server at least knows how to reach the client, which can be construed as a level of authentication. In the following protocol schema, the server is stateless only during the first roundtrip from it to the client and back. This is sufficient to prevents attacks like the SYN-flooding against the TCP protocol [2].

1. $C \longrightarrow S$:	$Msg1$
2. $S \longrightarrow C$:	$Msg2,\ T_S, State_S,\ MAC_{K_S^s}(T_S, State_S)$
3. $C \longrightarrow S$:	$Msg3,\ T_S, State_S,\ MAC_{K_S^s}(T_S, State_S)$
	Return address valid. S moves to stateful mode.
4. $S \longrightarrow C$:	$Msg4$
...	...

4.2 Statelessness during idle periods

In long-lived protocol runs, activity often ceases and is resumed later. In a stateful protocol, the server will have to maintain connection state data such as session keys throughout the idle periods. The server can be relieved of this duty by sending the state information to the client for temporary storage. The client has the responsibility for saving the data and recovering from data losses.

Depending on the protocol, the server can send its state to the client with every message, after certain messages, or after the connection has been idle for a threshold time. The client can either automatically return the state in its next message or check first whether the server still has the state in its cache.

The session parameters are usually known to both principals. Hence, it is not necessary to send everything to the client. Often only the session key needs to be encrypted with the server's secret key and sent over the channel along with a message authentication code for the key and the rest of the session parameters. The client knows the parameters and can return them with the message authentication code in its next message. If necessary, the client can encrypt confidential session parameters with the session key.

$$\boxed{\begin{array}{l} S \longrightarrow C : Msg1, T_S, E_{K_S^e}(K_{SES}),\ \ MAC_{K_S^s}(T_S, C, \text{sespar}, K_{SES}) \\ \qquad\quad \text{Possibly long idle period follows.} \\ C \longrightarrow S : Msg2, T_S, C, \text{sespar}, E_{K_S^e}(K_{SES}), MAC_{K_S^s}(T_S, C, \text{sespar}, K_{SES}) \end{array}}$$

When the session keys are transfered in this way, the server's key encryption keys must be treated with as much care as any master key and preferably changed periodically.

4.3 Stateless layers in protocol stacks

Communication protocols are organized in stacks where each layer uses the services of the layer below and provides services to the layer above. The stateful and stateless protocols should be viewed in this context. It seems that statelessness offers most benefits at the transport and application layers of the protocol stack. For example, in the TCP/IP protocol stack, denial-of-service attacks are usually targeted either at the TCP protocol [2] or at the application layer. In UDP based protocols, the application layer protocols are the natural target [3]. When an application layer protocol on the top of the stack is stateless, it usually makes sense to have all layers down to the network layer equally stateless.

A protocol can have more than one upper layer protocol accessing it. Then, regardless of whether the upper protocols are stateless or not, it is beneficial to make the lower layers stateless. The reason is that the alternative upper layer protocols should be able to operate independently of the resources consumed by each other and attacks against each other. To accomplish this, it is not always necessary to send the state data to the other end of the connection. The lower layer could also pass the information to the upper layers for storage.

4.4 State caching

The main disadvantage of fully stateless protocols is the hit on protocol performance. Fortunately, most performance benefits of stateful servers can be retained in stateless protocols by caching state data in the server.

The stateless server can cache state information as long as its memory capacity is not exhausted. Under normal server load, a caching server can behave just as a stateful server except for the cost of transmitting the state data. It can, for example, detect lost and duplicated messages, report the errors to the client, collect statistics on the channel characteristics, and dynamically adjust the transfer rate and packet size for optimum performance. Windowing techniques can be applied to avoid waiting for acknowledgements from the other party after each request. If server memory becomes scarce or the maintenance of the state data too burdensome, the server can immediately purge the data from its memory. In this way, the system can have nearly the performance of stateful servers while being resistant to denial-of-service. Although it may be difficult to design state caching servers with optimal performance, it should be feasible to find reasonable compromises. State caching has been succesfully applied in the stateless Sprite file server [9].

5 Application examples

In Sec. 5.1 we improve both robustness and performance of a key exchange protocol from the ISAKMP specification. Sec. 5.2 describes how the statefulness of the TCP protocol leads to the SYN flooding attack.

5.1 Stateless ISAKMP/Oakley

We show how to make stateless a version of the Oakley key exchange [4] used in the Internet Security Association and Key Management Protocol (ISAKMP).

The initiator A and the responder B start by exchanging nonces in order to to ensure that the initiator has given a correct reply address. The protocol uses public key signatures and Diffie-Hellman key exchange with reusable secret parameters and nonces for freshness.

$$\boxed{\begin{array}{l}
1.\ A \longrightarrow B : N_A \\
2.\ B \longrightarrow A : N_B, N_A \\
3.\ A \longrightarrow B : N_A, N_B, S_{K_A}(g^x, A, B, N'_A) \\
4.\ B \longrightarrow A : N_B, N_A, S_{K_B}(g^y, B, A, N'_B, N'_A) \\
5.\ A \longrightarrow B : N_A, N_B, S_{K_A}(g^x, A, B, N'_A, N'_B)
\end{array}}$$

The responder knows the initiator's key generation parameter to be fresh only after receiving Message 5. This leaves the protocol vulnerable to attacks where someone initiates a connection, executes Steps 1, 2 and 3 of the protocol, and then leaves the responder waiting forever.

We enhance the protocol by making the responder stateless until the receipt of the last message. The initial cookie exchange may be unnecessary, because the stateless responder is not as greatly affected by opening messages with forged return address. Thus, the protocol is more robust and has less messages than the original one.

$$\boxed{\begin{array}{l}
1.\ A \longrightarrow B : N_A, g^x, A, B, N'_A \\
2.\ B \longrightarrow A : N_B, N_A, S_{K_B}(g^y, g^x, B, A, N'_B, N'_A), \\
\qquad\qquad\quad MAC_{K_B^s}(T_B, g^y, g^x, N_B, N_A, N'_B, N'_A),\ E_{K_B^e}(y) \\
3.\ A \longrightarrow B : N_A, N_B, S_{K_A}(g^x, g^y, A, B, N'_A, N'_B), \\
\qquad\qquad\quad MAC_{K_B^s}(T_B, g^y, g^x, N_B, N_A, N'_B, N'_A),\ E_{K_B^e}(y)
\end{array}}$$

5.2 TCP resistance to SYN flooding

Recently, attention has been paid to the so called SYN flooding attack against the TCP/IP Transmission Control Protocol (TCP). In the TCP connection establishment, the parties exchange message sequence numbers in the so called SYN and SYN ACK messages. After receiving the first message, the responder creates a state called Transmission Control Block (TCB). In the SYN flooding attack, the attacker fills up the responder's TCB table by sending SYN messages to the server with forged IP return adresses.

Several people have independently suggested versions of this protocol where the server does not create a state initially. Instead, the responder can compute a message authentication code of the initiator sequence number and other session parameters. It sends the MAC to the client and receives it again in the next message.

The original and enhanced protocols are shown below. The MAC plays the dual role of functioning as ISN_S and the MAC for the server's state. ($MAC = MAC_{K_S^s}(ISN_C, parameters)$)

1. $C \longrightarrow S : ISN_C, SYN$	1. $C \longrightarrow S : ISN_C, SYN$
2. $S \longrightarrow C : ISN_S, ISN_C + 1, SYN\|ACK$	2. $S \longrightarrow C : MAC, ISN_C + 1, SYN\|ACK$
3. $C \longrightarrow S : ISN_C + 1, ISN_S + 1, ACK$	3. $C \longrightarrow S : ISN_C + 1, MAC + 1, ACK$

After these changes, the server avoids creating the TCB before it knows the initiator's IP address to be valid. Early experiments by the present authors indicate that the TCP protocol could be futher enhanced by making the responder completely stateless.

6 Conclusion

We described weaknesses in stateful protocols and showed how they they can be avoided by making the connections stateless. The stateless protocols behave more ideally under overload or denial-of-service attacks than their stateful counterparts, they recover faster from the attacks, and they effectively limit the number of potential attackers. In particular, stateless protocols resist attacks that leave connections in a half-open state.

References

1. Tuomas Aura and Pekka Nikander. Stateless connections. Technical Report A46, Helsinki University of Technology, Digital Systems laboratory, May 1997.
2. TCP SYN flooding and IP spoofing attack. CERT Advisory CA-96.21, CERT, November 1996.
3. UDP port denial-of-service attack. CERT Advisory CA-96.01, CERT, August 1996.
4. D. Harkins and D. Carrel. The resolution of ISAKMP with Oakley. Internet draft, IETF IPSEC Working Group, June 1996.
5. Recommendation x.509 (11/93) - the directory: Authentication framework. ITU, November 1993.
6. P. Janson, G. Tsudik, and M. Yung. Scalability and flexibility in authentication services: The KryptoKnight approach. In *IEEE INFOCOM'97*, Tokyo, April 1997.
7. David M. Kristol and Lou Montulli. HTTP state management mechanism. Internet draft, IETF HTTP Working group, July 1996.
8. Louis Perrochon. *Gateways in globalen Informationssystemen.* PhD thesis, ETH Zürich, 1996. Diss. ETH Nr. 11708.
9. Brent Welch, Mary Baker, Fred Douglis, John Hartman, Mendel Rosenblum, and John Ousterhout. Sprite position statement: Use distributed state for failure recovery. In *Proc. 2nd Workshop on Workstation Operating Systems WWOS-II*, pages 130–133, September 1989.