Now that you are interested..

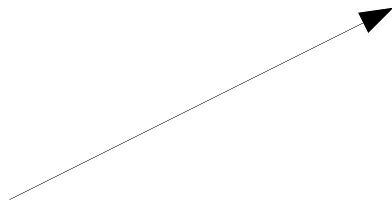
- Lets look at the socket API and SCTP
- In this short time we have I will try to highlight the basic outline of SCTP and socket's
- Trade off the two models
- And give you an idea on how to use streams
- I can't possibly tell you about all the knobs that you can tweak (socket options).
- But I will try to help you understand notifications.
- I will assume you know something about a socket :-D

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Sockets Basics

- **The socket call looks as follows:**

```
sd = socket(int domain, int type, int protocol);
```



int domain – represents what Internet domain you wish to create the socket in. AF_INET builds you a IPv4 socket. AF_INET6 builds you a IPv6 socket. Note that for AF_INET6, you may receive both IPv4 and IPv6 address types from the socket.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Sockets Basics

- **The socket call looks as follows:**

sd = socket(int domain, int type, int protocol);

int type – Will take on one of a number of values:

- SOCK_STREAM – used for TCP, but also used for SCTP to indicate a 1 to 1 model socket.
- SOCK_DGRAM – used for UDP, but also *possibly* used for SCTP's 1 to many model with atomic delivery.
- SOCK_SEQPACKET – used for SCTP's 1 to many model without atomic delivery.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Sockets Basics

- **The socket call looks as follows:**

```
sd = socket(int domain, int type, int protocol);
```

int protocol – Takes on the protocol type IPPROTO_UDP, IPPROTO_TCP and IPPROTO_SCTP are commonly used values.

Protocol	May be used with
IPPROTO_TCP	SOCK_STREAM
IPPROTO_UDP	SOCK_DGRAM
IPPROTO_SCTP	SOCK_STREAM, SOCK_SEQPACKET

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Socket API calls

- So after you have created a socket, what can you do with it? The following are some of the function calls you will be using shortly :-D

    recvmsg(), recvfrom(), sendmsg(), sendto()

    bind(), connect(), accept(), listen()

    setsockopt(), getsockopt()

  For SCTP:

    sctp_sendmsg(), sctp_send(), sctp_sendx(), sctp_recvmsg(), sctp_connectx(), sctp_bindx(), sctp_getpaddrs(), sctp_getladdrs(), sctp_freepaddrs(), sctp_freeladdrs()

© 2003-2006 Cisco Systems, Inc. All rights reserved.

SCTP Socket Types

- SCTP socket API comes in two forms: one-to-one (SOCK_STREAM) and one-to-many (SOCK_SEQPACKET).
- The one-to-one interface provides a “compatibility” mode that allows virtually seamless porting of TCP applications to SCTP.

  We have used this interface to transparently port many TCP applications to SCTP (such as Apache, Mozilla, ssh).
- The one-to-many interface provides a mechanism to give a UDP-like mechanism to an application (SOCK_SEQPACKET).

© 2003-2006 Cisco Systems, Inc. All rights reserved.

So what is the one-to-one model?

- The one-to-one model was developed to be compatible with TCP. When using this model a server will typically do the following:
  - listen()
  - accept()
- Accept returns a new “socket descriptor” just as if you had made a socket() call. The new socket will have the connection to the remote client.
- A client will typically do a
  - connect()
- And then send data after the blocking connect completes.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

One-2-One Model illustration

EP-A

EP-Z

s = socket(...)

listen(s, bl)

s1 = accept(s,..)

blocked

© 2003-2006 Cisco Systems, Inc. All rights reserved.

One-2-One Model illustration

EP-A

s = socket(...)

listen(s, bl)

s1 = accept(s,..)

blocked

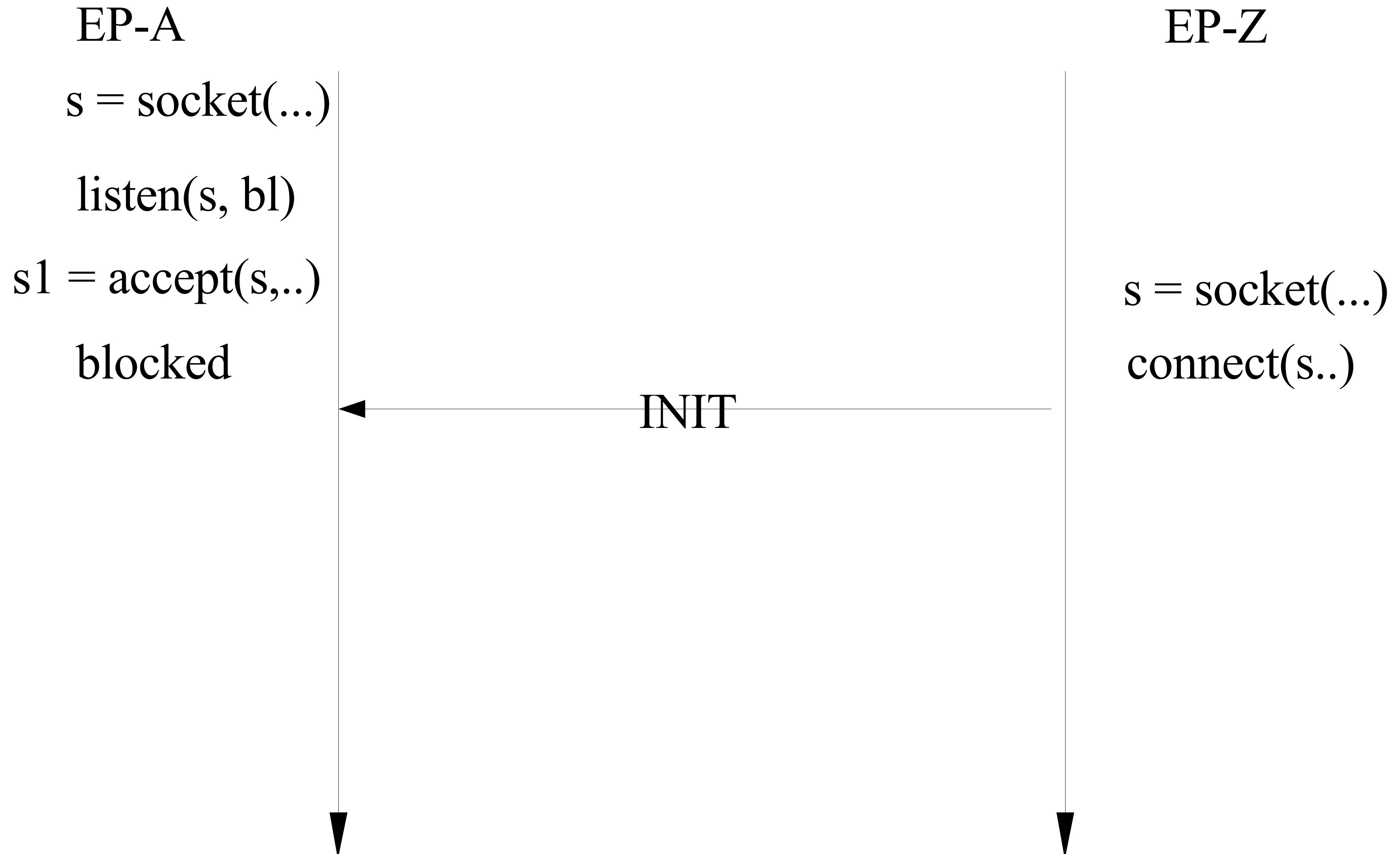
EP-Z

s = socket(...)

connect(s..)

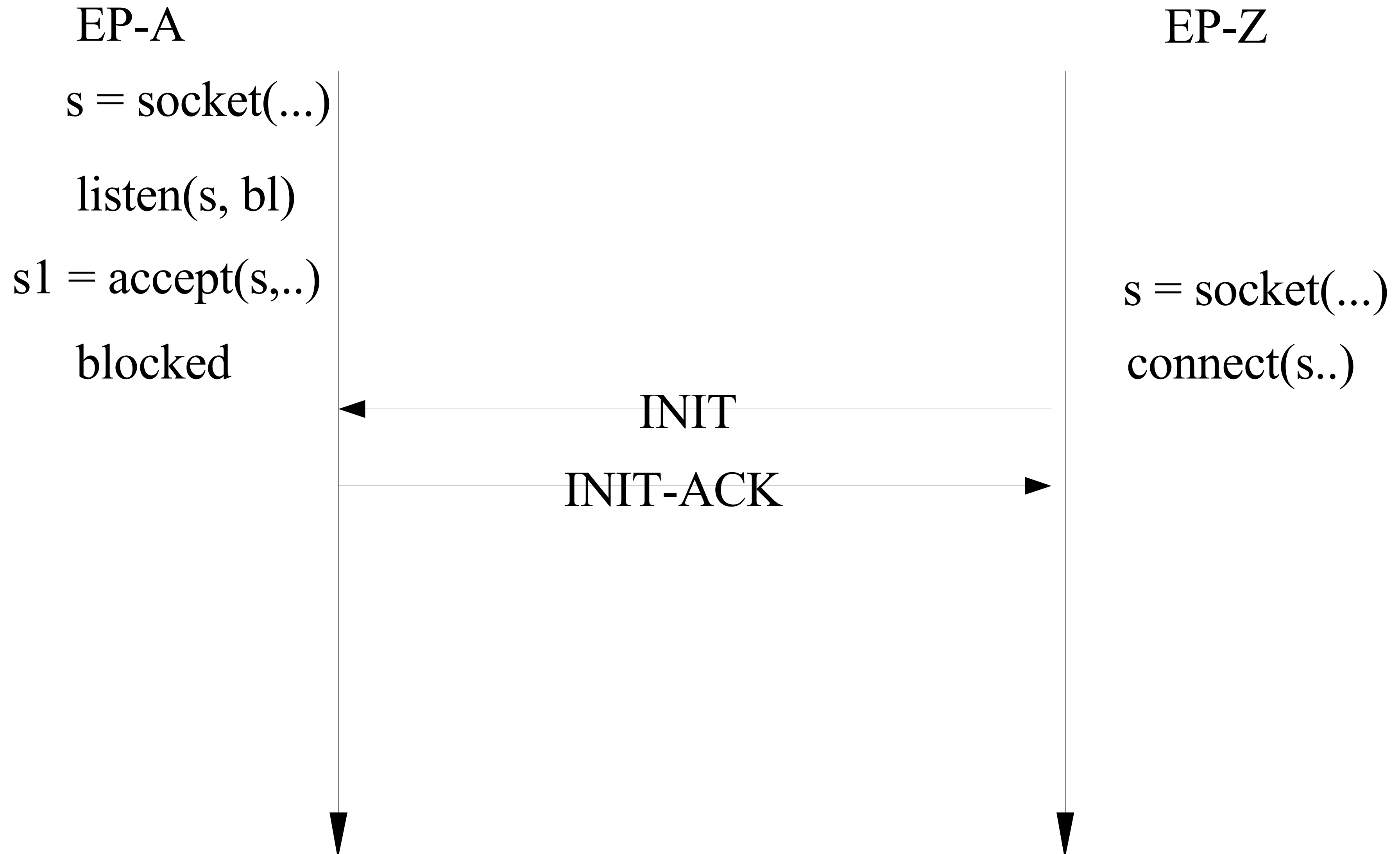
© 2003-2006 Cisco Systems, Inc. All rights reserved.

One-2-One Model illustration



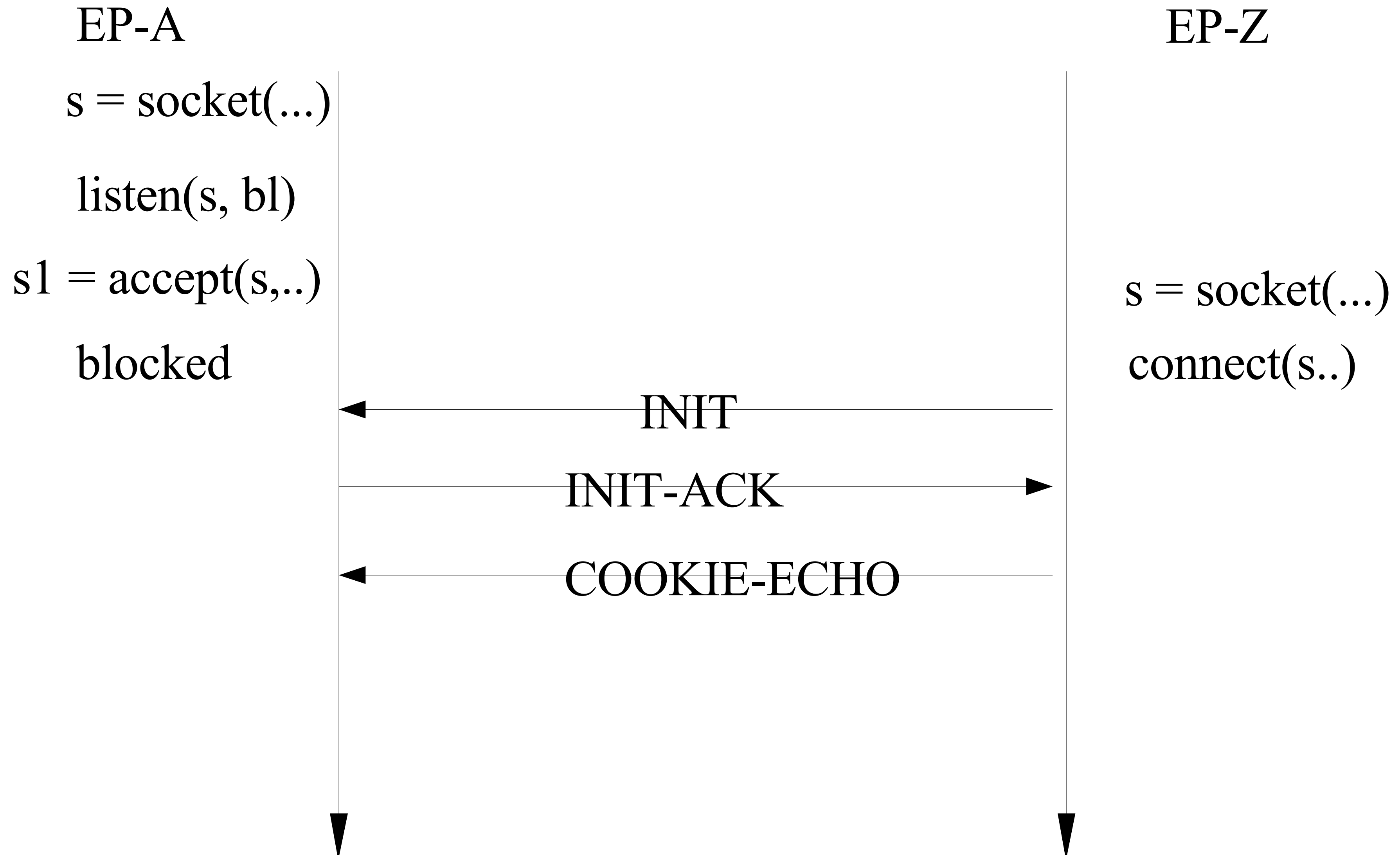
© 2003-2006 Cisco Systems, Inc. All rights reserved.

One-2-One Model illustration



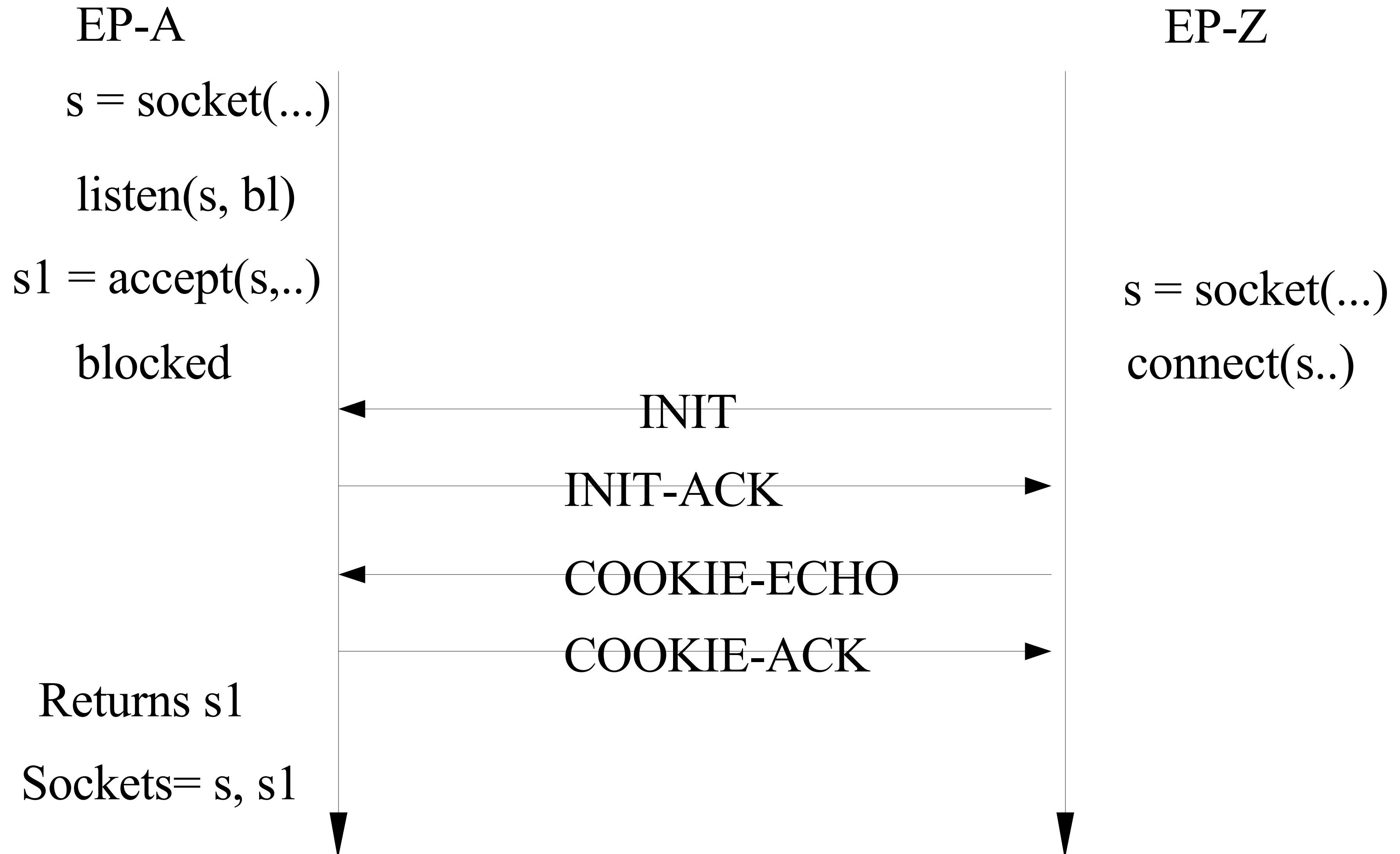
© 2003-2006 Cisco Systems, Inc. All rights reserved.

One-2-One Model illustration



© 2003-2006 Cisco Systems, Inc. All rights reserved.

One-2-One Model illustration



© 2003-2006 Cisco Systems, Inc. All rights reserved.

One-2-One Model illustration

EP-A

EP-Z'

s2 = accept(s,..)

blocked

Sockets= s, s1

© 2003-2006 Cisco Systems, Inc. All rights reserved.

One-2-One Model illustration

EP-A

EP-Z'

s2 = accept(s,..)

blocked

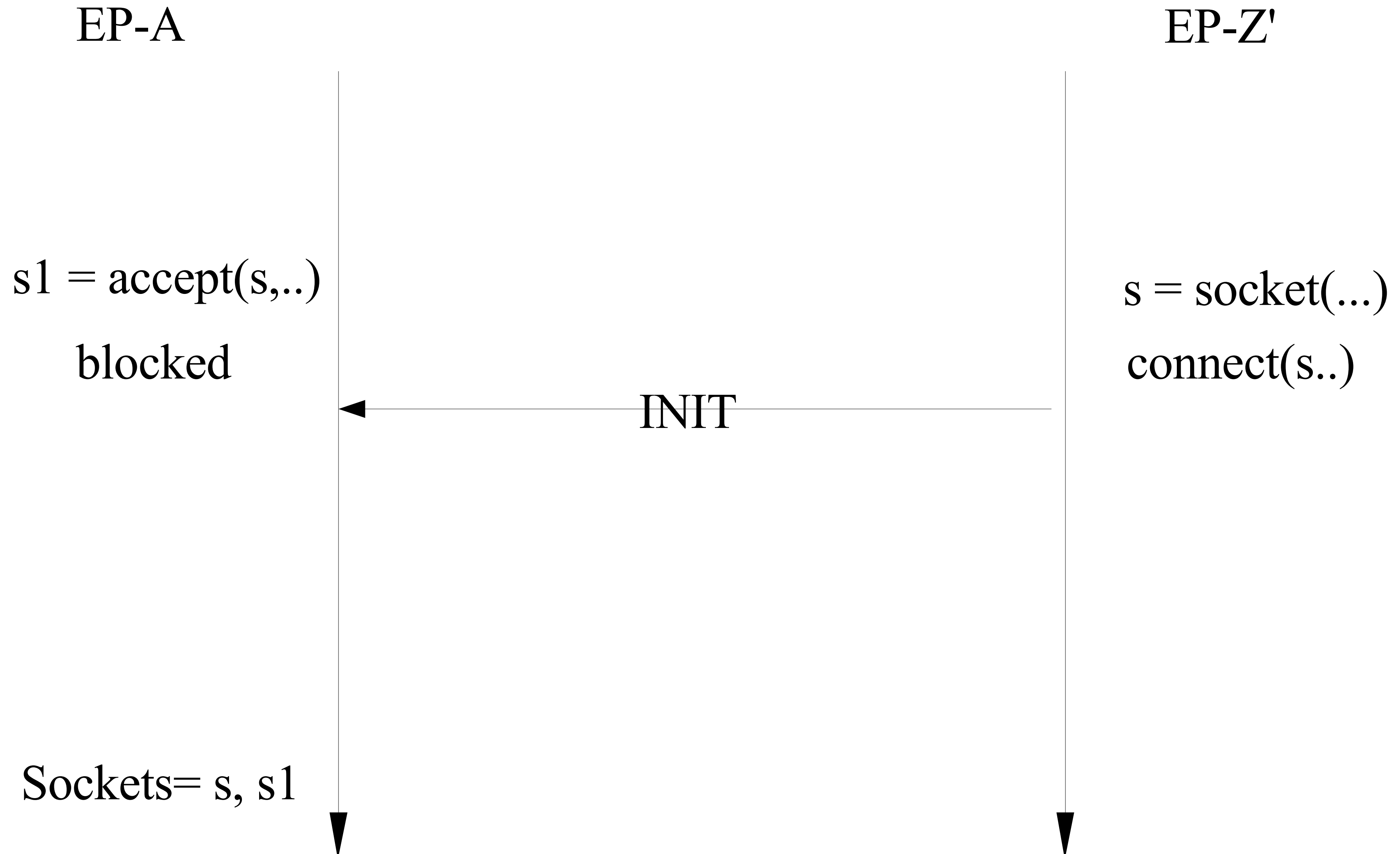
s = socket(...)

connect(s..)

Sockets= s, s1

© 2003-2006 Cisco Systems, Inc. All rights reserved.

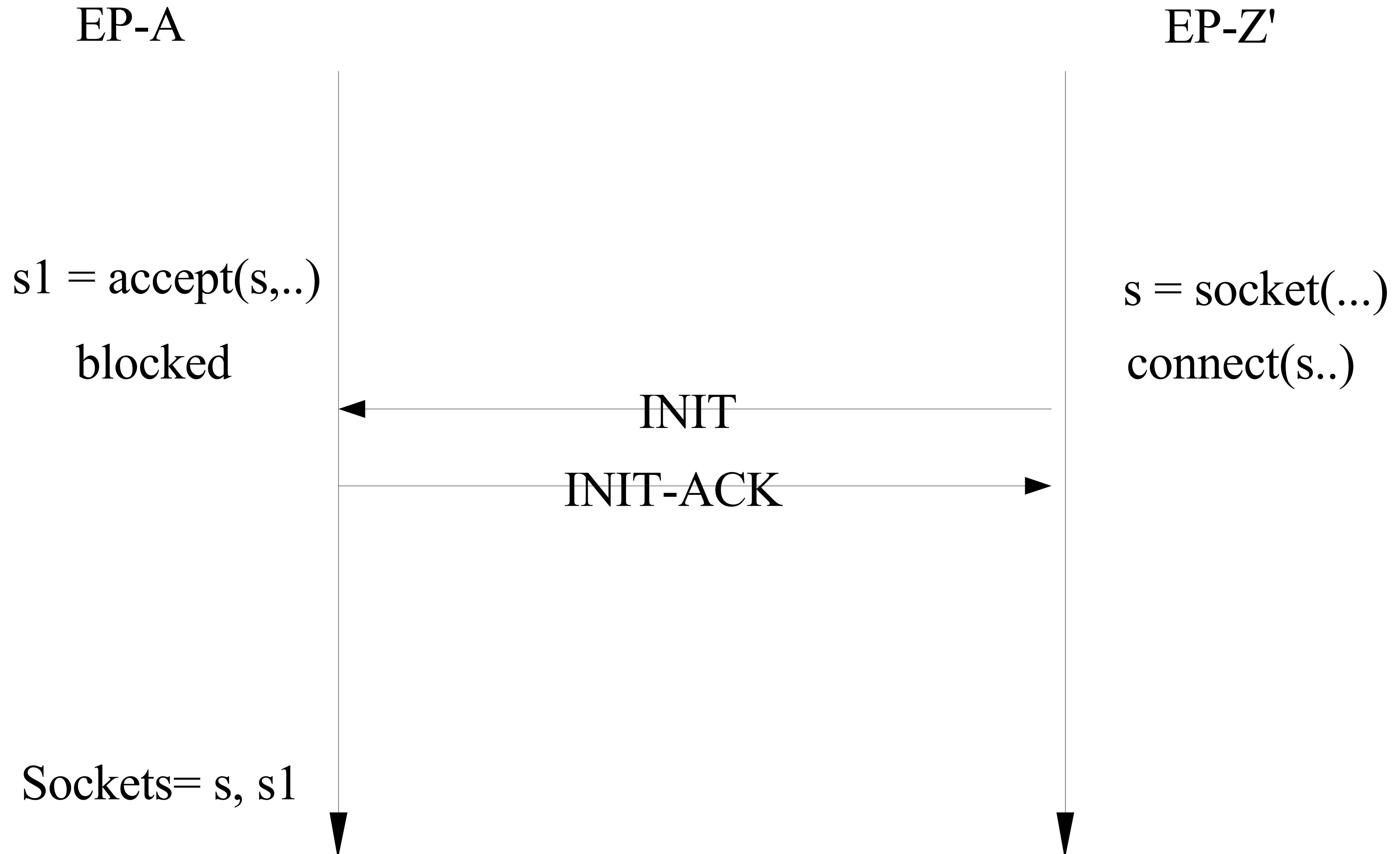
One-2-One Model illustration



© 2003-2006 Cisco Systems, Inc. All rights reserved.

One-2-One Model illustration

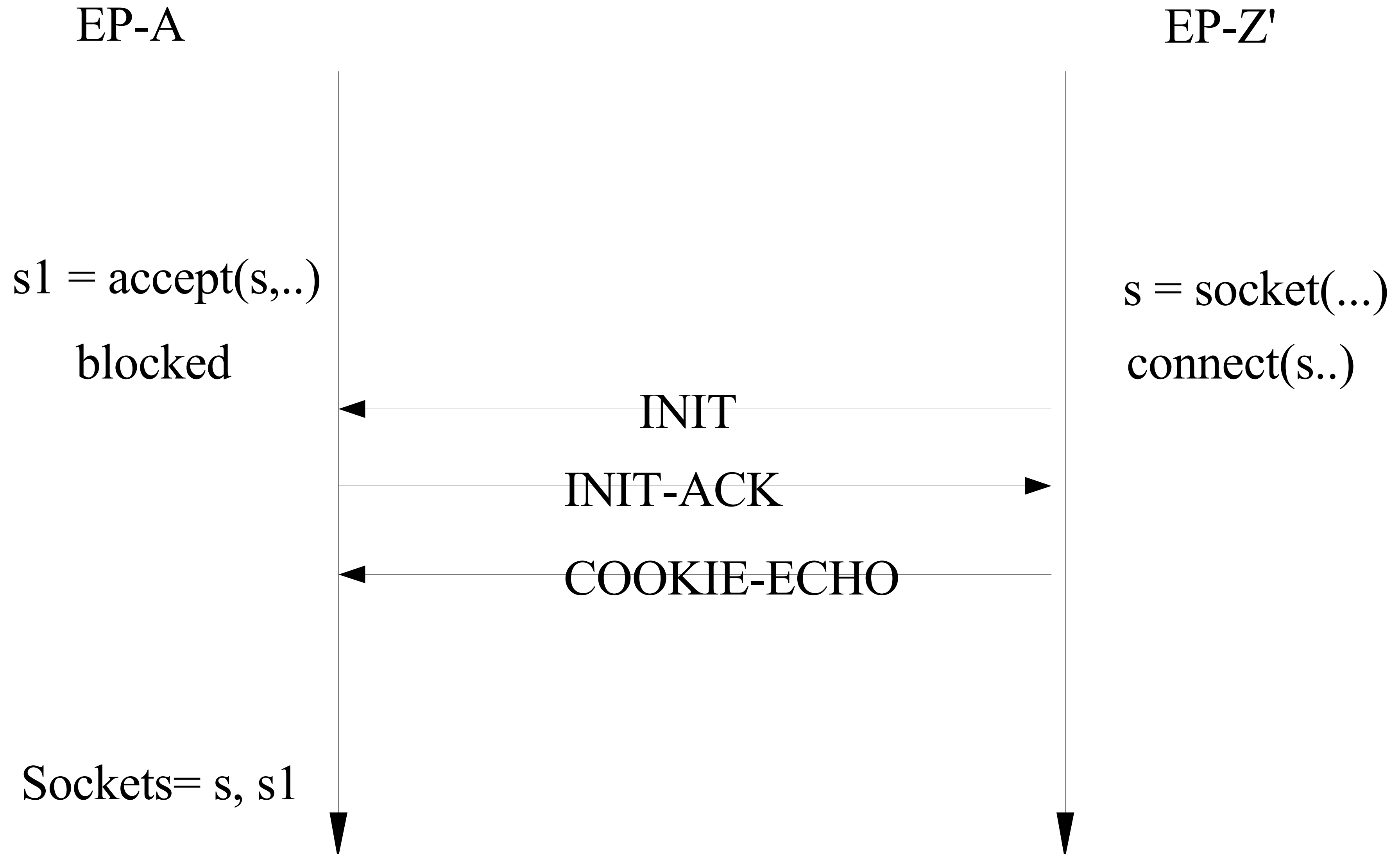
Cisco.com



© 2003-2006 Cisco Systems, Inc. All rights reserved.

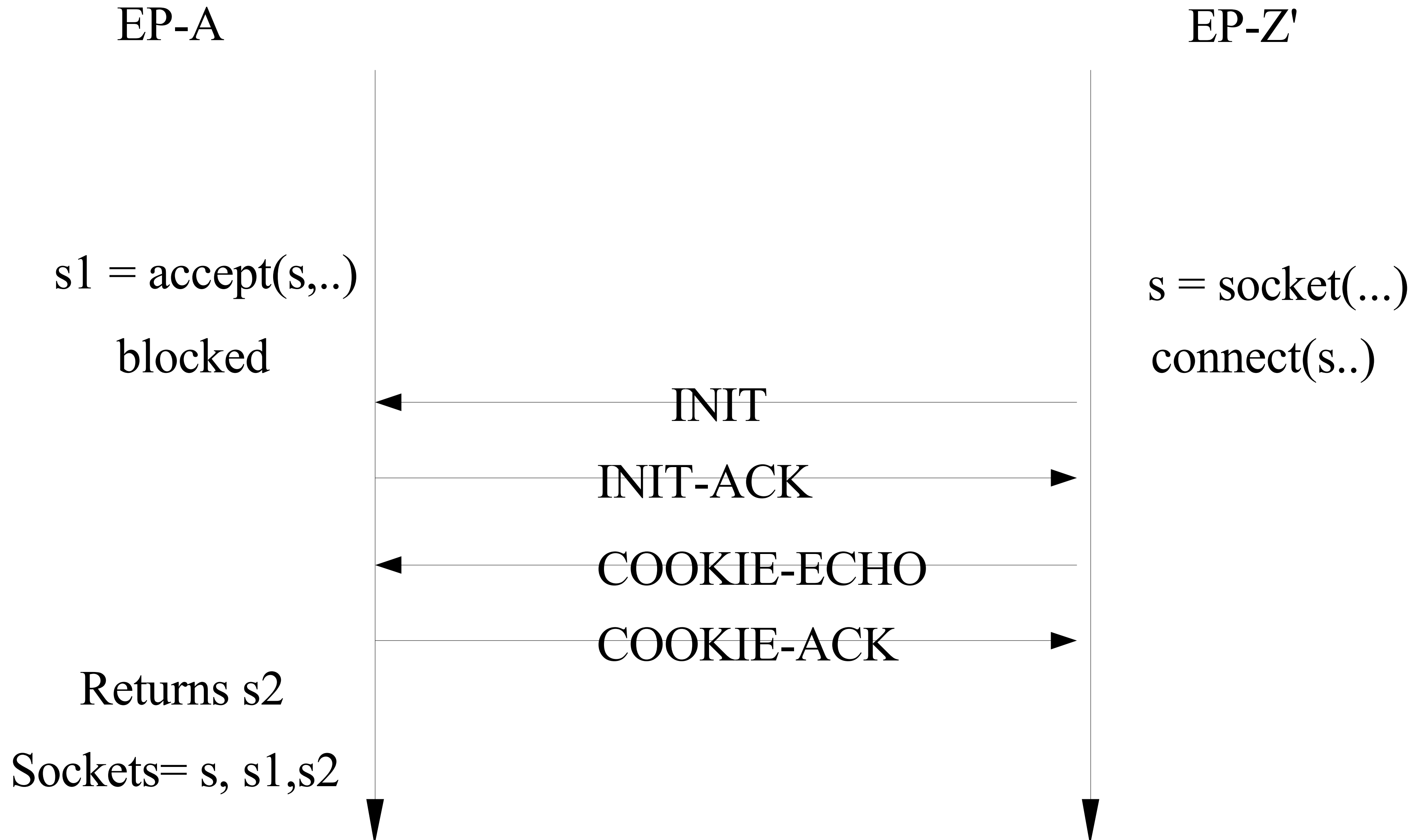
One-2-One Model illustration

Cisco.com



© 2003-2006 Cisco Systems, Inc. All rights reserved.

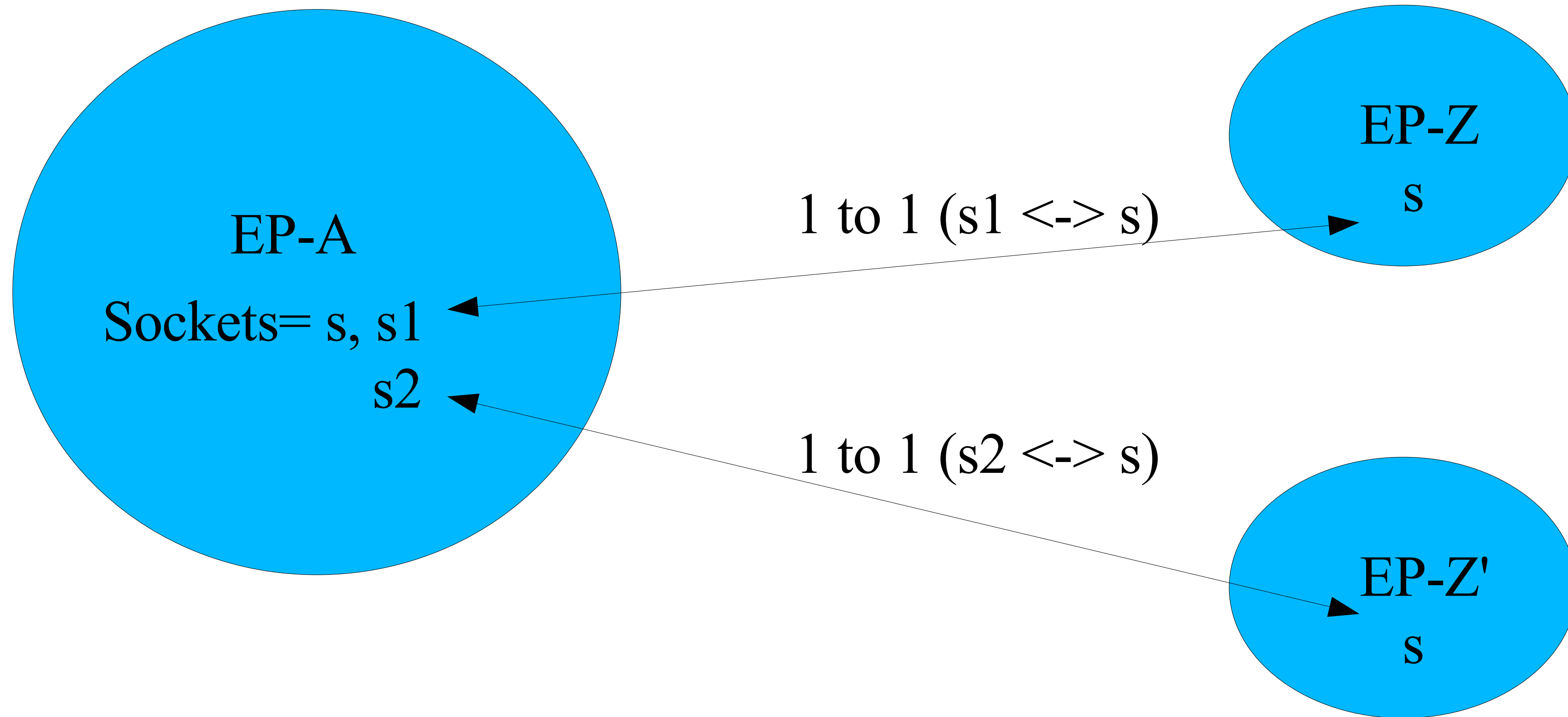
One-2-One Model illustration



© 2003-2006 Cisco Systems, Inc. All rights reserved.

One-2-One Model illustration

Cisco.com



© 2003-2006 Cisco Systems, Inc. All rights reserved.

One-to-one Example Server

Cisco.com

```
int sd, newfd, sosz;
struct sockaddr_in6 sin6;
sosz = sizeof(sin6);
sd = socket(AF_INET6, SOCK_STREAM, IPPROTO_SCTP);
listen(sd, 1);
while (1) {
    newfd = accept(sd, (struct sockaddr *)&sin6, &sosz)
    do_child_stuff(newfd, &sin6, sosz);
}
```


© 2003-2006 Cisco Systems, Inc. All rights reserved.

So what is the one-to-many model?

- The one-to-many model was developed especially for peer-to-peer applications. When using this model if you wish to receive new connections you must call listen().
- Whether you do listen() or not, you can start sending data to any address (just like you would with a UDP socket).
  - If an association does not exist for that address, one will be implicitly created.
  - Note that the data to send is queued until the association is established.
- Note that you can do a connect() to “pre-build” a association, but this is not required.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

So what is the one-to-many model?

- In the one-to-many model, a socket can have multiple associations under it. So this means that any receive call will return the next data from amongst all of the associations.
- This may also cause interesting complications with select on write. Since it is almost always true that one of the associations will be writable, the call is useless. If you want to write on a specific association and need to do a select on write, you need to use sctp_peeloff()

© 2003-2006 Cisco Systems, Inc. All rights reserved.

So what is the one-to-many model?

- Advantages of the one-to-many model:
  - Sending data that implicitly starts an association is the only way to get data sent with the cookie-echo.
  - An application does NOT need to track any connection state, basically all it needs to do is send and receive.
- If an application does wish to track connection state, using the association-id in the communication up/down notifications is the preferred method.
- If an application does NOT wish to track associations, using the AUTO_CLOSE socket option is advisable (covered later under socket options).

© 2003-2006 Cisco Systems, Inc. All rights reserved.

One-2-Many Model illustration

EP-A

EP-Z

s = socket(...)

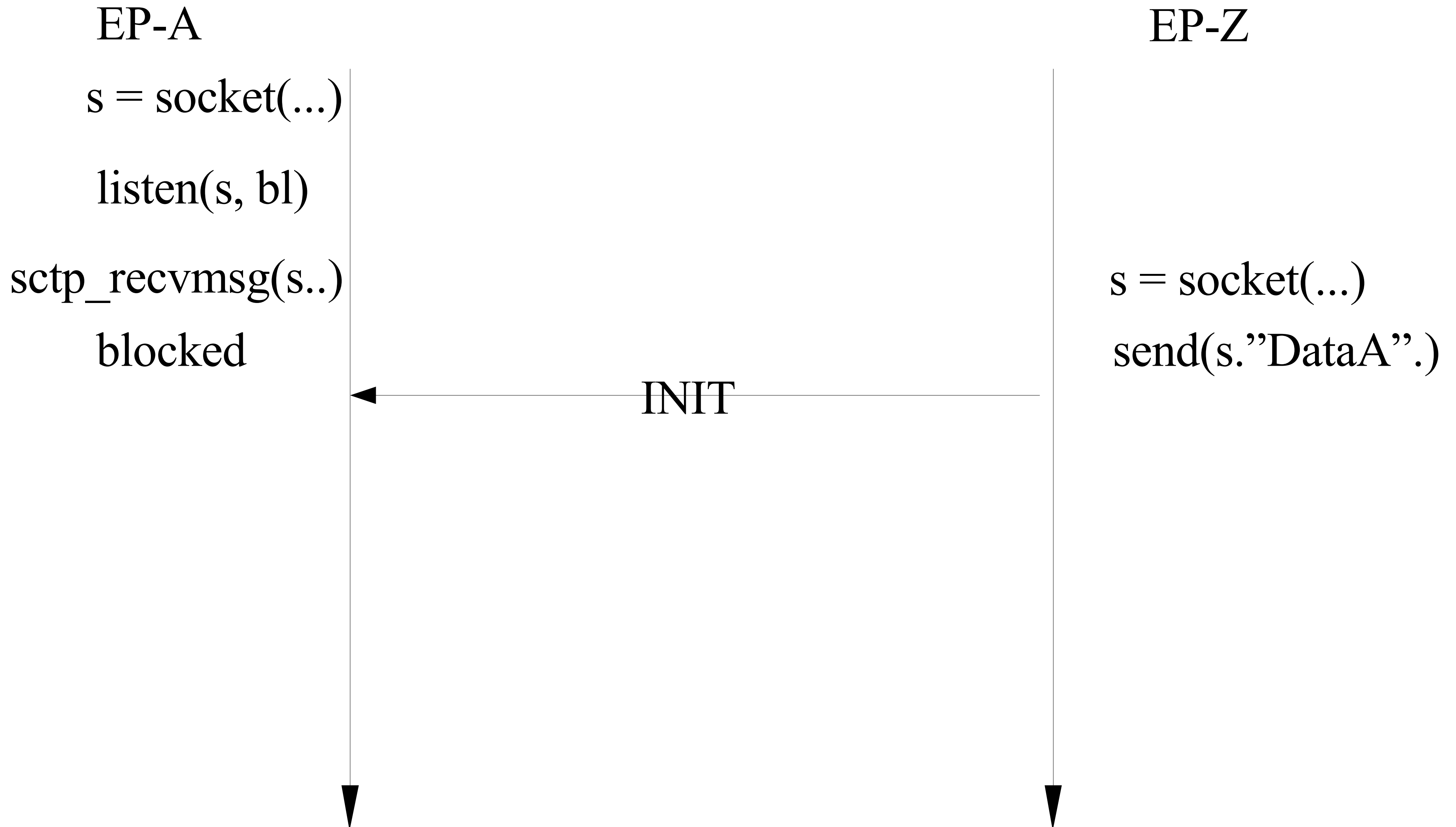
listen(s, bl)

sctp_recvmsg(s..)

blocked

© 2003-2006 Cisco Systems, Inc. All rights reserved.

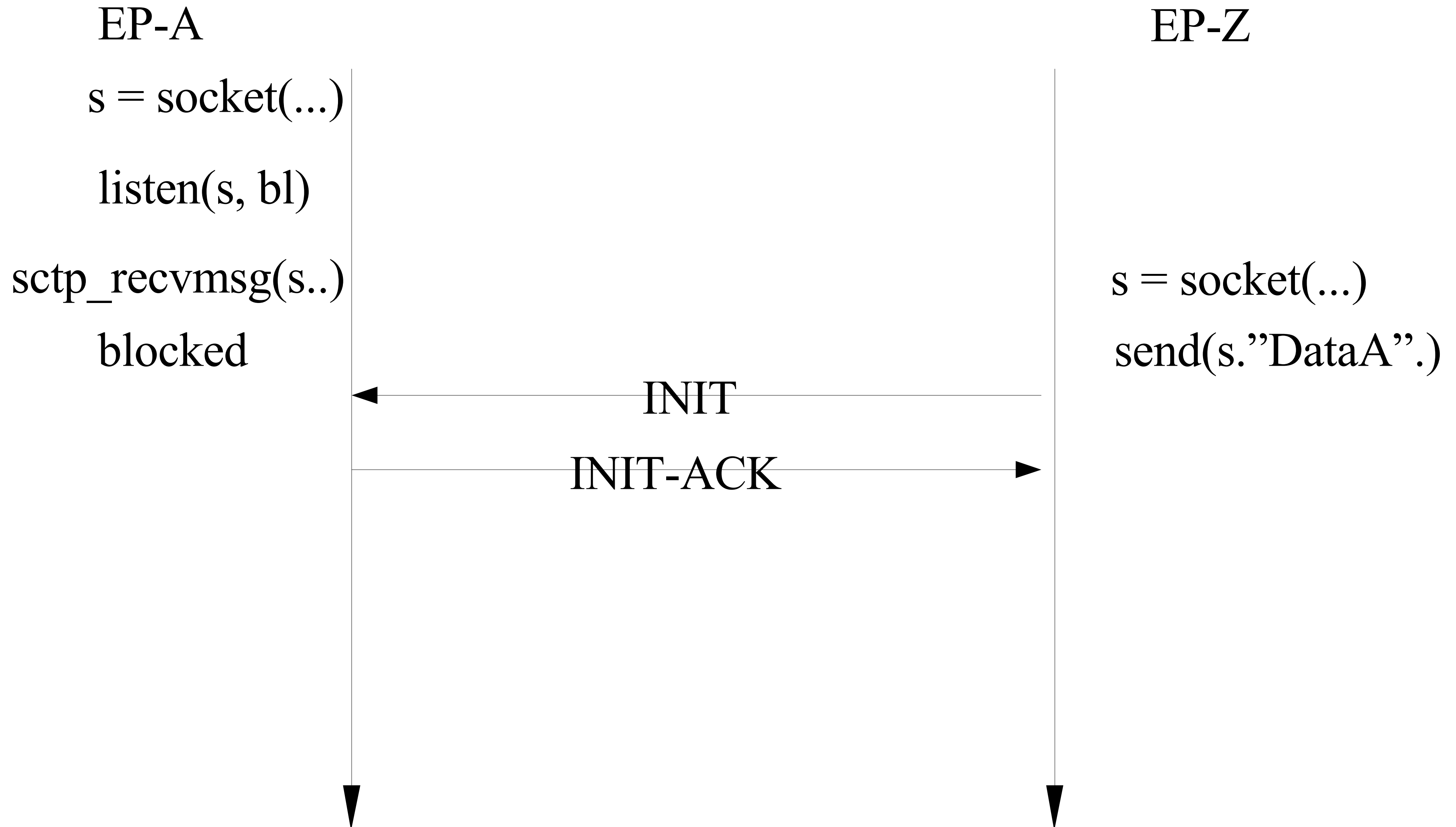
One-2-Many Model illustration



© 2003-2006 Cisco Systems, Inc. All rights reserved.

One-2-Many Model illustration

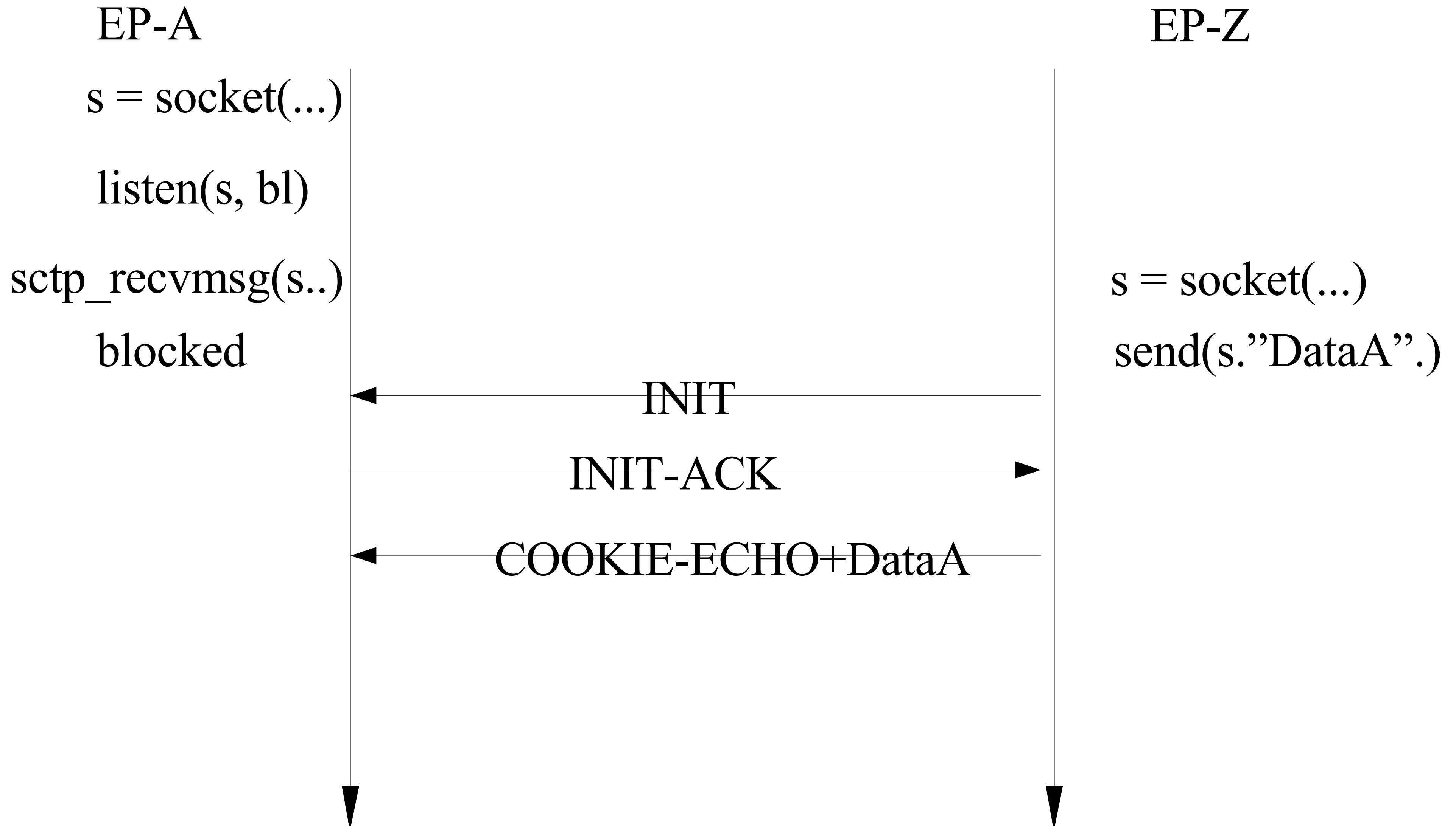
Cisco.com



© 2003-2006 Cisco Systems, Inc. All rights reserved.

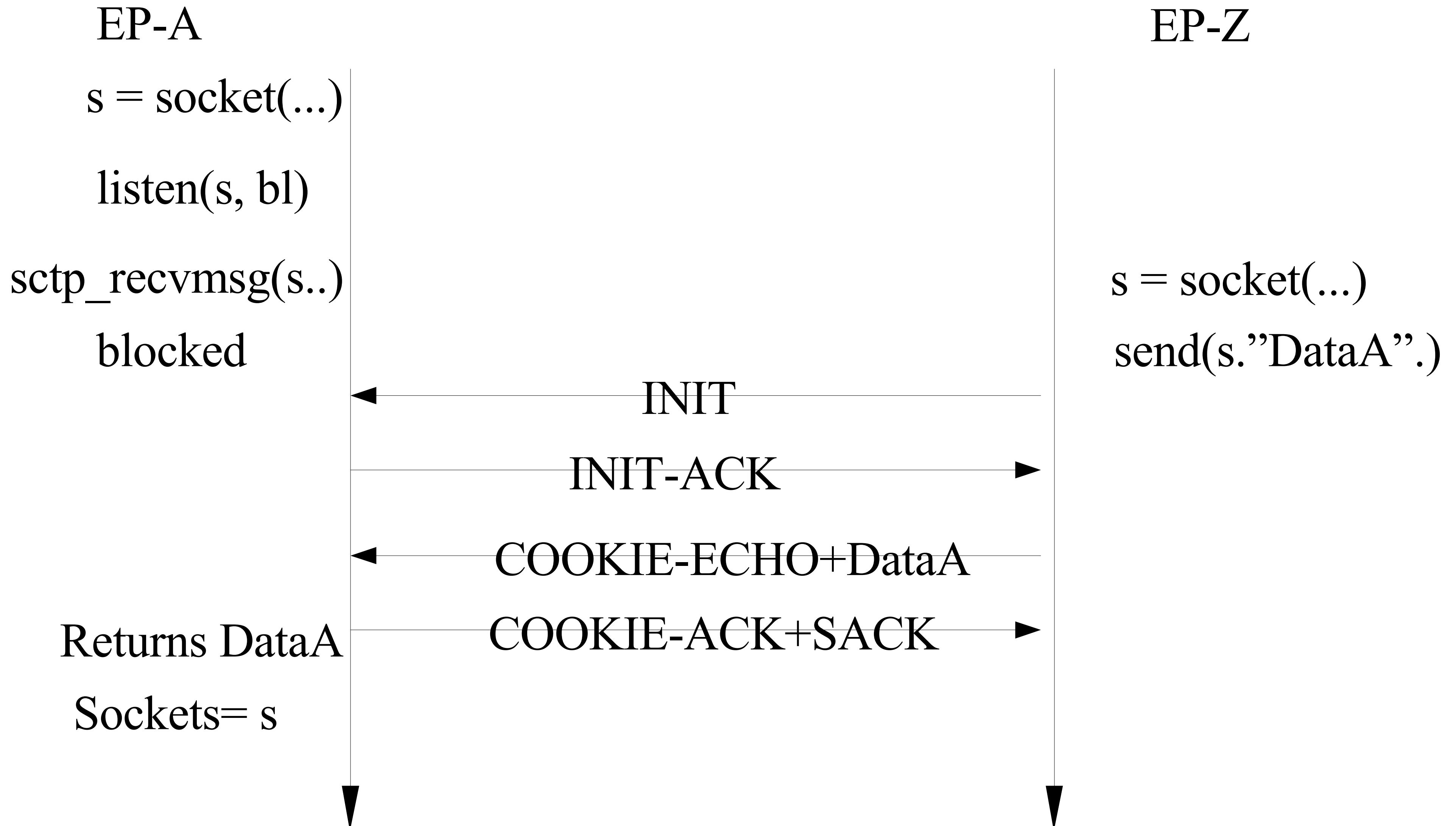
One-2-Many Model illustration

Cisco.com



© 2003-2006 Cisco Systems, Inc. All rights reserved.

One-2-Many Model illustration



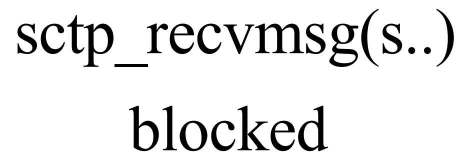
© 2003-2006 Cisco Systems, Inc. All rights reserved.

One-2-Many Model illustration

Cisco.com

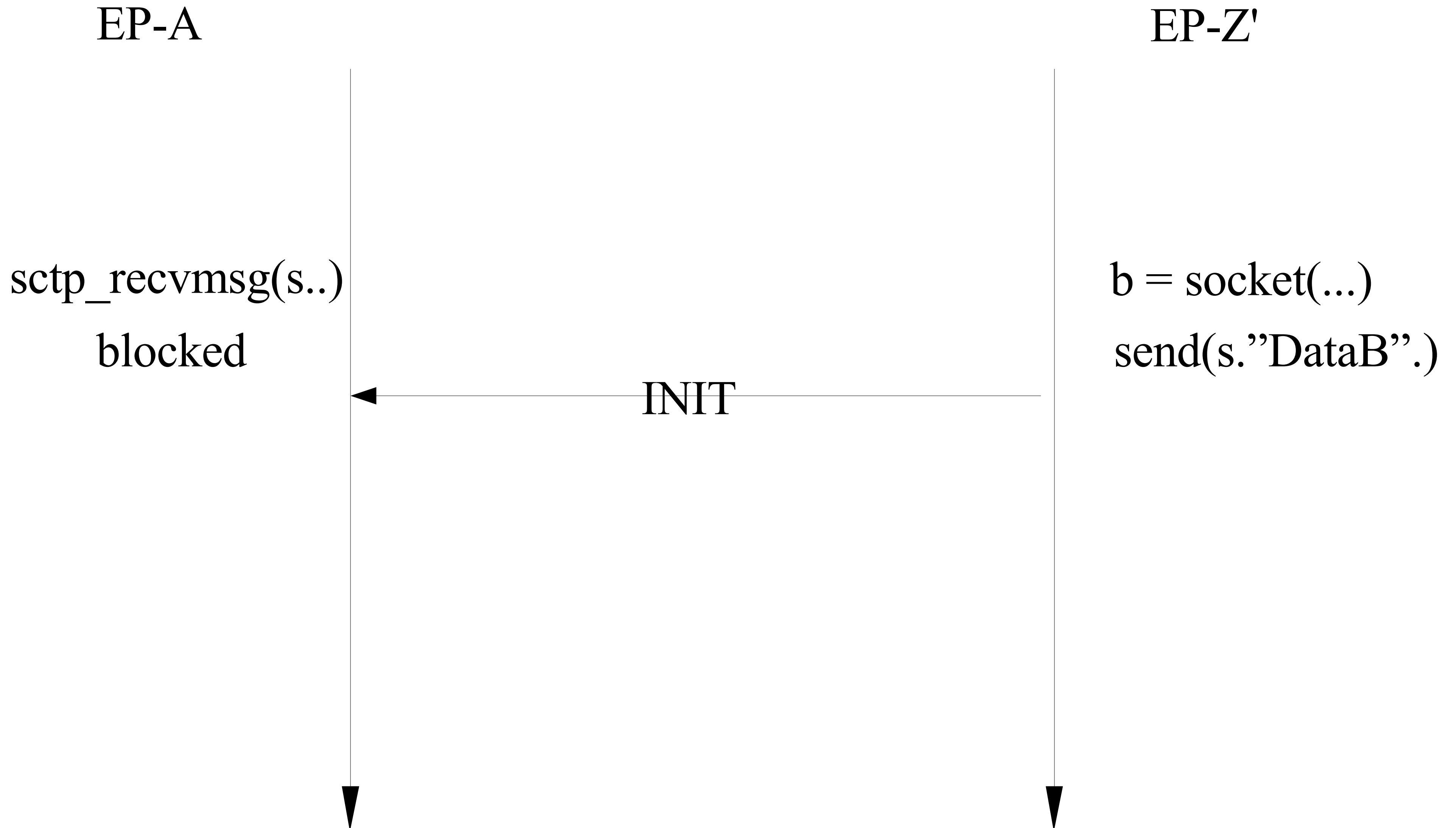
EP-A

EP-Z'



© 2003-2006 Cisco Systems, Inc. All rights reserved.

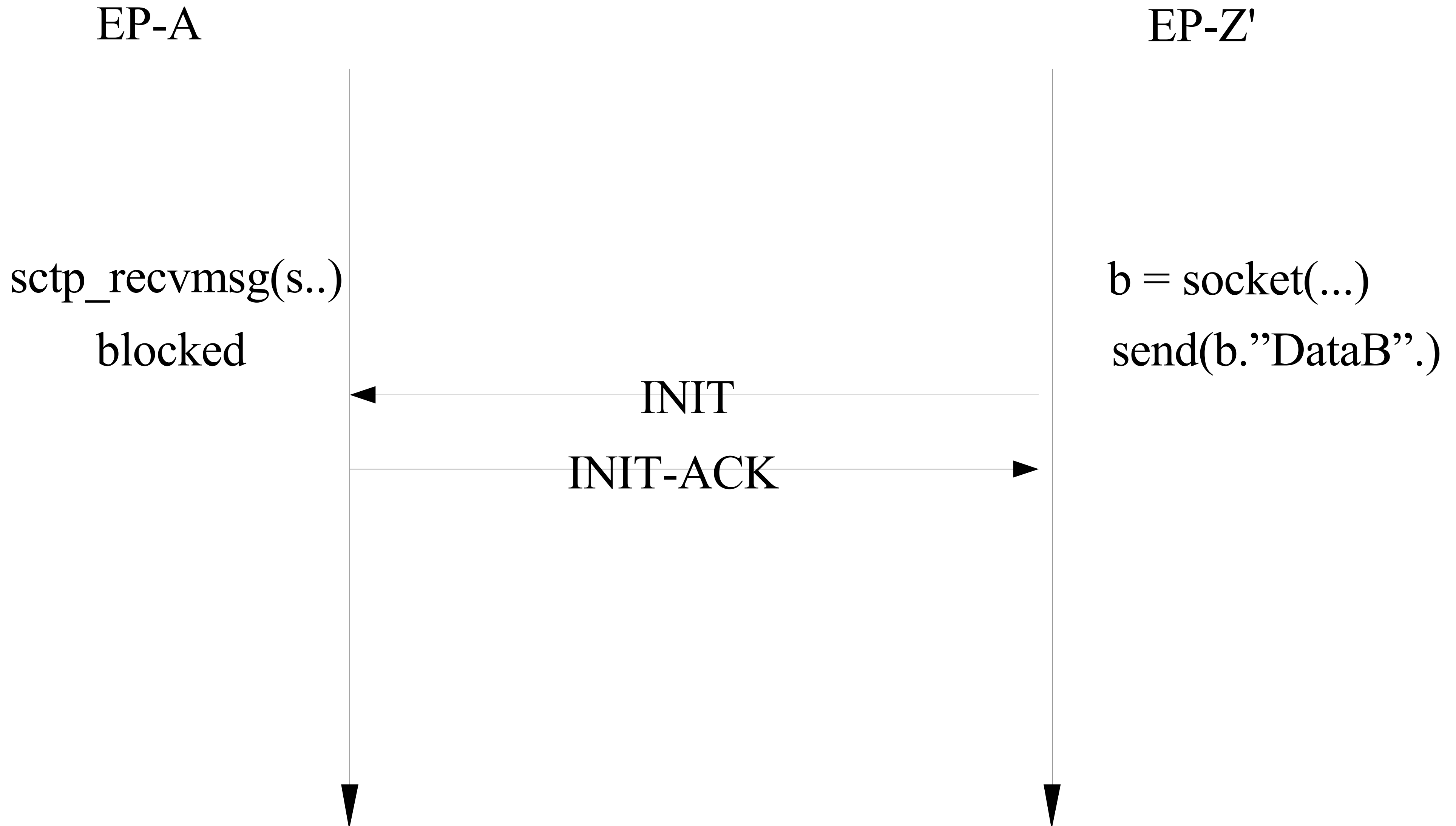
One-2-Many Model illustration



© 2003-2006 Cisco Systems, Inc. All rights reserved.

One-2-Many Model illustration

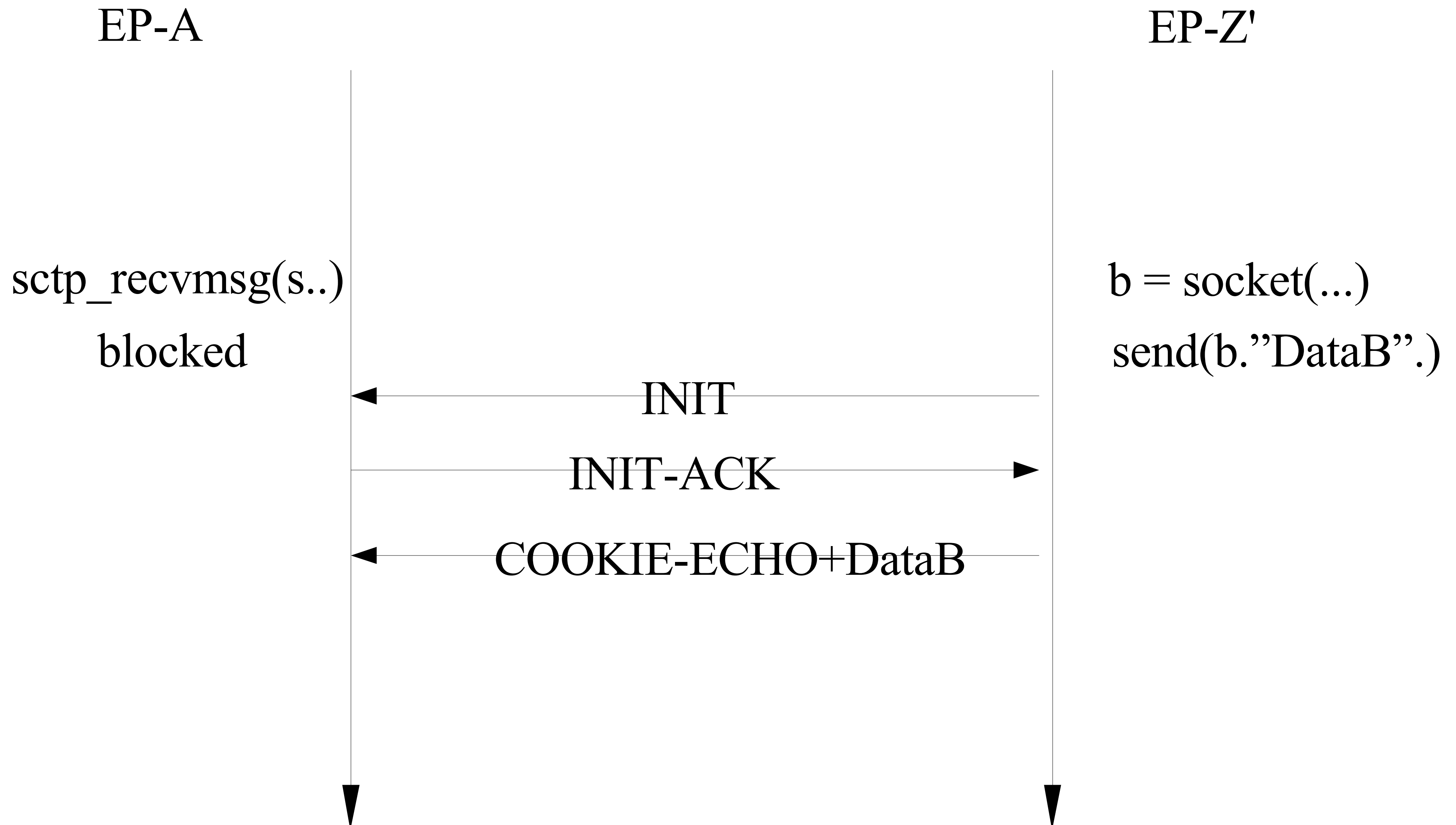
Cisco.com



© 2003-2006 Cisco Systems, Inc. All rights reserved.

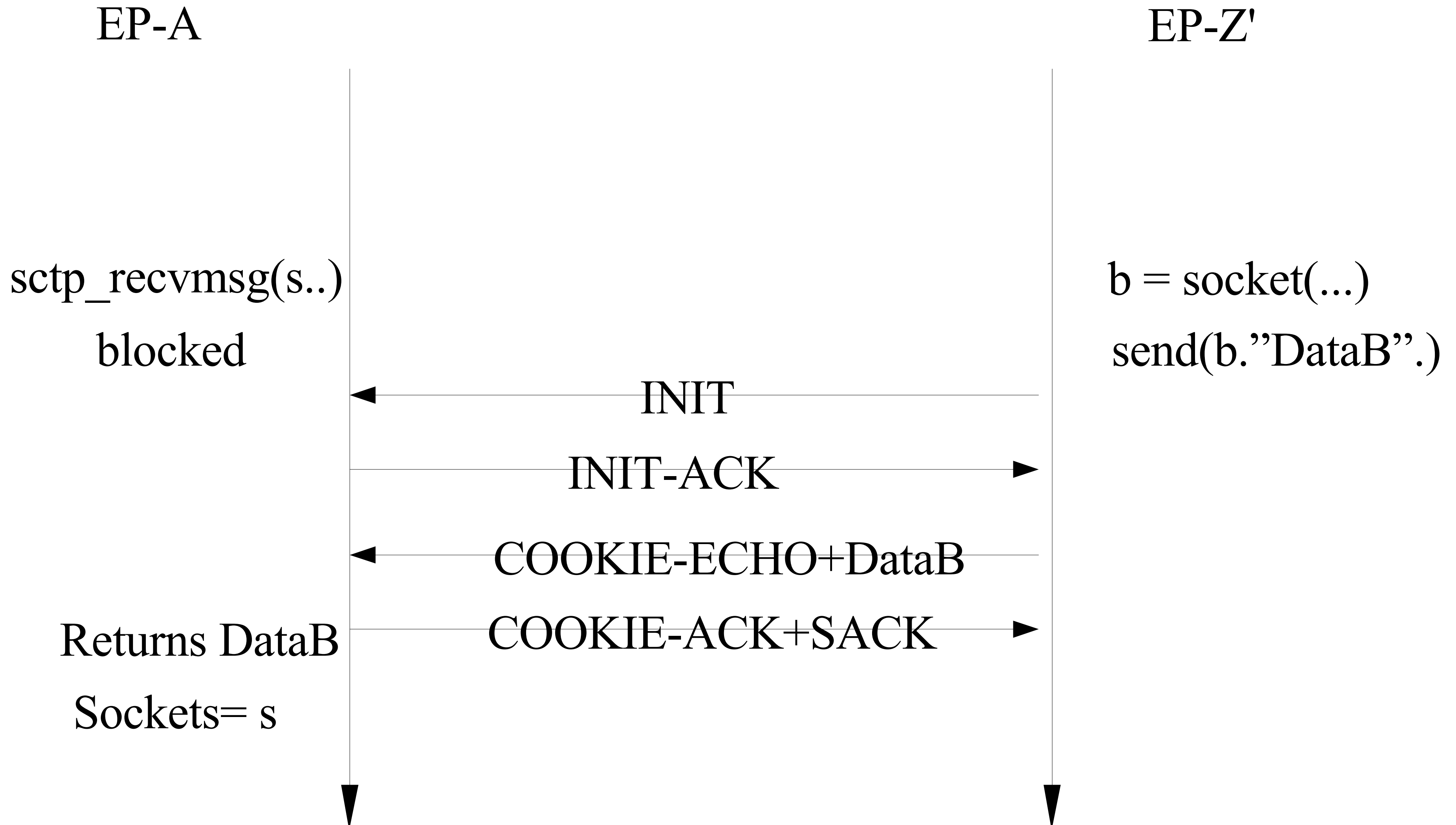
One-2-Many Model illustration

Cisco.com



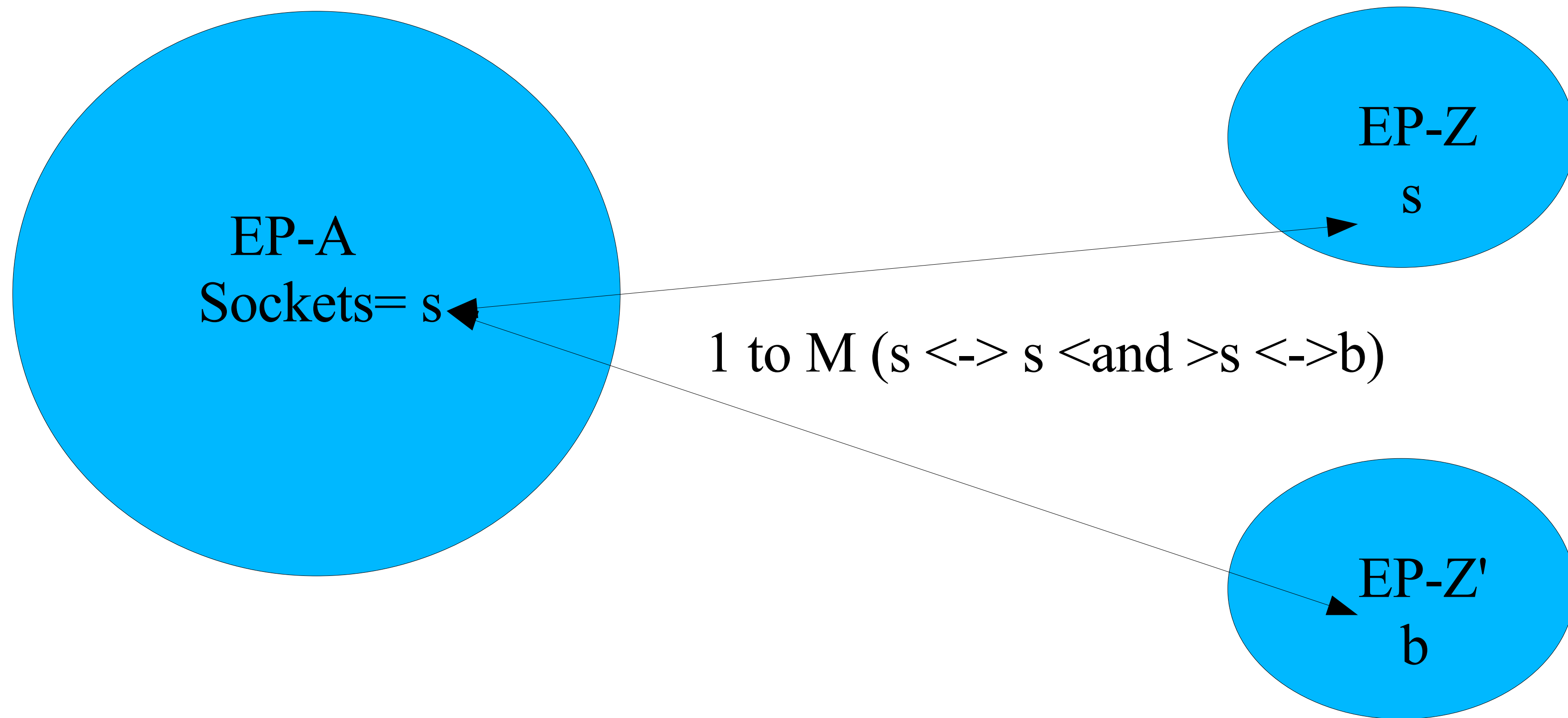
© 2003-2006 Cisco Systems, Inc. All rights reserved.

One-2-Many Model illustration



© 2003-2006 Cisco Systems, Inc. All rights reserved.

One-2-Many Model illustration



© 2003-2006 Cisco Systems, Inc. All rights reserved.

One-to-many Example Server


```
int sd, newfd, sosz, msg_flags;
struct sockaddr_in6 sin6;
struct sctp_sndrcvinfo snd_rcv;
char buf[8000];
sosz = sizeof(sin6);
sd = socket(AF_INET6, SOCK_SEQPACKET, IPPROTO_SCTP);
listen(sd, 1);
while (1) {
    len = sctp_recvmsg(sd, buf, sizeof(buf), (sockaddr *)&sin6, &sosz,
                       &snd_rcv, &msg_flags);
    do_msg_stuff(sd, buf, len, &sin6, &snd_rcv, msg_flags);
}
```


© 2003-2006 Cisco Systems, Inc. All rights reserved.

Using connect()

Cisco.com

- The connect() call causes an association to be setup to the specified address. Its syntax is:

```
int connect(int sd, const struct sockaddr *to,  socklen_t tolen)
```

- sd is the socket (returned from the socket call).
- to is a sockaddr_in or sockaddr_in6 address cast to a sockaddr type.
- tolen is the length of the address, e.g. sizeof(sockaddr_in) or sizeof(sockaddr_in6)
- For the one-to-one model, only the active open side may do a connect().

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Using connect()

- In most implementations, for the one-to-one model, connect() is blocking and only ONE connection may be open at a time.
- You may be able to re-use the socket after the association closes. You can use certain options to enact the close yourself without closing the socket with close().
- For the one-to-many model, connect() is non-blocking and it attempts to setup an association to the specified address.
- In the one-to-many model, connect() can be called multiple times with different addresses.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Using listen

- listen() tells the SCTP stack you are willing to accept connections, provided you supply a non-zero value for **backlog** with the call which has the form:

```
int listen(int sd, int backlog)
```

- A **backlog** value of zero turns off acceptance of associations.
- In the one-to-one model, once you do a listen with a non-zero **backlog**, you may not do a connect().
- For the one-to-many model this restriction does not apply.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Using accept()

- accept() is only used in the one-to-one model.
- A call to accept() in the one-to-many model should return an error.
- Accept has the form:

```
int accept(int sd, struct sockaddr *from, socklen_t *fromlen);
```

- The int returned is a new socket descriptor for the connected association.
- from returns an address of the peer.
- fromlen returns the length of the from address.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Socket Options

- Socket options are a method that one can control the protocol stack giving it direction as to how you would like it to function.
- You can both get and set various options.
- For TCP and UDP only a few options exists.
- For SCTP this is NOT the case, there are LOADs of options that we will NOT discuss all these “knobs” if you have a FreeBSD Current box you can go explore them at your leisure . :-)

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Sending Messages

- There are three functions that are often used with sockets to send messages.
  - sendmsg(int sd, struct msghdr *msg, int flags);
  - sendto(int sd, void *msg, size_t len, int flags,
    struct sockaddr *to, socklen_t tolen);
  - send(int sd, void *msg, size_t len, int flags);
- These all work the same for SCTP as they do for TCP and UDP, note that you cannot use send() on a one-2-many style socket.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

recv()/recvmsg()/recvfrom()

- **The receive functions recv(), recvmsg() and recvfrom() are the parallel functions to send(), sendmsg(), and sendto().**
- **They provide a way to receive data from a socket and take the following forms:**

```
ssize_t recv(int sd, void *buf, size_t len, int flags)

ssize_t recvfrom(int sd, void *buf, size len, int flags,
                        sockaddr *from, socklen_t * fromlen);

ssize_t recvmsg(int sd, struct msghdr *msg, int flags);
```


© 2003-2006 Cisco Systems, Inc. All rights reserved.

How to Tell SCTP to Send on a Stream?

- One thing that you will often want to do is send to various streams in an SCTP association.
- The standard socket API has no easy way for you to pass additional data to a SCTP association in its send...() calls.
- The only additional way to pass data is through the use of ancillary data.
- Ancillary data can only be passed or received with the sendmsg() or recvmsg() calls.
- This is what goes in the msg_control and msg_controllen fields.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

How to Tell SCTP to Send on a Stream?

- Pass an sctp_sndrcvinfo structure on your call via ancillary data.
- The sctp_sndrcvinfo structure:

```
struct sctp_sndrcvinfo {
    u_int16_t     sinfo_stream;
    u_int16_t     sinfo_ssn;
    u_int16_t     sinfo_flags;
    u_int32_t     sinfo_ppid;
    u_int32_t     sinfo_context;
    u_int32_t     sinfo_timetolive;
    u_int32_t     sinfo_tsn;
    u_int32_t     sinfo_cumtsn;
    sctp_assoc_t  sinfo_assoc_id;
};
```


© 2003-2006 Cisco Systems, Inc. All rights reserved.

How to Tell SCTP to Send on a Stream?

Cisco.com

```
struct sctp_sndrcvinfo {
      u_int16_t    sinfo_stream;
      u_int16_t    sinfo_ssn;
      u_int16_t    sinfo_flags;
      u_int32_t    sinfo_ppid;
      u_int32_t    sinfo_context;
      u_int32_t    sinfo_timetolive;
      u_int32_t    sinfo_tsn;
      u_int32_t    sinfo_cumtsn;
      sctp_assoc_t sinfo_assoc_id;
  };
```

The sinfo_stream field indicates what stream you would like to send information on. If you receive a sctp_sndrcvinfo structure this field tells you what stream the message was received on.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

How to Tell SCTP to Send on a Stream?


```
struct sctp_sndrcvinfo {
      u_int16_t     sinfo_stream;
      u_int16_t     sinfo_ssn;
      u_int16_t     sinfo_flags;
      u_int32_t     sinfo_ppid;
      u_int32_t     sinfo_context;
      u_int32_t     sinfo_timetolive;
      u_int32_t     sinfo_tsn;
      u_int32_t     sinfo_cumtsn;
      sctp_assoc_t sinfo_assoc_id;
  };
```

The sinfo_ssn is the stream sequence number (SSN) that this message was assigned (on reading). On sending this field as no effect (you can't control what the SSN will be only the stack does that based on your sending order).

© 2003-2006 Cisco Systems, Inc. All rights reserved.

How to Tell SCTP to Send on a Stream?


```
struct sctp_sndrcvinfo {
      u_int16_t    sinfo_stream;
      u_int16_t    sinfo_ssn;
      u_int16_t    sinfo_flags;
      u_int32_t    sinfo_ppid;
      u_int32_t    sinfo_context;
      u_int32_t    sinfo_timetolive;
      u_int32_t    sinfo_tsn;
      u_int32_t    sinfo_cumtsn;
      sctp_assoc_t sinfo_assoc_id;
  };
```

The sinfo_flags field holds a number of various flags that were originally begun with a MSG_xxx format but now have an SCTP_xxx format.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Meanings of the sctp_sndrcvinfo fields

- The current **sinfo_flags** field may hold one or more of the following:

  SCTP_UNORDERED – On sending request un-ordered service. On receiving indicates the message is unordered and the sinfo_ssn field has no meaning.

  SCTP_ADDR_OVER – On sending request that the address included on the send (msg_name) be used even if the primary address points elsewhere. Unused for receive.

  SCTP_ABORT – Request that the association be aborted. Any data in the message send is treated as a “user abort code” and passed with the abort as a “user reason”. Note this will not be seen on receive.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Meanings of the sctp_sndrcvinfo fields

- The current sinfo_flags field may hold one or more of the following:

    SCTP_EOF – Request that after this message is queued the graceful shutdown procedures be initiated. This puts the sender into SHUTDOWN_PENDING and the local side will NOT be allowed to send more data. Unused on receive.

    SCTP_SENDALL – This flag is only effective with the one-to-many model. It requests that the data in the send be transmitted to ALL associations. This provides a method to send the same message in a reliable way with only one send call instead of multiple send calls.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Meanings of the sctp_sndrcvinfo fields

- The current sinfo_flags field may hold one or more of the following:

    SCTP_PR_SCTP_TTL – Enable PR-SCTP and use the time-to-live profile where the sinfo_timetolive holds the number of milliseconds the message is good for.

    SCTP_PR_SCTP_BUF – Enable PR-SCTP and use the buffer bounded base sending where sinfo_timetolive holds the largest my send buffers can grow to before I start skipping data.

    SCTP_PR_SCTP_RTX – Enable PR-SCTP and use the number of retransmissions set in sinfo_timetolive. Once I reach that limit I will skip it.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

How to Tell SCTP to Send on a Stream?


```
struct sctp_sndrcvinfo {
      u_int16_t     sinfo_stream;
      u_int16_t     sinfo_ssn;
      u_int16_t     sinfo_flags;
      u_int32_t     sinfo_ppid;
      u_int32_t     sinfo_context;
      u_int32_t     sinfo_timetolive;
      u_int32_t     sinfo_tsn;
      u_int32_t     sinfo_cumtsn;
      sctp_assoc_t sinfo_assoc_id;
  };
```

The sinfo_ppid flag holds the payload protocol identifier. On send, it places that value into the outbound SCTP data chunk. On receive it indicates what PPID the peer sent. Note that you must put this in network byte order and translate it (if needed).

© 2003-2006 Cisco Systems, Inc. All rights reserved.

How to Tell SCTP to Send on a Stream?


```
struct sctp_sndrcvinfo {
       u_int16_t    sinfo_stream;
       u_int16_t    sinfo_ssn;
       u_int16_t    sinfo_flags;
       u_int32_t    sinfo_ppid;
       u_int32_t    sinfo_context;
       u_int32_t    sinfo_timetolive;
       u_int32_t    sinfo_tsn;
       u_int32_t    sinfo_cumtsn;
       sctp_assoc_t sinfo_assoc_id;
  };
```

The sinfo_context is a 32-bit opaque field that can be used on send failure reports. If a send fails, this value will be passed up with the application data that could not be sent. A user may wish to use this for specific context lookup (like finding a state machine).

© 2003-2006 Cisco Systems, Inc. All rights reserved.

How to Tell SCTP to Send on a Stream?

Cisco.com

```
struct sctp_sndrcvinfo {
      u_int16_t     sinfo_stream;
      u_int16_t     sinfo_ssn;
      u_int16_t     sinfo_flags;
      u_int32_t     sinfo_ppid;
      u_int32_t     sinfo_context;
      u_int32_t     sinfo_timetolive;
      u_int32_t     sinfo_tsn;
      u_int32_t     sinfo_cumtsn;
      sctp_assoc_t sinfo_assoc_id;
  };
```

The sinfo_timetolive field is the amount of time the data will be considered valid for. This is effectivve for PR-SCTP (the partial reliability feature) it also may be effective without PR-SCTP if the data remains in queue but is never put on the wire.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

How to Tell SCTP to Send on a Stream?

Cisco.com

```
struct sctp_sndrcvinfo {
      u_int16_t    sinfo_stream;
      u_int16_t    sinfo_ssn;
      u_int16_t    sinfo_flags;
      u_int32_t    sinfo_ppid;
      u_int32_t    sinfo_context;
      u_int32_t    sinfo_timetolive;
      u_int32_t    sinfo_tsn;
      u_int32_t    sinfo_cumtsn;
      sctp_assoc_t sinfo_assoc_id;
  };
```

The sinfo_tsn and sinfo_cumtsn are only valid on receive and hold the TSN that was assigned by this peer to the data chunk and the current cumulative TSN. Note that for multi-chunk messages this will hold ONE of the TSN's that made up the message.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

How to Tell SCTP to Send on a Stream?


```
struct sctp_sndrcvinfo {
      u_int16_t     sinfo_stream;
      u_int16_t     sinfo_ssn;
      u_int16_t     sinfo_flags;
      u_int32_t     sinfo_ppid;
      u_int32_t     sinfo_context;
      u_int32_t     sinfo_timetolive;
      u_int32_t     sinfo_tsn;
      u_int32_t     sinfo_cumtsn;
      sctp_assoc_t sinfo_assoc_id;
  };
```

The sinfo_assoc_id can be used for both sending and receiving. It indicates on the receive side which association the data came from (this is useful for the one-2-many model). And it can be used to indicate which association is to be sent to. This field will take precedence over the address (if supplied) in identifying the association.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Meanings of the sctp_sndrcvinfo fields

- So now that you know WHAT structure to fill in, how do you pass it in a msg_control field (it is NOT just a simple place in the msg_control field a pointer to the sndrcv_info structure)?
- An example is the best way to explain how to proceed.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Using ancillary data with SCTP


```
struct sctp_sndrcvinfo *s_info;
ssize_t sz;
int sinfo_found=0;
struct msghdr msg;
struct iovec iov[2];
char controlVector[SCTP_CONTROL_VEC_SIZE_RCV];
struct cmsghdr *cmsg;

if (msg_flags == NULL) {
        errno = EINVAL;
        return (-1);
}
msg.msg_flags = 0;
iov[0].iov_base = dbuf;
iov[0].iov_len = len;
iov[1].iov_base = NULL;
```


© 2003-2006 Cisco Systems, Inc. All rights reserved.

Using ancillary data with SCTP (cont)


```
iov[1].iov_len = 0;
msg.msg_name = (caddr_t)from;
if (fromlen == NULL)
        msg.msg_namelen = 0;
else
        msg.msg_namelen = *fromlen;
msg.msg_iov = iov;
msg.msg_iovlen = 1;
msg.msg_control = (caddr_t)controlVector;
msg.msg_controllen = sizeof(controlVector);
errno = 0;
sz = recvmsg(s,&msg,0);
if(sz <= 0)
        return(sz);
```


© 2003-2006 Cisco Systems, Inc. All rights reserved.

Using ancillary data with SCTP (cont)


```
        s_info = NULL;
        len = sz;
        *msg_flags = msg.msg_flags;
        if(sinfo)
                sinfo->sinfo_assoc_id = 0;

        if ((msg.msg_controllen) && sinfo) {
                /* parse through and see if we find
                 * the sctp_sndrcvinfo (if the user wants it).
                 */
                cmsg = (struct cmsghdr *)controlVector;
                while (cmsg) {
                        if((cmsg->cmsg_len == 0) || (cmsg->cmsg_len >
msg.msg_controllen)) {
                                break;
                        }
```


© 2003-2006 Cisco Systems, Inc. All rights reserved.

Using ancillary data with SCTP (cont)


```
            if (cmsg->cmsg_level == IPPROTO_SCTP) {
                if (cmsg->cmsg_type == SCTP_SNDRCV) {
                    s_info = (struct sctp_sndrcvinfo *)
                        CMSG_DATA(cmsg);
                    /* Copy it to the user */
                    if(sinfo)
                        *sinfo = *s_info;
                    sinfo_found = 1;
                    break;
                }
            }
            cmsg = CMSG_NXTHDR(&msg,cmsg);
        }
    }
    return(sz);
```


© 2003-2006 Cisco Systems, Inc. All rights reserved.

sctp_send, sctp_sendmsg, sctp_recvmsg()

- **Now that you have had a small taste of ancillary data you may think - "ugh do I really have to do all that"?**
- **The answer, thank goodness, is NO!**
- **There are a standard set of defined "SCTP" calls that allow you to not to worry about the use of ancillary data.**
- **sctp_send, sctp_sendmsg and sctp_recvmsg provide you with the functionality you want without the headache of using ancillary data. Often times they are pure library functions that implement the ancillary data for you (no longer true in FreeBSD 6.x/7.x however).**

© 2003-2006 Cisco Systems, Inc. All rights reserved.

sctp_sendmsg()

- The sctp_sendmsg() call is the most verbose with arguments:

```
sctp_sendmsg(int sd, const void *data, size_t len,
             const struct sockaddr *to, socklen_t tolen,
             u_int32_t ppid, u_int32_t flags,
             u_int32_t stream_no, u_int32_t timetolive,
             u_int32_t context)
```

- You will note an easy mapping to the sndrcvinfo structure. Note that this is quite useful for the sender that is either sending to a one-to-one model socket OR to one that is going to always supply a socket address to send to.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

sctp_sendmsg()


```
sctp_sendmsg(int sd, const void *data, size_t len,
             const struct sockaddr *to, socklen_t tolen,
             u_int32_t ppid, u_int32_t flags,
             u_int32_t stream_no, u_int32_t timetolive,
             u_int32_t context)
```

- The sd field is of course the socket.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

sctp_sendmsg()


```
sctp_sendmsg(int sd, const void *data, size_t len,
                     const struct sockaddr *to, socklen_t tolen,
                     u_int32_t ppid, u_int32_t flags,
                     u_int32_t stream_no, u_int32_t timetolive,
                     u_int32_t context)
```

- **The data and len argument hold the message and the length of the message respectively.**

© 2003-2006 Cisco Systems, Inc. All rights reserved.

sctp_sendmsg()


```
sctp_sendmsg(int sd, const void *data, size_t len,
             const struct sockaddr *to, socklen_t tolen,
             u_int32_t ppid, u_int32_t flags,
             u_int32_t stream_no, u_int32_t timetolive,
             u_int32_t context)
```

- The **to** and **tolen** hold the socket address and length of the address.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

sctp_sendmsg()


```
sctp_sendmsg(int sd, const void *data, size_t len,
             const struct sockaddr *to, socklen_t tolen,
             u_int32_t ppid, u_int32_t flags,
             u_int32_t stream_no, u_int32_t timetolive,
             u_int32_t context)
```

- **The ppid argument holds the same value that goes in the sinfo_ppid**

© 2003-2006 Cisco Systems, Inc. All rights reserved.

sctp_sendmsg()


```
sctp_sendmsg(int sd, const void *data, size_t len,
             const struct sockaddr *to, socklen_t tolen,
             u_int32_t ppid, u_int32_t flags,
             u_int32_t stream_no, u_int32_t timetolive,
             u_int32_t context)
```

- The **flags** argument holds the same value that goes in the sinfo_flags.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

sctp_sendmsg()


```
sctp_sendmsg(int sd, const void *data, size_t len,
             const struct sockaddr *to, socklen_t tolen,
             u_int32_t ppid, u_int32_t flags,
             u_int32_t stream_no, u_int32_t timetolive,
             u_int32_t context)
```

- The stream_no holds the same value that goes in the sinfo_stream.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

sctp_sendmsg()


```
sctp_sendmsg(int sd, const void *data, size_t len,
             const struct sockaddr *to, socklen_t tolen,
             u_int32_t ppid, u_int32_t flags,
             u_int32_t stream_no, u_int32_t timetolive,
             u_int32_t context)
```

- The timetolive holds the same value that goes in the sinfo_timetolive.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

sctp_sendmsg()


```
sctp_sendmsg(int sd, const void *data, size_t len,
             const struct sockaddr *to, socklen_t tolen,
             u_int32_t ppid, u_int32_t flags,
             u_int32_t stream_no, u_int32_t timetolive,
             u_int32_t context)
```

- **The context holds the same value that goes into the sinfo_context.**

© 2003-2006 Cisco Systems, Inc. All rights reserved.

sctp_recvmsg()

- The sctp_recvmsg() call takes the form:

```
sctp_recvmsg (int sd, const void *data, size_t len,
              struct sockaddr *from, socklen_t *fromlen,
              const struct sctp_sndrcvinfo *sinfo,
              int *msg_flags);
```

- The sd is of course the socket.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

sctp_recvmsg()

- The sctp_recvmsg() call takes the form:

```
sctp_recvmsg (int sd, const void *data, size_t len,
              struct sockaddr *from, socklen_t *fromlen,
              const struct sctp_sndrcvinfo *sinfo,
              int *msg_flags);
```

- The data and len hold the message and maximum message length respectively.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

sctp_recvmsg()

- The sctp_recvmsg() call takes the form:

```
sctp_recvmsg (int sd, const void *data, size_t len,
              struct sockaddr *from, socklen_t *fromlen,
              const struct sctp_sndrcvinfo *sinfo,
              int *msg_flags);
```

- The from and fromlen hold the space for the address and the length of that space.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

sctp_recvmsg()

Cisco.com

- The sctp_recvmsg() call takes the form:

```
sctp_recvmsg (int sd, const void *data, size_t len,
              struct sockaddr *from, socklen_t *fromlen,
              const struct sctp_sndrcvinfo *sinfo,
              int *msg_flags);
```

- The sinfo argument holds a pointer to the sctp_sndrcvinfo structure that will be filled in.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

sctp_recvmsg()

- The sctp_recvmsg() call takes the form:

```
sctp_recvmsg (int sd, const void *data, size_t len,
              struct sockaddr *from, socklen_t *fromlen,
              const struct sctp_sndrcvinfo *sinfo,
              int *msg_flags);
```

- The **msg_flags** field will, on return, hold the value that would have been returned in the recvmsg() call's msg_flags field including:
  - MSG_NOTIFICATION – the message is from SCTP not the peer (we will learn about these soon).
  - MSG_EOR – This is a complete message (or completion of a message if the last read did NOT have MSG_EOR).

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Notifications

- The SCTP also has a LOT of various transport events that can be communicated to an application.
- These events are called notifications.
- No application can receive notifications unless they subscribe to them.
- The above restriction is placed on notifications since they appear in the data path and an uninformed application might mistake them for user data.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Types of Notifications

- **There are eight currently defined notifications (and one other that may soon appear). We will only discuss the currently defined notifications.**

  SCTP_ASSOC_CHANGE

  SCTP_PEER_ADDR_CHANGE

  SCTP_REMOTE_ERROR

  SCTP_SEND_FAILED

  SCTP_SHUTDOWN_EVENT

  SCTP_ADAPTION_INDICATION

  SCTP_PARTIAL_DELIVERY_EVENT

  SCTP_AUTHENTICATION_EVENT

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Types of Notifications

- Each event has a specific data structure associated with it. We will now look at each event, why you get it, and its appropriate structure.
- SCTP_ASSOC_CHANGE tells an application about changes in associations. These include:
  - SCTP_COMM_UP communication is up to endpoint.
  - SCTP_COMM_LOST lost communication with an endpoint.
  - SCTP_RESTART a restart of an association has occurred.
  - SCTP_SHUTDOWN_COMP - A shutdown of an association completed.
  - SCTP_CANT_STR_ASSOC – An association cannot be setup to a peer.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Association Change notification

- **The structure passed to the application looks as follows:**

```
struct sctp_assoc_change {
    uint16_t      sac_type;
    uint16_t      sac_flags;
    uint32_t      sac_length;
    uint16_t      sac_state;
    uint16_t      sac_error;
    uint16_t      sac_outbound_streams;
    uint16_t      sac_inbound_streams;
    sctp_assoc_t  sac_assoc_id;
};
```

- **The sac_type will be set to SCTP_ASSOC_CHANGE**

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Association Change notification

Cisco.com

```
struct sctp_assoc_change {
      uint16_t     sac_type;
      uint16_t     sac_flags;
      uint32_t     sac_length;
      uint16_t     sac_state;
      uint16_t     sac_error;
      uint16_t     sac_outbound_streams;
      uint16_t     sac_inbound_streams;
      sctp_assoc_t sac_assoc_id;
  };
```

- sac_flags are 0 and sac_length is set to the size of the notification structure.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Association Change notification


```
struct sctp_assoc_change {
      uint16_t     sac_type;
      uint16_t     sac_flags;
      uint32_t     sac_length;
      uint16_t     sac_state;
      uint16_t     sac_error;
      uint16_t     sac_outbound_streams;
      uint16_t     sac_inbound_streams;
      sctp_assoc_t sac_assoc_id;
 };
```

- sac_state will hold the type: SCTP_COMM_UP, etc.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Association Change notification

Cisco.com

```
struct sctp_assoc_change {
     uint16_t     sac_type;
     uint16_t     sac_flags;
     uint32_t     sac_length;
     uint16_t     sac_state;
     uint16_t     sac_error;
     uint16_t     sac_outbound_streams;
     uint16_t     sac_inbound_streams;
     sctp_assoc_t sac_assoc_id;
 };
```

- sac_error would indicate any error.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Association Change notification


```
struct sctp_assoc_change {
      uint16_t      sac_type;
      uint16_t      sac_flags;
      uint32_t      sac_length;
      uint16_t      sac_state;
      uint16_t      sac_error;
      uint16_t      sac_outbound_streams;
      uint16_t      sac_inbound_streams;
      sctp_assoc_t sac_assoc_id;
  };
```

- **sac_outbound_streams will indicate how many streams are setup outbound.**

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Association Change notification


```
struct sctp_assoc_change {
      uint16_t     sac_type;
      uint16_t     sac_flags;
      uint32_t     sac_length;
      uint16_t     sac_state;
      uint16_t     sac_error;
      uint16_t     sac_outbound_streams;
      uint16_t     sac_inbound_streams;
      sctp_assoc_t sac_assoc_id;
 };
```

- **sac_inbound_streams will indicate how many inbound streams were setup.**

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Association Change notification

Cisco.com

```
struct sctp_assoc_change {
      uint16_t      sac_type;
      uint16_t      sac_flags;
      uint32_t      sac_length;
      uint16_t      sac_state;
      uint16_t      sac_error;
      uint16_t      sac_outbound_streams;
      uint16_t      sac_inbound_streams;
      sctp_assoc_t sac_assoc_id;
  };
```

- sac_assoc_id is the unique association id.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Address Change notification

- SCTP_PEER_ADDR_CHANGE is passed to the user when a event occurs that effects a peer address.
- The structure passed looks as follows:

```
struct sctp_paddr_change {
    uint16_t      spc_type;
    uint16_t      spc_flags;
    uint32_t      spc_length;
    struct sockaddr_storage spc_addr;
    uint32_t      spc_state;
    uint32_t      spc_error;
    sctp_assoc_t spc_assoc_id;
};
```


© 2003-2006 Cisco Systems, Inc. All rights reserved.

Address Change notification


```
struct sctp_paddr_change {
      uint16_t      spc_type;
      uint16_t      spc_flags;
      uint32_t      spc_length;
      struct sockaddr_storage spc_addr;
      uint32_t      spc_state;
      uint32_t      spc_error;
      sctp_assoc_t spc_assoc_id;
  };
```

- spc_type holds the value SCTP_PEER_ADDR_CHANGE.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Address Change notification


```
struct sctp_paddr_change {
      uint16_t     spc_type;
      uint16_t     spc_flags;
      uint32_t     spc_length;
      struct sockaddr_storage spc_addr;
      uint32_t     spc_state;
      uint32_t     spc_error;
      sctp_assoc_t spc_assoc_id;
  };
```

- **spc_flags is 0 and spc_length is the length of this structure.**

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Address Change notification


```
struct sctp_paddr_change {
      uint16_t     spc_type;
      uint16_t     spc_flags;
      uint32_t     spc_length;
      struct sockaddr_storage spc_addr;
      uint32_t     spc_state;
      uint32_t     spc_error;
      sctp_assoc_t spc_assoc_id;
 };
```

- **spc_addr is the address that the notification is about.**

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Address Change notification


```
struct sctp_paddr_change {
      uint16_t      spc_type;
      uint16_t      spc_flags;
      uint32_t      spc_length;
      struct sockaddr_storage spc_addr;
      uint32_t      spc_state;
      uint32_t      spc_error;
      sctp_assoc_t spc_assoc_id;
  };
```

- **spc_state indicate one of a number of events**
  - **SCTP_ADDR_AVAILABLE – address is now available and working.**
  - **SCTP_ADDR_UNREACHABLE – an address is no longer reachable (it hit a threshold for failure). Heartbeats continue to this address but data will not be sent to this address.**

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Address Change notification


```
struct sctp_paddr_change {
        uint16_t      spc_type;
        uint16_t      spc_flags;
        uint32_t      spc_length;
        struct sockaddr_storage spc_addr;
        uint32_t      spc_state;
        uint32_t      spc_error;
        sctp_assoc_t spc_assoc_id;
  };
```

–SCTP_ADDR_REMOVED – A dynamic address event has removed an address from an association.

–SCTP_ADDR_ADDED – A dynamic address event has added an address to the association.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Address Change notification


```
struct sctp_paddr_change {
      uint16_t      spc_type;
      uint16_t      spc_flags;
      uint32_t      spc_length;
      struct sockaddr_storage spc_addr;
      uint32_t      spc_state;
      uint32_t      spc_error;
      sctp_assoc_t spc_assoc_id;
  };
```

–**SCTP_ADDR_MADE_PRIM – An address was made primary (normally in response to a peer request).**

–**SCTP_ADDR_CONFIRMED – An address has been confirmed.**

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Address Change notification


```
struct sctp_paddr_change {
      uint16_t      spc_type;
      uint16_t      spc_flags;
      uint32_t      spc_length;
      struct sockaddr_storage spc_addr;
      uint32_t      spc_state;
      uint32_t      spc_error;
      sctp_assoc_t spc_assoc_id;
  };
```

- `spc_error` holds any error that may have caused this.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Address Change notification


```
struct sctp_paddr_change {
      uint16_t      spc_type;
      uint16_t      spc_flags;
      uint32_t      spc_length;
      struct sockaddr_storage spc_addr;
      uint32_t      spc_state;
      uint32_t      spc_error;
      sctp_assoc_t spc_assoc_id;
  };
```

- `spc_assoc_id` indicates the association id for the address change.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Remote Error notification

- The SCTP_REMOTE_ERROR notification relays error notifications sent by the peer SCTP stack. Such notifications could indicate the use of a invalid stream.
- The structure is as follows:

```
struct sctp_remote_error {
    uint16_t      sre_type;
    uint16_t      sre_flags;
    uint32_t      sre_length;
    uint16_t      sre_error;
    sctp_assoc_t sre_assoc_id;
    uint8_t       sre_data[4];
};
```

- Note this is NOT a common notification and most applications do not subscribe to this event.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Remote Error notification


```
struct sctp_remote_error {
      uint16_t      sre_type;
      uint16_t      sre_flags;
      uint32_t      sre_length;
      uint16_t      sre_error;
      sctp_assoc_t  sre_assoc_id;
      uint8_t       sre_data[4];
  };
```

- **sre_type** holds SCTP_REMOTE_ERROR, **sre_flags** is set to 0, and **sre_length** is the size of this structure plus the length of the error being passed in **sre_data**.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Remote Error notification

Cisco.com

```
struct sctp_remote_error {
        uint16_t      sre_type;
        uint16_t      sre_flags;
        uint32_t      sre_length;
        uint16_t      sre_error;
        sctp_assoc_t  sre_assoc_id;
        uint8_t       sre_data[4];
  };
```

- sre_error holds the error being delivered

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Remote Error notification

Cisco.com

```
struct sctp_remote_error {
      uint16_t      sre_type;
      uint16_t      sre_flags;
      uint32_t      sre_length;
      uint16_t      sre_error;
      sctp_assoc_t sre_assoc_id;
      uint8_t       sre_data[4];
  };
```

- `sre_assoc_id` indicates the association id that encountered the error.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Remote Error notification


```
struct sctp_remote_error {
     uint16_t      sre_type;
     uint16_t      sre_flags;
     uint32_t      sre_length;
     uint16_t      sre_error;
     sctp_assoc_t sre_assoc_id;
     uint8_t       sre_data[4];
  };
```

- sre_data is the actual TLV of the error being sent to the SCTP stack.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Send Failure notification

- SCTP_SEND_FAILED indicates a message was not sent and is usually followed closely by a communication lost notification.
- Its format is:

```
struct sctp_send_failed {
    uint16_t          ssf_type;
    uint16_t          ssf_flags;
    uint32_t          ssf_length;
    uint32_t          ssf_error;
    struct sctp_sndrcvinfo ssf_info;
    sctp_assoc_t      ssf_assoc_id;
    uint8_t           ssf_data[4];
};
```


© 2003-2006 Cisco Systems, Inc. All rights reserved.

Send Failure notification


```
struct sctp_send_failed {
      uint16_t          ssf_type;
      uint16_t          ssf_flags;
      uint32_t          ssf_length;
      uint32_t          ssf_error;
      struct sctp_sndrcvinfo ssf_info;
      sctp_assoc_t      ssf_assoc_id;
      uint8_t           ssf_data[4];
  };
```

- ssf_type holds the value SCTP_SEND_FAILED.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Send Failure notification


```
struct sctp_send_failed {
      uint16_t          ssf_type;
      uint16_t          ssf_flags;
      uint32_t          ssf_length;
      uint32_t          ssf_error;
      struct sctp_sndrcvinfo ssf_info;
      sctp_assoc_t      ssf_assoc_id;
      uint8_t           ssf_data[4];
  };
```

- `ssf_flags` is either `SCTP_DATA_UNSENT` to indicate the message was NEVER put on the wire or `SCTP_DATA_SENT` for having been sent but not acknowledged..

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Send Failure notification

Cisco.com

```
struct sctp_send_failed {
      uint16_t            ssf_type;
      uint16_t            ssf_flags;
      uint32_t            ssf_length;
      uint32_t            ssf_error;
      struct sctp_sndrcvinfo ssf_info;
      sctp_assoc_t        ssf_assoc_id;
      uint8_t             ssf_data[4];
  };
```

- ssf_length is set to the size of this structure plus the size of the data message.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Send Failure notification


```
struct sctp_send_failed {
      uint16_t          ssf_type;
      uint16_t          ssf_flags;
      uint32_t          ssf_length;
      uint32_t          ssf_error;
      struct sctp_sndrcvinfo ssf_info;
      sctp_assoc_t      ssf_assoc_id;
      uint8_t           ssf_data[4];
  };
```

- ssf_error field is any error type the stack would like to communicate.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Send Failure notification


```
struct sctp_send_failed {
      uint16_t           ssf_type;
      uint16_t           ssf_flags;
      uint32_t           ssf_length;
      uint32_t           ssf_error;
      struct sctp_sndrcvinfo ssf_info;
      sctp_assoc_t       ssf_assoc_id;
      uint8_t            ssf_data[4];
  };
```

- ssf_info is the sctp_sndrcvinfo structure that was sent with the message (the context is found here).

 © 2003-2006 Cisco Systems, Inc. All rights reserved.

Send Failure notification


```
struct sctp_send_failed {
      uint16_t          ssf_type;
      uint16_t          ssf_flags;
      uint32_t          ssf_length;
      uint32_t          ssf_error;
      struct sctp_sndrcvinfo ssf_info;
      sctp_assoc_t      ssf_assoc_id;
      uint8_t           ssf_data[4];
  };
```

- **ssf_assoc_id is the association that had the failure.**

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Send Failure notification


```
struct sctp_send_failed {
      uint16_t          ssf_type;
      uint16_t          ssf_flags;
      uint32_t          ssf_length;
      uint32_t          ssf_error;
      struct sctp_sndrcvinfo ssf_info;
      sctp_assoc_t      ssf_assoc_id;
      uint8_t           ssf_data[4];
  };
```

- ssf_data is the the actual user message that was attempted to be sent.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Shutdown Event notification

- SCTP_SHUTDOWN_EVENT is received when a graceful shutdown has begun: i.e. the other side did a shutdown and you have just entered the SHUTDOWN-RECEIVED state.
- Its structure is as follows:

```
struct sctp_shutdown_event {
    uint16_t     sse_type;
    uint16_t     sse_flags;
    uint32_t     sse_length;
    sctp_assoc_t sse_assoc_id;
};
```


© 2003-2006 Cisco Systems, Inc. All rights reserved.

Shutdown Event notification


```
struct sctp_shutdown_event {
    uint16_t     sse_type;
    uint16_t     sse_flags;
    uint32_t     sse_length;
    sctp_assoc_t sse_assoc_id;
};
```

- **sse_type field is set to SCTP_SHUTDOWN_EVENT**
- **sse_flags is 0 and sse_length is the size of the shutdown structure.**
- **sse_assoc_id is the association id that is being shutdown.**

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Adaption Indication notification

- The SCTP_ADAPTION_INDICATION is only received if the peer is providing an adaption indication. The indication is used to communicate a 32-bit indicator that the peer set to be sent on association setup.
- The structure sent is as follows:

```
struct sctp_adaptation_event {
    uint16_t     sai_type;
    uint16_t     sai_flags;
    uint32_t     sai_length;
    uint32_t     sai_adaption_ind;
    sctp_assoc_t sai_assoc_id;
};
```


© 2003-2006 Cisco Systems, Inc. All rights reserved.

Adaption Indication notification

Cisco.com

```
struct sctp_adaptation_event {
    uint16_t      sai_type;
    uint16_t      sai_flags;
    uint32_t      sai_length;
    uint32_t      sai_adaption_ind;
    sctp_assoc_t sai_assoc_id;
};
```

- **sai_type** is set to SCTP_ADAPTION_INDICATION.
- **sai_flags** is set to 0 and **sai_length** is the size of this structure.
- **sai_adaptation_ind** is the 32 bit indicator
- **sai_assoc_id** can be used to identify the association for this notification

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Partial Delivery Event notification

Cisco.com

- The SCTP_PARTIAL_DELIVERY_EVENT notification is used to notify an application that something has transpired in the current data being partially delivered (normally its been aborted).
- The structure looks as follows:

```
struct sctp_pdapi_event {
    uint16_t     pdapi_type;
    uint16_t     pdapi_flags;
    uint32_t     pdapi_length;
    uint32_t     pdapi_indication;
    sctp_assoc_t pdapi_assoc_id;
};
```


© 2003-2006 Cisco Systems, Inc. All rights reserved.

Partial Delivery Event notification


```
struct sctp_pdapi_event {
    uint16_t      pdapi_type;
    uint16_t      pdapi_flags;
    uint32_t      pdapi_length;
    uint32_t      pdapi_indication;
    sctp_assoc_t pdapi_assoc_id;
};
```

- **pdapi_type is set to SCTP_PARTIAL_DELIVERY_EVENT**
- **pdapi_flags is 0 and pdapi_length is the size of this structure.**
- **pdapi_assoc_id is the association id with the event.**
- **pdapi_indication is most likely set to SCTP_PARTIAL_DELIVERY_ABORTED.**

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Authentication Event notification

Cisco.com

- The SCTP_AUTHENTICATION_EVENT notification is used to notify an application that a change as occurred in respect to the authentication state.
- The structure looks as follows:

```
struct sctp_authkey_event {
        uint16_t auth_type;
        uint16_t auth_flags;
        uint32_t auth_length;
        uint16_t auth_keynumber;
        uint16_t auth_altkeynumber;
        uint32_t auth_indication;
        sctp_assoc_t auth_assoc_id;
    };
```


© 2003-2006 Cisco Systems, Inc. All rights reserved.

Authentication Event notification


```
struct sctp_authkey_event {
        uint16_t auth_type;
        uint16_t auth_flags;
        uint32_t auth_length;
        uint16_t auth_keynumber;
        uint16_t auth_altkeynumber;
        uint32_t auth_indication;
        sctp_assoc_t auth_assoc_id;
    };
```

- auth_type is set to SCTP_AUTHENTICATION_EVENT.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Authentication Event notification


```
struct sctp_authkey_event {
        uint16_t auth_type;
        uint16_t auth_flags;
        uint32_t auth_length;
        uint16_t auth_keynumber;
        uint16_t auth_altkeynumber;
        uint32_t auth_indication;
        sctp_assoc_t auth_assoc_id;
    };
```

- auth_flags is set to 0, and auth_length is the size of this structure.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Authentication Event notification


```
struct sctp_authkey_event {
        uint16_t auth_type;
        uint16_t auth_flags;
        uint32_t auth_length;
        uint16_t auth_keynumber;
        uint16_t auth_altkeynumber;
        uint32_t auth_indication;
        sctp_assoc_t auth_assoc_id;
    };
```

- **auth_keynumber is the key number that this notification is about (note an event may be about more than one key).**

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Authentication Event notification


```
struct sctp_authkey_event {
        uint16_t auth_type;
        uint16_t auth_flags;
        uint32_t auth_length;
        uint16_t auth_keynumber;
        uint16_t auth_altkeynumber;
        uint32_t auth_indication;
        sctp_assoc_t auth_assoc_id;
    };
```

- **auth_altkeynumber is an additional key number used by some notifications (future?).**

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Authentication Event notification


```
struct sctp_authkey_event {
        uint16_t auth_type;
        uint16_t auth_flags;
        uint32_t auth_length;
        uint16_t auth_keynumber;
        uint16_t auth_altkeynumber;
        uint32_t auth_indication;
        sctp_assoc_t auth_assoc_id;
};
```

- auth_indication tells you what type of event as transpired, currently the only defined value is SCTP_AUTH_NEWKEY which indicates the auth_keynumber field holds the newly set active key (the key number is provided by the user).

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Authentication Event notification


```
struct sctp_authkey_event {
        uint16_t auth_type;
        uint16_t auth_flags;
        uint32_t auth_length;
        uint16_t auth_keynumber;
        uint16_t auth_altkeynumber;
        uint32_t auth_indication;
        sctp_assoc_t auth_assoc_id;
    };
```

- **auth_assoc_id is the association id with the event.**

© 2003-2006 Cisco Systems, Inc. All rights reserved.

How Do We Get Notifications?

Cisco.com

- Now that you have seen the types of events how do you subscribe to these?
- You use a socket option SCTP_EVENTS passing the structure:

```
struct sctp_event_subscribe {
    uint8_t sctp_data_io_event;
    uint8_t sctp_association_event;
    uint8_t sctp_address_event;
    uint8_t sctp_send_failure_event;
    uint8_t sctp_peer_error_event;
    uint8_t sctp_shutdown_event;
    uint8_t sctp_partial_delivery_event;
    uint8_t sctp_adaptation_layer_event;
    uint8_t sctp_authentication_event;
};
```


© 2003-2006 Cisco Systems, Inc. All rights reserved.

One other thing: Data I/O events

- Besides notifications the event structure also lets you subscribe to a SCTP_SNDRCV (aka. data_io) event.
- This is how you get the sctp_sndrcvinfo structure.
- If you do not subscribe to this, you may NOT receive this information on each read.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Subscribing to Notifications

- **Each uint8_t in the structure represents a boolean value that is set to 1 to indicate you wish the notifications. It set to 0 to indicate that you do NOT want the event.**
- **So to set the event structure event one would setup the structure and do:**

```
setsockopt(sd, IPPROTO_SCTP, SCTP_EVENTS,
           &event, sizeof(event));
```


© 2003-2006 Cisco Systems, Inc. All rights reserved.

Identifying a Notification vs. User Data

- Remember the sctp_recvmsg() call?
- The last argument was a int *msg_flags
- This is filled in with the same return value as the msg_flags in the recvmsg() structure.
- The flag may contain MSG_NOTIFICATION. This indicates that the data is NOT data but a notification.
- It would also have MSG_EOR or'ed in with it (since these are delivered as a complete message.
- So now lets look at a snippet of code that will identify and print a notification.

© 2003-2006 Cisco Systems, Inc. All rights reserved.

Example 2 - main


```
main()
{
        int sd,ret;
        char buffer[100];
        struct sockaddr_in6 sin6;
        struct sctp_event_subscribe event;
        socklen_t fromlen;
        sd = socket(PF_INET6, SOCK_SEQPACKET, IPPROTO_SCTP);
        if(sd == -1) {
                printf("Could not open socket error:%d\n", errno);
                exit(0);
        }
        memset(&sin6, 0, sizeof(sin6));
```


© 2003-2006 Cisco Systems, Inc. All rights reserved.

Example 2 - main


```
sin6.sin6_len = sizeof(sin6);
sin6.sin6_family = AF_INET6;
sin6.sin6_port = htons(2222);
fromlen = sizeof(sin6);
if(bind(sd, (struct sockaddr *)&sin6, fromlen) == -1) {
        printf("Can't bind port error:%d\n", errno);
        exit(0);
}
if(listen(sd, 1) == -1) {
        printf("Can't listen error:%d\n", errno);
        exit(0);
}
memset(&event, 0, sizeof(event));
```


© 2003-2006 Cisco Systems, Inc. All rights reserved.

Example 2 - main


```
        event.sctp_data_io_event = 1;
        event.sctp_association_event = 1;
        event.sctp_address_event = 1;
        event.sctp_send_failure_event = 1;
        event.sctp_peer_error_event = 1;
        event.sctp_shutdown_event = 1;
        event.sctp_partial_delivery_event = 1;
        event.sctp_adaptation_layer_event = 1;
        if(setsockopt(sd, IPPROTO_SCTP, SCTP_EVENTS, &event,
sizeof(event)) != 0) {
                printf("Can't do SET_EVENTS socket option! err:%d\n", errno);
                exit(0);
        }
```


© 2003-2006 Cisco Systems, Inc. All rights reserved.

Example 2 - main


```
again:
        ret = sctpReadInput(sd);
        if(ret) {
                printf("Got %d bytes of data.. want notify\n", ret);
                goto again;
        }
}
```


© 2003-2006 Cisco Systems, Inc. All rights reserved.

Example 2 - sctpReadInput


```
int
sctpReadInput(int sd)
{
          /* receive some number of datagrams and
           * act on them.
           */
          struct sctp_sndrcvinfo s_info;
          int sz, msg_flags=0, len;
          struct sockaddr_storage from;
          socklen_t flen;
          char readBuffer[65535];
          len = sizeof(readBuffer);
          flen = sizeof(from);
```


© 2003-2006 Cisco Systems, Inc. All rights reserved.

Example 2 - sctpReadInput

Cisco.com

```
        sz = sctp_recvmsg (sd,
                           readBuffer,
                           (size_t)len,
                           (struct sockaddr *)&from,
                           &flen,
                           &s_info,
                           &msg_flags);
        if (msg_flags & MSG_NOTIFICATION) {
                handle_notification(readBuffer);
                return(0);
        }
        return(sz);
}
```


© 2003-2006 Cisco Systems, Inc. All rights reserved.

Example 2 – handle_notification


```
void
handle_notification(char *buf)
{
        union sctp_notification *notify;
        notify = (union sctp_notification *)buf;
        switch(notify->sn_header.sn_type) {
        case SCTP_ASSOC_CHANGE:
                printf("Got a assoc change: %d\n",
                        notify->sn_assoc_change.sac_state);
                break;
        case SCTP_PEER_ADDR_CHANGE:
                printf("Got a peer address change: %d\n",
                        notify->sn_paddr_change.spc_state);
                break;
```


© 2003-2006 Cisco Systems, Inc. All rights reserved.

Example 2 – handle_notification

Cisco.com

```
case SCTP_REMOTE_ERROR:
        printf("Got a remote error: %d\n",
            notify->sn_remote_error.sre_error);
        break;
case SCTP_SEND_FAILED:
        printf("Got a send failed type %s\n",
            ((notify->sn_send_failed.ssf_flags ==
              SCTP_DATA_UNSENT) ?
                "Data unsent" : "Data Sent but un-acked"));
        break;
case SCTP_SHUTDOWN_EVENT:
        printf("Got a shutdown event assoc:%x\n",
            notify->sn_shutdown_event.sse_assoc_id);
        break;
```


© 2003-2006 Cisco Systems, Inc. All rights reserved.

Example 2 – handle_notification


```
case SCTP_ADAPTION_INDICATION:
        printf("Adaption indication %x\n",
            notify->sn_adaption_event.sai_adaption_ind);
        break;
case SCTP_PARTIAL_DELIVERY_EVENT:
        printf("Partial delivery event aborted?\n");
        break;
case SCTP_AUTHENTICATION_EVENT:
        printf(“Authentication event\n”);
        break;
}
```


© 2003-2006 Cisco Systems, Inc. All rights reserved.