EBU Workshop, 09/24/2014 Geneva

Developing for SSM and AMT

Duanpei Wu, duanpei@cisco.com

Toerless Eckert, eckert@cisco.com

Content

- **AMT Relay Setup and Check – with ASR 1k**
  - Check Router Configuration
  - Use 4 programs developed based on AMT GW library
- **AMT Gateway Library**
  - What is implemented and what is not yet implemented
  - Lib Architecture
  - Detail APIs
- **Test and Sample Code**
  - 4 examples

Project background and where to get source code?

- There are several AMT related projects in Cisco. Open source ***amt_gw_lib*** is an effort under one of these projects.

- The publicly accessible version at publication of these slides is 20140920

- The source code could be retrieved from
  - https://github.com/cisco/SSMAMTtools.git
    -- If you have a Linux machine, you may get the code with
    git clone https://github.com/cisco/SSMAMTtools.git

- This deck of slides can be obtained from
  -- …/[amt_ge_lib]/docs/amt-gw-lib.pdf

Reminder – IP Multicast API(s) before our library

ASM – Any Source Multicast
Traditional IP Multicast Service model. 1990, RFC1112:

- Sources simply send traffic to a host-group <group-address>,
  They do not know who the receivers are.
  But the application needs to have <group-addresses> allocated for its use!

```
sendto(sock, packet, …, <group-address>,…)
```

- Receivers join a host-group G

```
setsockopt(sock,…, MCAST_JOIN/LEAVE_GROUP,
                   (<interface>, <group-adress>), …)
```

- …and then receive traffic from all sources sending to G.

```
recvfrom(sock, &packet, … &source, …)
```

- *Receivers not need to know who the sources are.*
  …but may learn once they receive the packets of course.

SSM– Source Specific Multicast
Introduced ca. 2000. RFC4607 (2006)

- Sources simply send traffic to a host-group <group-address>,
  Code *unchanged from ASM*
  But no need to allocate the <group-address> (pick randomn one).

```
sendto(sock, packet, …, <group-address>,…)
```

- Receivers *subscribe* to (S,G) channel

```
setsockopt(sock,…, MCAST_JOIN/LEAVE_SOURCE_GROUP,
(<interface>, <source-address>, <group-adress>), …)
```

- …and then receive traffic from channel “S sending to G”.

```
recvfrom(sock, &packet, … &source, …)
```

- *Receivers need to know the source(s) IN ADVANCE*!

More details – sender socket

- socket(,UDP,)
- bind(,{local-IP-address }, source-UDP-Port,)
  - If local-IP-address is specified, packets will be sent out from the local IP interface with this address. If local-IP-address is not specified (0.0.0.0), the packet will be sent out on the interface selected by the hosts routing table. setsockopt(,IP_MULTICAST_TTL,)
  - By default Multicast packets are sent with TTL=1 !!!!!!!!
- Sending:

  connect(multicast-group-address, destination-UDP-port)

  send(data,datalength)
- Or

  sendto({multicast-group-address,destination-UDP-port), data, datalength)

More details – receiver socket

- socket(, UDP,)
- setsockopt(,Group or (Source,Group), interface-IP-address, “JOIN/LEAVE”)
  - Interface-IP-address determines on which interface the IGMP/MLD membership will be sent – and therefore the IP multicast traffic received.
  - If not specified, the interface is determined by the hosts routing table
- setsockopt(,SO_REUSEADDR,)
  - Allows multiple apps on same host receive same traffic (eg: Vio receiver app and separate quality monitoring app).
- bind(,{multicast-group-address}, destination-UDP-Port)
  - You must specify the destination-UDP-Port. If you do not specify the multicast-group-address, then you would also receive unicast packets for the same destination-UDP-port
- Optional “connect(…)”
- Recv/recvfrom({sender-ip-address,source-UDP-port), data, datalen)

Planned steps

1. Build up “operator” experience

   Understand how to step by step set up a testbed

   Understand what “app” to run to test the setp

   Understand how to troubleshoot on the router

   Software ? … black box… no idea how it works internally (client, gateway library)

- Software development

  Explain / overview of how library works

  Repeat build-up steps

  Now look at what the demo apps using the library do / source-code

- Integration experiments of library into your own applications

  Free form.. Pending on time available.

AMT Relay Setup and Check – with ASR 1k

Testbed example

- Incrementally build/view/experience the steps in native SSM and AMT traffic flows
  - Network path delivering native SSM to the receiver application
  - Network path not supporting native SSM to receiver application – must use AMT
  - Connect to either paths to validate how SSM+AMT gateway library in app will automatically choose.

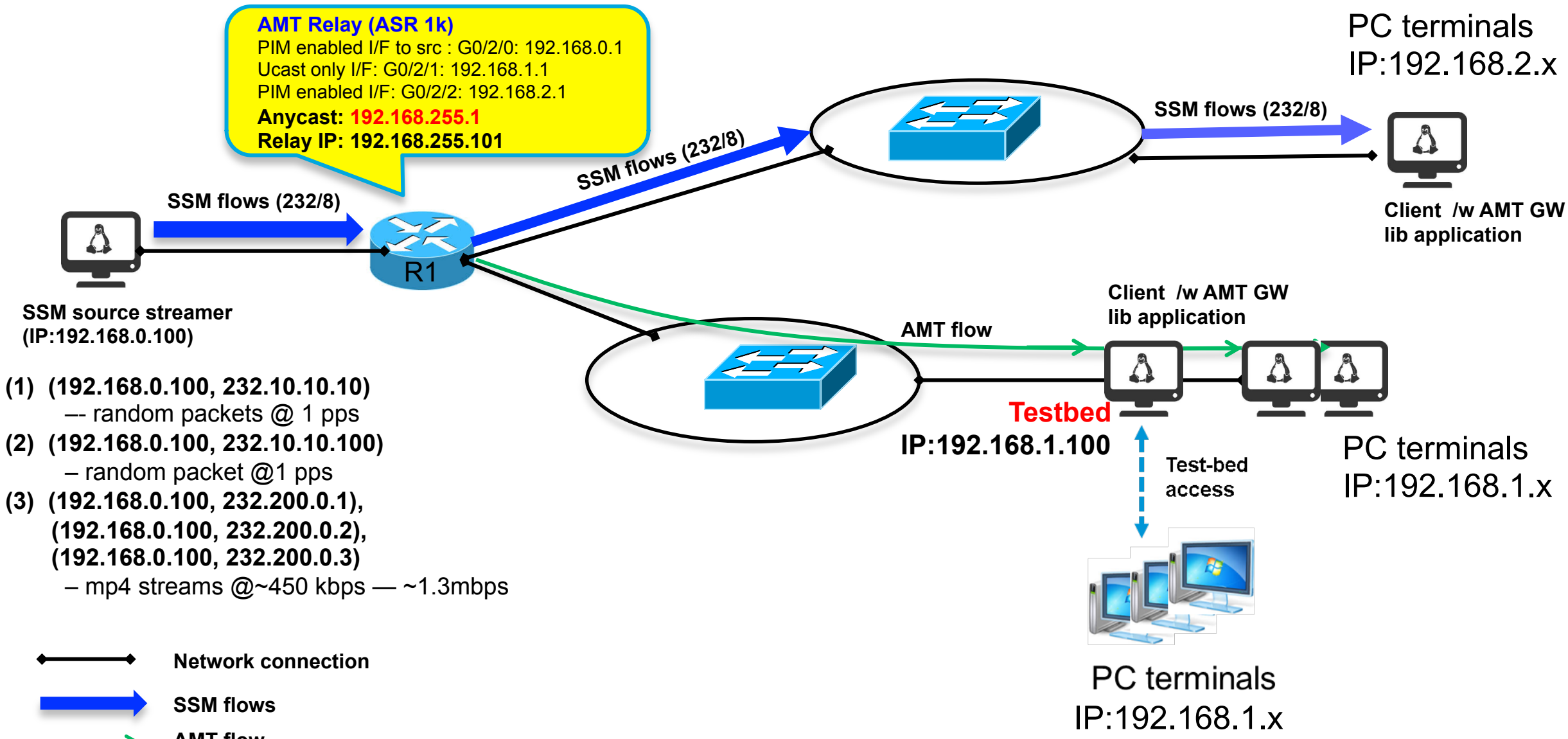
- Think of this testbed as a starting point for your own development testbed.

- **If you use your own Linux PC, the sample code is under**
  - ***…/amt_gw_lib_11/examples****, assume you have cloned the code in directory …/amt_gw_lib_11/*
  - Build lib

  > cd build/linux

  - *Make*

Testbed AMT Environment



(1) (192.168.0.100, 232.10.10.10)
–- random packets @ 1 pps

(2) (192.168.0.100, 232.10.10.100)
– random packet @1 pps

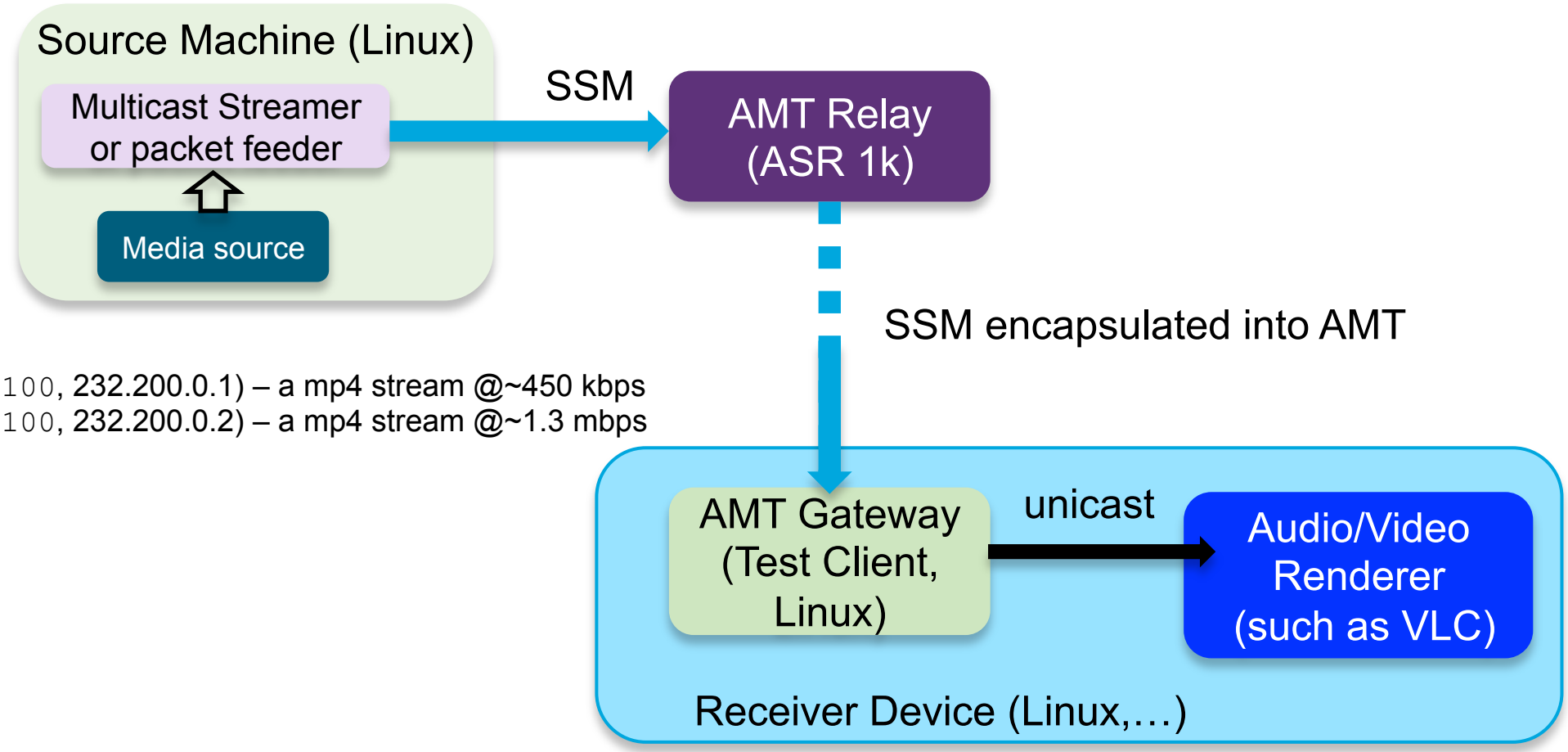
(3) (192.168.0.100, 232.200.0.1),
(192.168.0.100, 232.200.0.2),
(192.168.0.100, 232.200.0.3)
– mp4 streams @~450 kbps — ~1.3mbps

Network connection

SSM flows

AMT flow

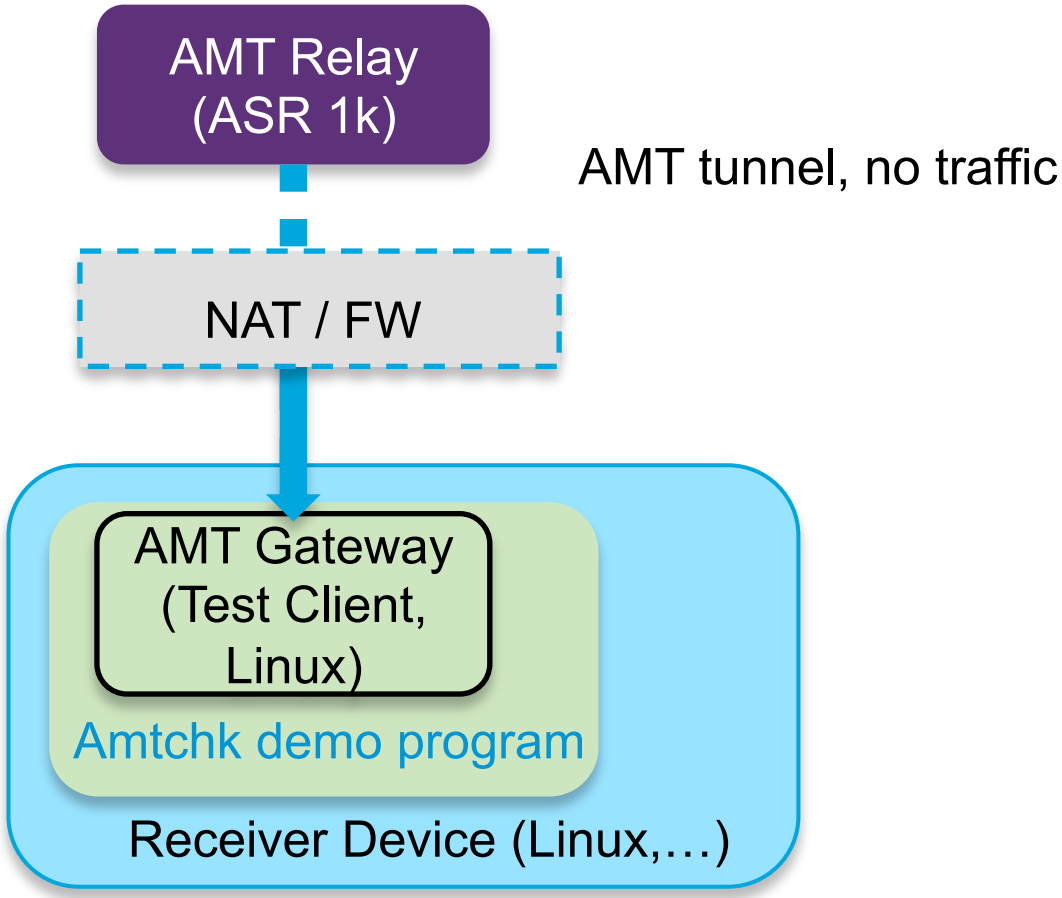
Lets abstract the workflow – for AMT case



(1) (192.168.0.100, 232.200.0.1) – a mp4 stream @~450 kbps
(2) (192.168.0.100, 232.200.0.2) – a mp4 stream @~1.3 mbps

1.Minimal starting point – relay discovery

- No SSM/IP-Multicast source!
- Configure / explain AMT-Relay
- Start minimal demo application – amtchck
- Will test a large part of relay config:
- Receiver devices needs to be able to reach AMT-Relay anycast address.
- EX1 app will do initial part of AMT protocol:
  - Send AMT Relay-Discovery to anycast address
  - Displays AMT unicast Relay address returned by AMT-relay in AMT message



AMT Relay Configuration

```
ip multicast-routing
ip pim ssm default

interface GigabitEthernet0/0/0
  description Multicast source input interface
  ip address 192.168.0.1 255.255.255.0
  ip pim sparse-mode

!
interface GigabitEthernet0/0/1
  description Unicast only network – AMT output
  ip address 192.168.1.1 255.255.255.0

interface Loopback0
  descr Relay Anycast address (same on multiple relays)
  ip address 192.168.255.1 255.255.255.255!

interface Loopback1
  descr Relay unicast address – unique on this router
  ip address 192.168.255.101 255.255.255.255
  ip pim sparse-mode ! May be redundant
```

```
! Not necessary in testbed, but for explanation:
! Announce addresses into IGP (OSPG)
router ospf 100
   network 192.168.255.1      0.0.0.0 area 100
   network 192.168.255.101    0.0.0.0 area 100
```

```
interface Tunnel11
  description AMT Relay Tunnel
  no ip address
  ip pim passive
  ip igmp version 3
  tunnel source Loopback1
  tunnel mode udp multipoint
  amt relay traffic ip
```

AMT Relay Check: anycast IP reachable

- Check if anycast address is reachable by using “ping”

```
THKERNEN-M-V02F:~ tkernen$ ping 192.168.255.1
PING 192.168.255.1 (192.168.255.1): 56 data bytes
64 bytes from 192.168.255.1: icmp_seq=0 ttl=255 time=0.286 ms
64 bytes from 192.168.255.1: icmp_seq=1 ttl=255 time=0.386 ms
64 bytes from 192.168.255.1: icmp_seq=2 ttl=255 time=0.298 ms
64 bytes from 192.168.255.1: icmp_seq=3 ttl=255 time=0.286 ms
64 bytes from 192.168.255.1: icmp_seq=4 ttl=255 time=0.273 ms
64 bytes from 192.168.255.1: icmp_seq=5 ttl=255 time=0.279 ms
64 bytes from 192.168.255.1: icmp_seq=6 ttl=255 time=0.264 ms
64 bytes from 192.168.255.1: icmp_seq=7 ttl=255 time=0.283 ms
64 bytes from 192.168.255.1: icmp_seq=8 ttl=255 time=0.275 ms
64 bytes from 192.168.255.1: icmp_seq=9 ttl=255 time=0.281 ms
64 bytes from 192.168.255.1: icmp_seq=10 ttl=255 time=0.297 ms
64 bytes from 192.168.255.1: icmp_seq=11 ttl=255 time=0.270 ms
64 bytes from 192.168.255.1: icmp_seq=12 ttl=255 time=0.284 ms
64 bytes from 192.168.255.1: icmp_seq=13 ttl=255 time=0.270 ms
64 bytes from 192.168.255.1: icmp_seq=14 ttl=255 time=0.298 ms
```

AMT Relay Check: relay IP

- Check if the relay returns the relay IP – tests part of AMT

amtchk is a program developed based on the amt_gw_lib presented in this workshop. It sets up a AMT channel between the client and the relay using the provided anycast IP. The source code is in …/[amt_gw_11]/test/amtchk.c

Relay Operations Explained

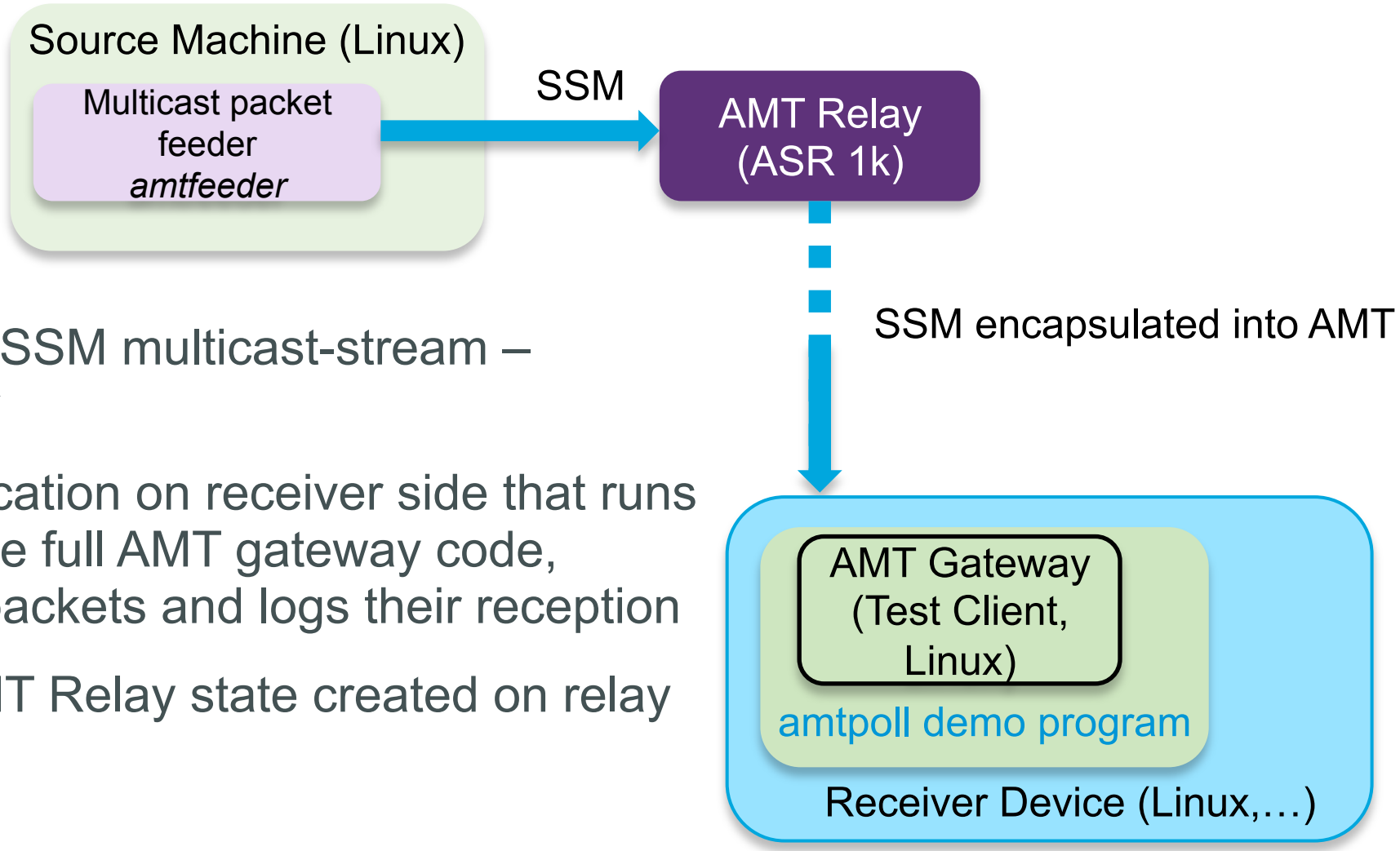
- In IOS-XE, Anycast loopback for AMT work like any other Anycast uses in IP multicast
  - No special tricks for AMT

- Relevant packet flow
  - AMT-Gateway sends AMT-Relay-Discovery to Relays Anycast-IP address
  - AMT-relay (ASR1k/CSR1kv) receives AMT-Relay-Discovery. Creates AMT-Relay-Advertisement reply message containing the Relay-IP-Address.
  - Relay-IP-address is the “tunnel source <ip-address>” of the AMT relay tunnel interface
  - There can only be one AMT relay tunnel interface

- If operator manually shuts down “interface loopback 2” (Anycast), additional AMT-gateways would get AMT-relay service from another router configured for the same Anycast address. Existing AMT-gateways would stay using this router.

- In IOS-XR (high end routers), we also implement a key trick (option in AMT spec):
  - AMT-relay count active number of connected AMT-relays
  - If number exceeds configured threshold, Anycast address gets withdrawn

2. Check IP multicast through AMT tunnel - synthetic



- Generate SSM multicast-stream – amtfeeder
- Run application on receiver side that runs through the full AMT gateway code, receives packets and logs their reception
- Check AMT Relay state created on relay

Start synthetic traffic - amtfeeder

```
bram@testengine2:~/amt_gw_lib_11/test/linux$ ./amtfeeder -if em1 -ssm 232.10.10.10 -v
feed to: ssm (source IP=192.168.1.100 group IP=232.10.10.10 ) dstport=9010 len=64 interval=1000 (ms) srcMedia=<generated>

pkt ssrc=0x327b23c6 remote:0xe80a0a0a len=64 seq #000000:_AMT..............................  !"#$%&'()*+,-./
pkt ssrc=0x327b23c6 remote:0xe80a0a0a len=64 seq #000482:_AMT.....................................
pkt ssrc=0x327b23c6 remote:0xe80a0a0a len=64 seq #000964:_AMT.....................................
pkt ssrc=0x327b23c6 remote:0xe80a0a0a len=64 seq #001446:_AMT.....................................
pkt ssrc=0x327b23c6 remote:0xe80a0a0a len=64 seq #001928:_AMT.....................................
pkt ssrc=0x327b23c6 remote:0xe80a0a0a len=64 seq #002409:_AMTijklmnopqrstuvwxyz{|}~...............
pkt ssrc=0x327b23c6 remote:0xe80a0a0a len=64 seq #002891:_AMTKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz
pkt ssrc=0x327b23c6 remote:0xe80a0a0a len=64 seq #003373:_AMT-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\
pkt ssrc=0x327b23c6 remote:0xe80a0a0a len=64 seq #003855:_AMT.................  !"#$%&'()*+,-./0123456789:;<=>
pkt ssrc=0x327b23c6 remote:0xe80a0a0a len=64 seq #004337:_AMT.....................................
pkt ssrc=0x327b23c6 remote:0xe80a0a0a len=64 seq #004819:_AMT.....................................
pkt ssrc=0x327b23c6 remote:0xe80a0a0a len=64 seq #005301:_AMT.....................................
pkt ssrc=0x327b23c6 remote:0xe80a0a0a len=64 seq #005783:_AMT.....................................
```

amtfeeder is a program developed based on the amt_gw_lib presented in this workshop. It generates a sequence of packets or read packets from a file and send them out to a SSM. The source code is in …/[amt_gw_10]/test/amtfeeder.c

In this example "–if" to specify the interface you output traffic to (e.g.: em1) sending to 232.10.10.10 and a verbose output "-v"

Start receiver application

```
bram@testengine2:~/amt_gw_lib_11/test/linux$ ./amtpoll -anycast 192.168.255.1 -src 192.168.0.100 -ssm 232.10.10.10 -v
getLocalIPDev:get hostname failed
recive from: amt:192.168.255.1 (source IP=192.168.0.100 group IP=232.10.10.10 ) udp port=9010
0: pkt len=64: 80 7e 33 87 28 a8 38 3c 32 7b 23 c6 5f 41 4d 54  .~3.(.8<2{#._AMT
1001: pkt len=64: 80 7e 35 69 28 c0 7d 84 32 7b 23 c6 5f 41 4d 54  .~5i(.}.2{#._AMTijklmnopqrstuvwx
2001: pkt len=64: 80 7e 37 4a 28 d8 b5 e8 32 7b 23 c6 5f 41 4d 54  .~7J(...2{#._AMTJKLMNOPQRSTUVWXY
3000: pkt len=64: 80 7e 39 2b 28 f0 ee 4c 32 7b 23 c6 5f 41 4d 54  .~9+(..L2{#._AMT+,-./0123456789:
4000: pkt len=64: 80 7e 3b 0c 29 09 26 b0 32 7b 23 c6 5f 41 4d 54  .~;.).&.2{#._AMT
5000: pkt len=64: 80 7e 3c ed 29 21 5f 14 32 7b 23 c6 5f 41 4d 54  .~<.)!_.2{#._AMT
6001: pkt len=64: 80 7e 3e cf 29 39 a4 5c 32 7b 23 c6 5f 41 4d 54  .~>.)9.\2{#._AMT
7001: pkt len=64: 80 7e 40 b0 29 51 dc c0 32 7b 23 c6 5f 41 4d 54  .~@.)Q..2{#._AMT
8000: pkt len=64: 80 7e 42 91 29 6a 15 24 32 7b 23 c6 5f 41 4d 54  .~B.)j.$2{#._AMT
9001: pkt len=64: 80 7e 44 73 29 82 5a 6c 32 7b 23 c6 5f 41 4d 54  .~Ds).Zl2{#._AMTstuvwxyz{|}~....
10001: pkt len=64: 80 7e 46 54 29 9a 92 d0 32 7b 23 c6 5f 41 4d 54  .~FT)...2{#._AMTTUVWXYZ[\]^_`abc
11000: pkt len=64: 80 7e 48 36 29 b2 d8 18 32 7b 23 c6 5f 41 4d 54  .~H6)...2{#._AMT6789:;<=>?@ABCDE
12000: pkt len=64: 80 7e 4a 1c 29 cb 50 f0 32 7b 23 c6 5f 41 4d 54  .~J.).P.2{#._AMT.... !"#$%&'()*+
```

amtpoll is a program developed based on the amt_gw_lib presented in this workshop. It subscribes a AMT channel and prints out the packet received. The source code is in …/[amt_gw_11]/test/amtpoll.c

In this example: Anycast address = 192.168.255.1, multicast source is 192.168.0.100, multicast group is 232.10.10.10

Validate state on AMT relay

- **show ip amt tunnel**
  - Shows active AMT gateways connected
    Outgoing interfaces with AMT are AMT gateways identified by client-IP, UDP-Port
  - AMT Relay tunnel:
    Local address UDP port
    192.168.255.101 2268 (0x8DC)
    Remote address Expire time
    192.168.1.2 49164(0xC00C) 00:03:26
    192.168.1.2 49170(0xC012) 00:03:31
    192.168.1.5 49175(0xC017) 00:03:26

- **show ip mroute [source, group] [count]**
  - Shows multicast (S,G) state – Not specific to AMT.
  - With count argument, it will show if packets are flowing

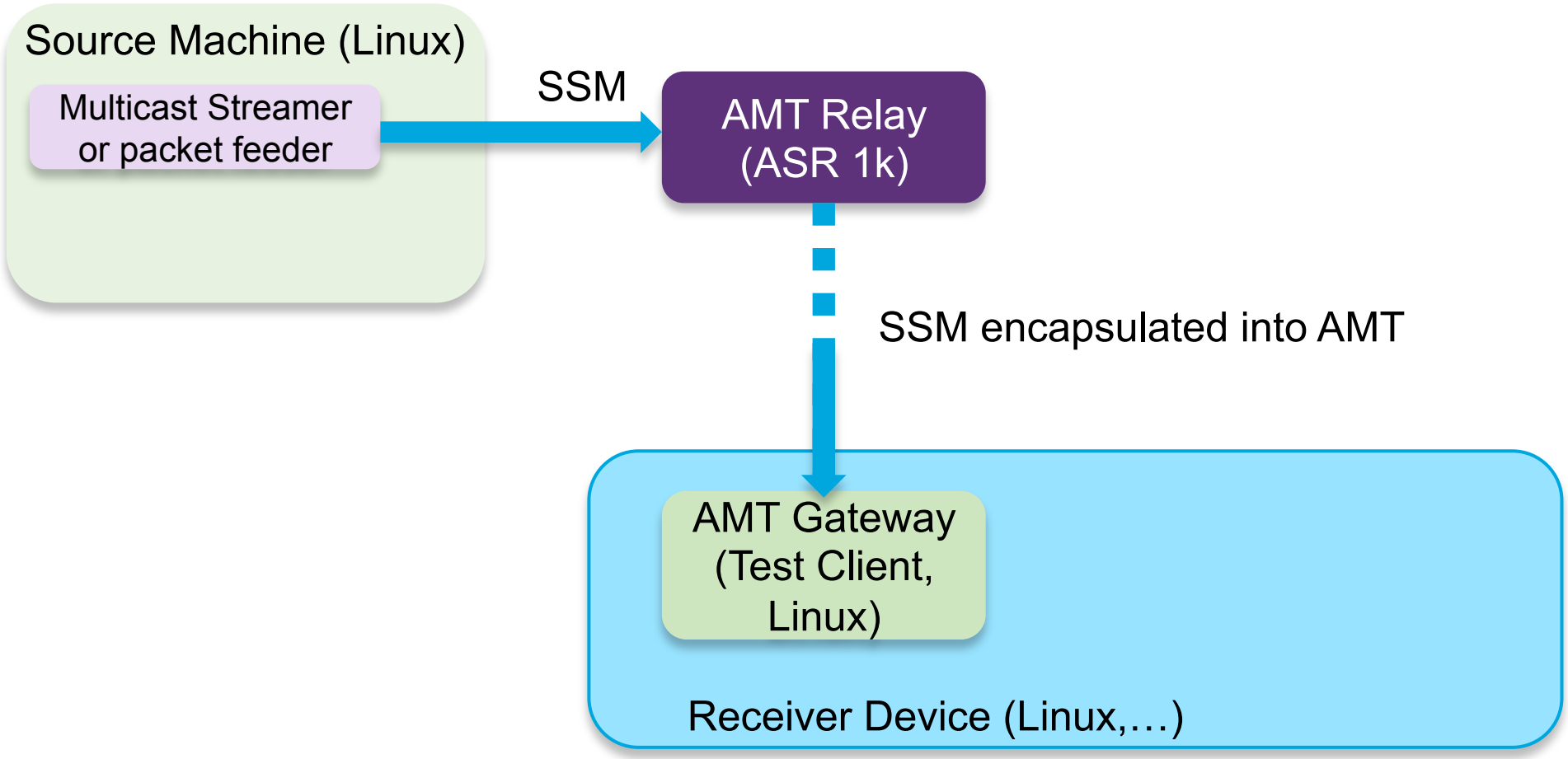
- **debug ip amt relay event**
- **debug ip amt relay packets**
  - Shows when changes/packets happen
  - Usually you need to setup terminal with “term monitor” upfront to see these debug messages

Validate state on AMT relay
More troubleshooting options

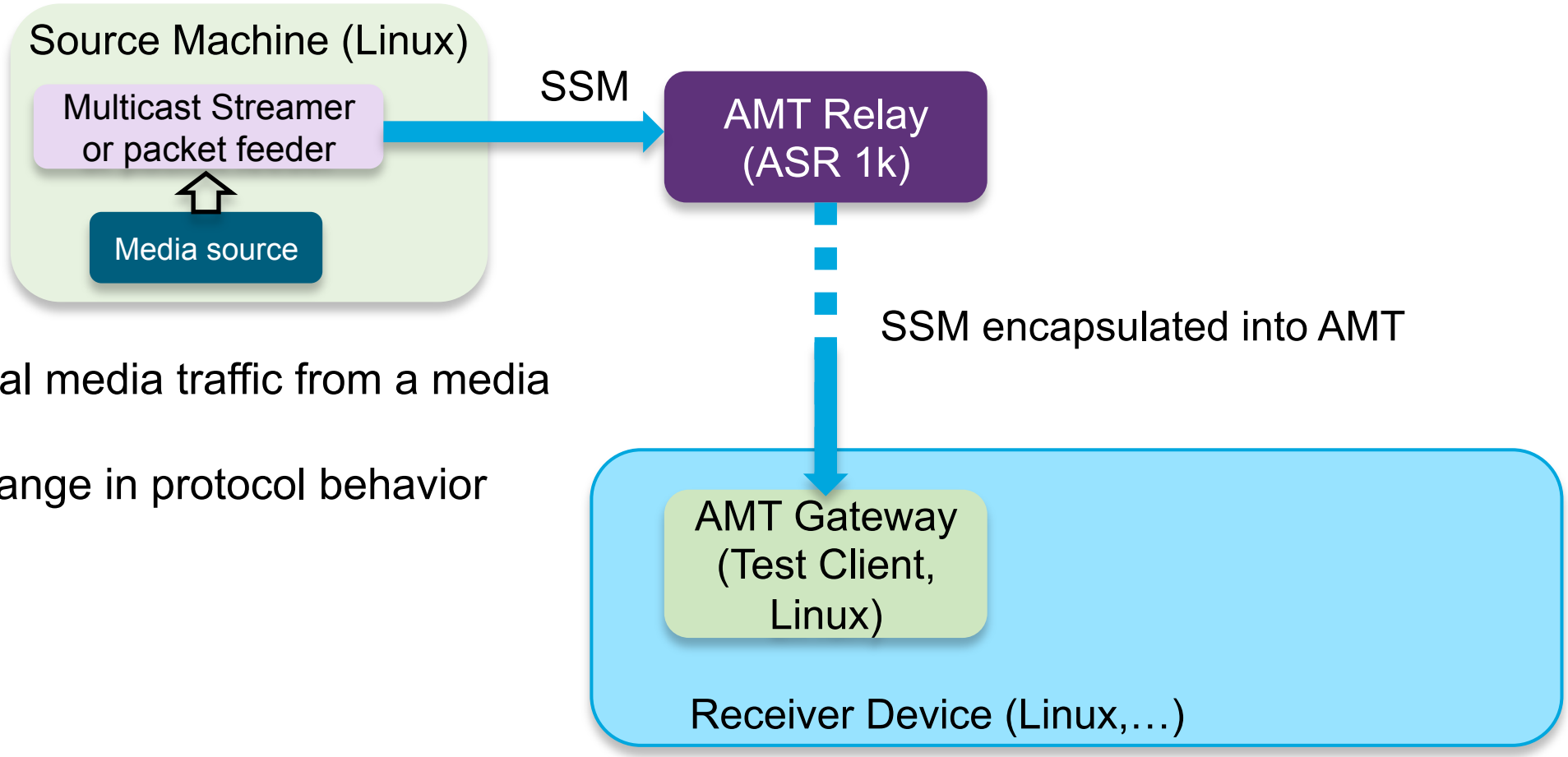
- **show tunnel endpoints**
  - Like “show ip amt tunnel”, but troubleshoots the IOS tunnel software. Only needed when one has to suspect the router has a problem
- **show ip mfib**
  - like “show ip mroute”, but troubleshoots what happens inside the routers packet forwarding functions. Only needed if one has to suspect the router has a problem.

- **“Embedd Packet Capture” (EPC)**
  - http://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-embedded-packet-capture/index.html
  - Capture traffic flows on the router (AMT relay or other), export to your PC, use tools (eg: wireshark) to examine packets.
  - Amt.lua file to decode AMT packets
  - See wireshark directory in AMT library (currently only on Box location).

3. Pass actual traffic via the AMT tunnel

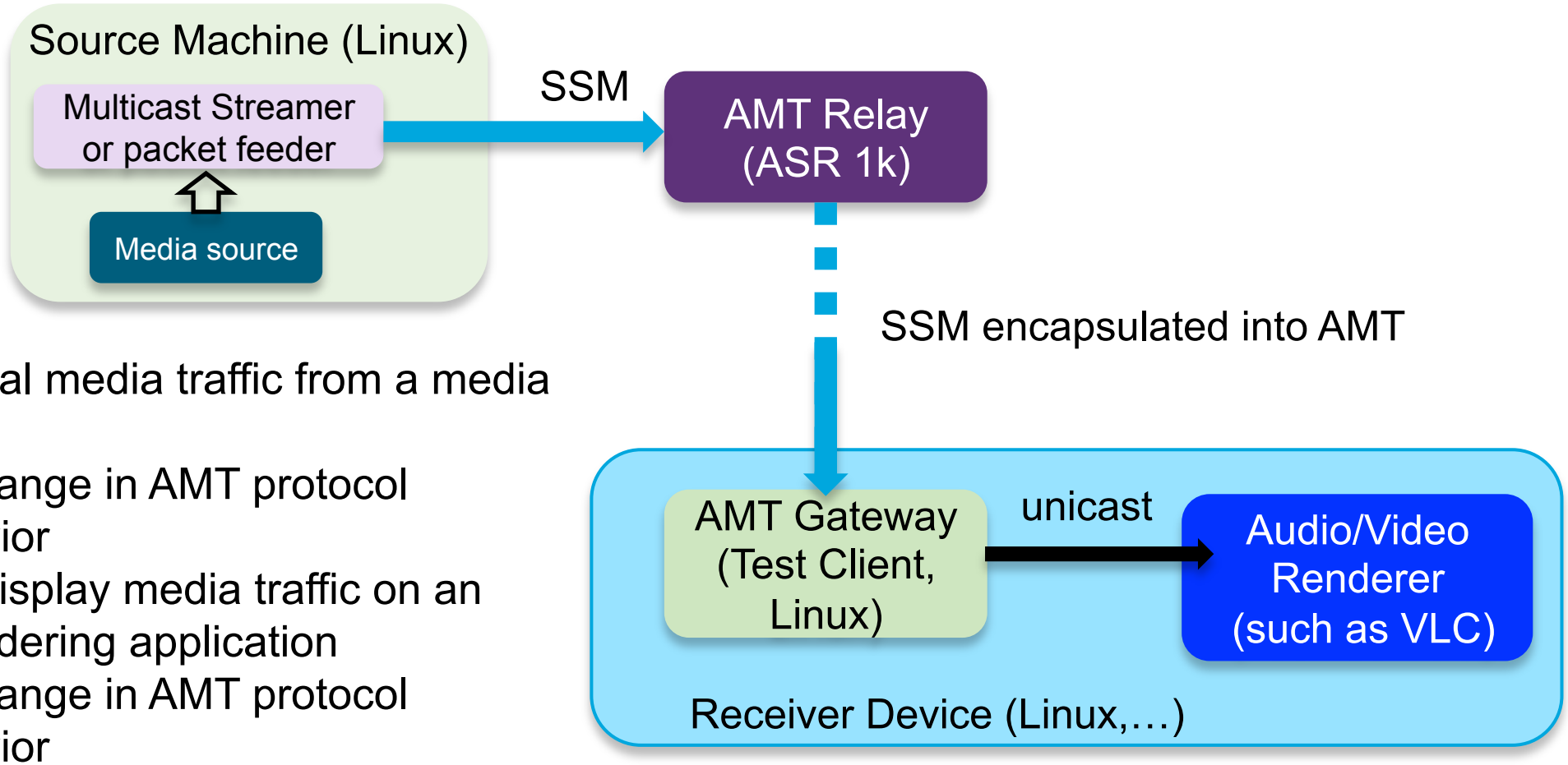


3. Pass actual traffic via the AMT tunnel



- Feed actual media traffic from a media source
  - No change in protocol behavior

3. Pass actual traffic via the AMT tunnel



- Feed actual media traffic from a media source
  - No change in AMT protocol behavior
- Receive/display media traffic on an actual rendering application
  - No change in AMT protocol behavior

AMT Gateway Library

AMT Gateway Protocols Implementation and Status

- **Automatic Multicast Tunneling -- version 17**

Implemented*:*

(1) Six messages: Relay Discovery, Relay Advertisement, Request, Membership Query , Membership Update and multicast data

(2) IPv4

(3) Periodic Three-way handshake : Request ➔Membership Query ➔membership Update

(4) Retry with timing: initial time ➔➔ 2^#iterations

(5) default timeout: as specified in protocols such as 125 seconds for query interval

Not yet implemented at initial client code release:

(1) One message: Teardown (optional gateway message)

(2) IPv6

AMT Gateway Protocols Implementation and Status

- **Internet Group Management Protocol, IGMPv3**

  Implemented: Version 3 for IPv4

- **Multicast Listener Discovery, MLDv2**

  Not yet implemented: Version 2 for IPv6

- **Socket Interface Extensions for Multicast Source Filters**

  Through socket lib,

  setsockopt (…, IP_ADD_SOURCE_MEMBERSHIP, …)

- **Others**

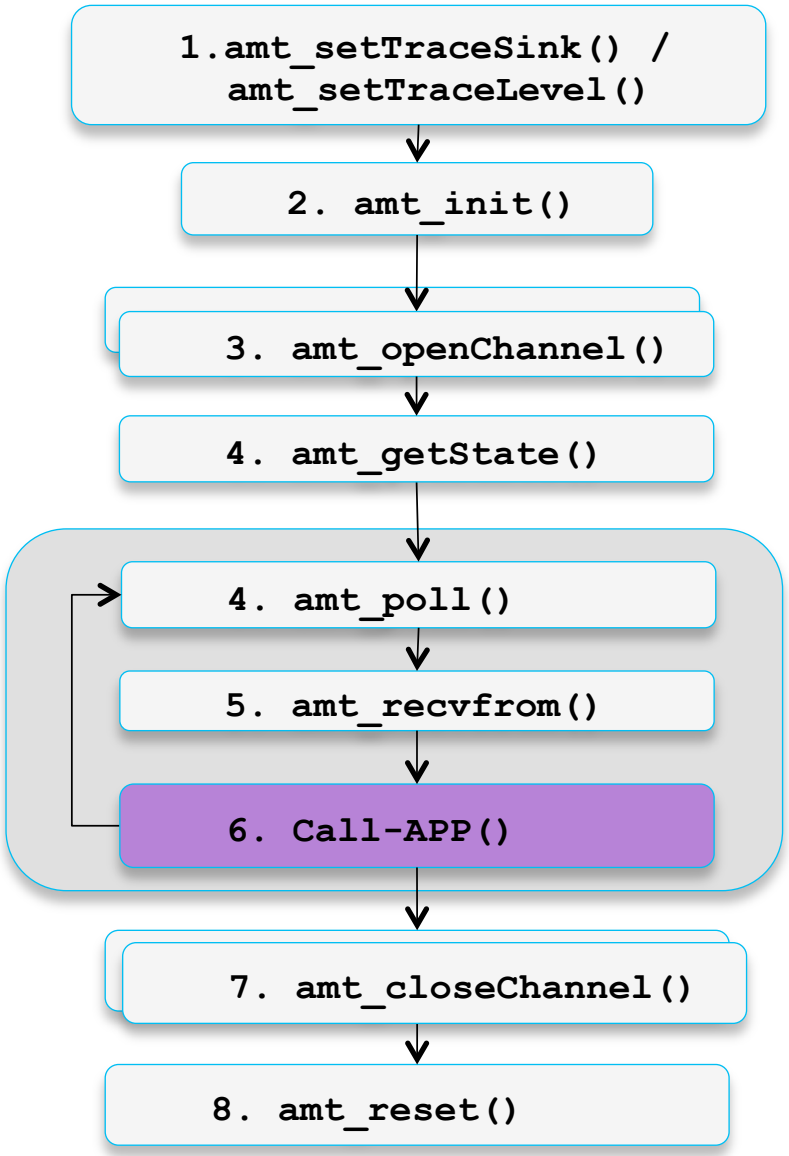
  -- GNU C lib: such as Socket

  -- …

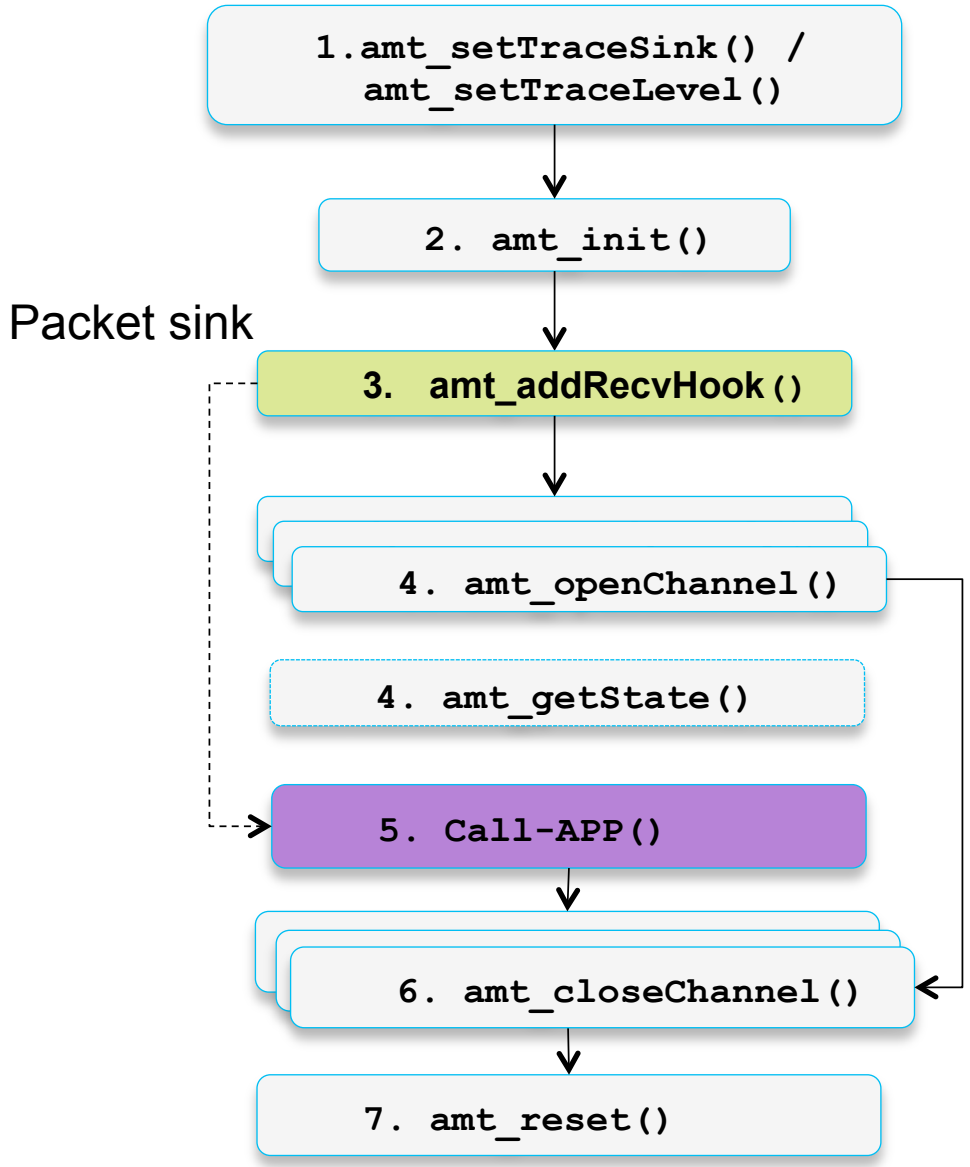
AMT Gateway Library: All APIs

- **Feature APIs**
  - **amt_openChannel()** -- open a channel through ssm or amt with an option to try both and to use whichever gets first connected
  - **amt_closeChannel()** -- close the opened channel
  - **amt_poll()** -- poll the channel status: data-in, data-close, data-err
  - **amt_recvfrom()** -- receive packets (buffered) from the network through AMT/SSM
- **Sink API**
  - **amt_addRecvHook()** -- callback functions to send received packets from AMT/SSM to the App
- **Status Check API**
  - **amt_getState()** -- Get the channel or relay state: joining, joined, etc.
- **Initialization/reset APIs**
  - **amt_init()** -- do initialization (optional)
  - **amt_reset()** -- reset the module to clear up resources and set the module to the initial state
- **Trace APIs**
  - **amt_setTraceSink()** -- set trace the hook
  - **amt_setTraceLevel()** -- set trace level
- **Other APIs**
  - amt_getVer() -- get the version of the module: the date code posted

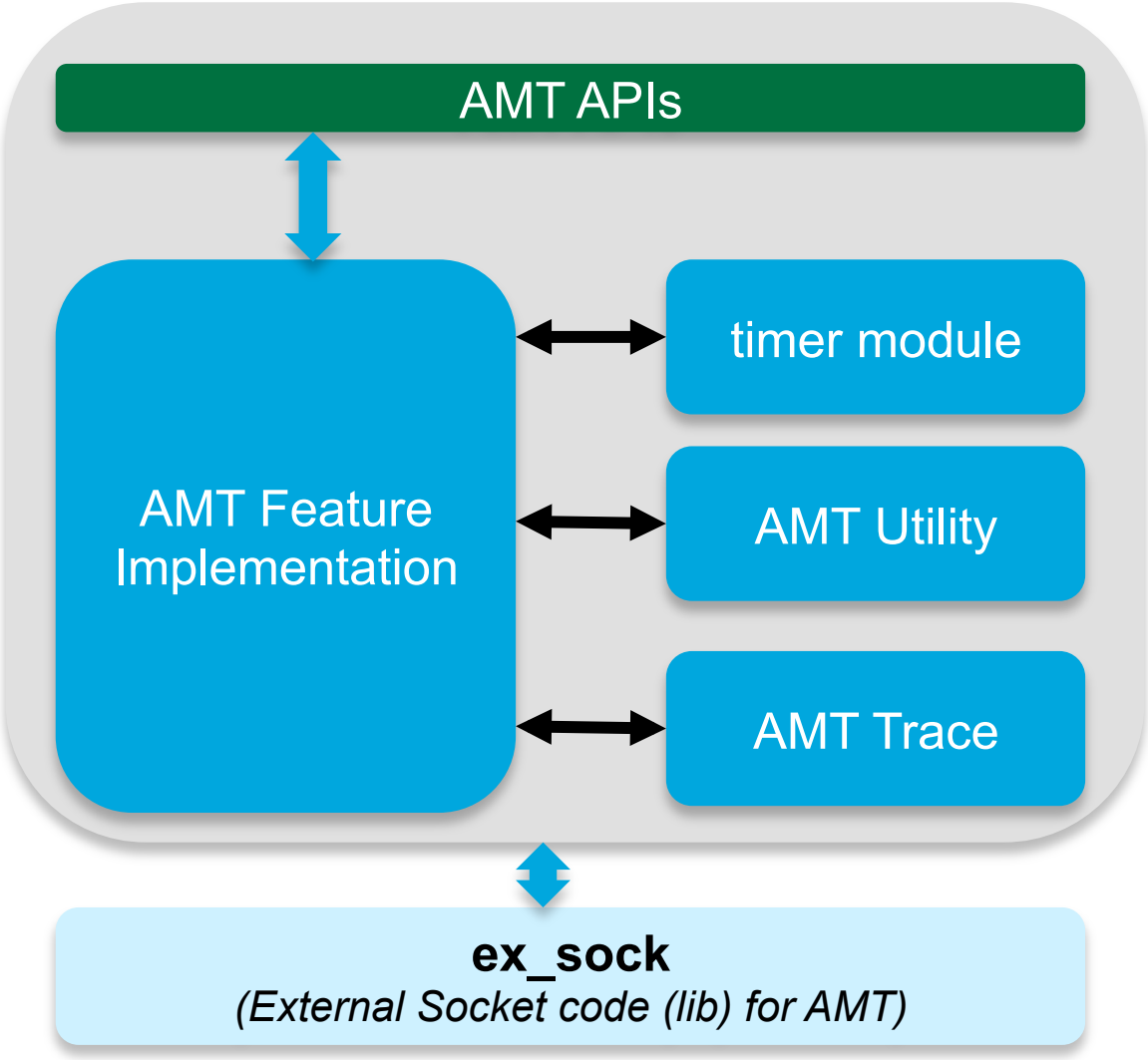
Typical Call-flows



Or

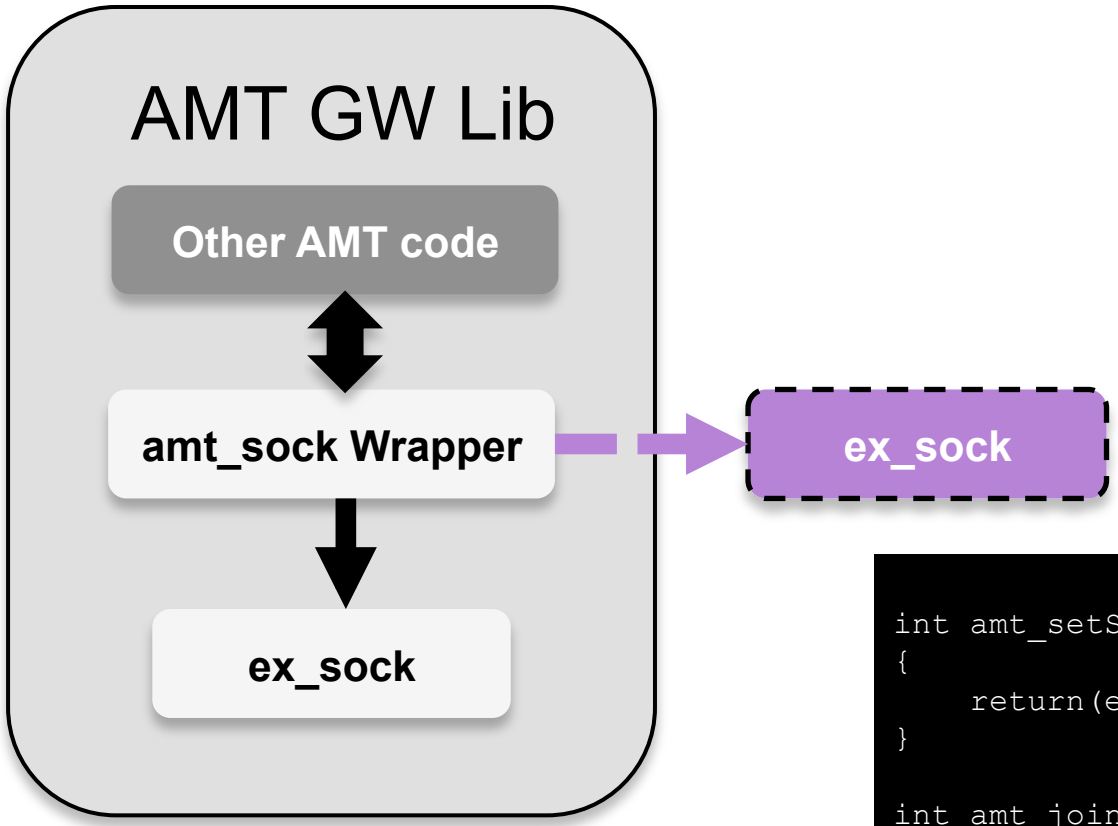


AMT Gateway Library: Architecture



- **AMT APIs:**
  -- A layer to provide AMT features to the Apps
- **AMT feature Implementation**:
  -- The actual AMT feature implementation module for such as tunnel setup, query, packet receiving and dispatching.
- **Timer Module**:
  -- A simple timing wheel to provide the keep-alive three-way handshaking and request retrying timer
- **AMT Trace**:
  -- A trace module for debugging. The trace could be shown internally using “print” or through a hook API to the App.
- **AMT Utility**:
  -- A bound of handy code to provide cross-platform utility features, such as getting the current time and creating a thread. (Currently, only Linux supported)
- **ex_sock**:
  -- Code to implement the required socket features for the AMT module lib. The code is intended to be replaced for different applications.

Socket code



(1) The ex_sock code provides the necessary socket functions

(2) amt_sock wraps the ex_sock code or lib. The other amt lib code calls the APIs provided in amt_sock.

(3) The ex_sock could be compiled as integrated part of lib or used as an external lib.

```
int amt_setSSMTTL(s32 s, u32 ttl)
{
    return(ex_setSSMTTL(s, ttl));
}

int amt_joinSSMGroup(s32 s, u32 groupIP, u32 sourceIP, u32 interfaceIP)
{
    return(ex_joinSSMGroup(s, groupIP, sourceIP, interfaceIP));
}
```

AMT Gateway Library: Source Code Structure

src:
amt.c amt_impl.c amt_impl.h amt_sock.c amt_sock.h amt_trace.c amt_trace.h amt_utility.c amt_utility.h

include: amt.h

Lib:
- **android:**
- **ios:**
- **linux:** libamt.a libamt64.a
- **mac:**
- **win32:**

Build:
- **android:**
- **ios:**
- **linux:** makefile
- **mac:**
- **win32:**

Test:
amtchk.c amtfeeder.c amtsink.c amtstream.c
- **android:**
- **ios:**
- **linux:**
  - makefile
  - **amtchk amtfeeder amtsink amtstream**
  - **trace-server:** makefile traceServer.c
  - **traceServer**
- **mac:**
- **win32:**

ex_sock:
ex_sock.c ex_sock_exp.h ex_sock.h

docs:
amt-gw-lib.pdf

examples:
ex1.c ex2.c ex3.c ex4.c

AMT Gateway Library: **amt_openChannel()**

```
/*
 * open a ssm and amt channel or a try with both
 * return a handle for success or NULL for failure
 */
amt_handle_t amt_openChannel( ipv4_t anycast,          /* an IP to find a relays;
                                                          not used if req for ssm only */
                              ipv4_t group,            /* group IP of SSM */
                              ipv4_t source,           /* source IP of SSM */
                              unsigned short port,     /* the destination IP of packets */
                              amt_connect_req_e req    /* through ssm or amt or both */
                              );
```

- Use this API to open a channel
- The API is implemented asynchronously
- The channel state could be accessed through **amt_getState**()

```
typedef enum {
    AMT_CONNECT_REQ_NONE  = 0,
    AMT_CONNECT_REQ_SSM   = 1,          // req for ssm connection only. anycast not used.
    AMT_CONNECT_REQ_RELAY = 2,          // req for amt connection only
    AMT_CONNECT_REQ_ANY   = 3,          // req for either ssm or amt connection. Whichever gets
                                        //  connected first is selected as the connection.

    AMT_CONNECT_REQ_END,
} amt_connect_req_e;
```

AMT Gateway Library: **amt_closeChannel()**

```
/*
 * close an opened channel
 *  return 0 for success and -1 for failure
 */
int amt_closeChannel(amt_handle_t handle      /* the handle open with amt_openChannel()
*/
                    );
```

- Use this API to close the opened channel with **amt_openChannel()**
- The API is implemented asynchronously for the connection through AMT and synchronously for SSM.
- For the connection through AMT, the actual channel close is done through AMT message of ***Membership Update*** with (S,G) blocked.
- Once the channel is closed, the handle is not longer valid.

AMT Gateway Library: `amt_poll()`

```
/*
 * poll to check if there are packets for the given handle array.
 * return n events for success with event set in rs, and -1 for failure.
 */
int amt_poll(amt_read_event_t *rs,    /* points to a handle array */
             int              size,   /* handle array size  */
             int              timeout /* in millisecond */
             );
```

- Use this API to to check the channel event for the opened channels with **`amt_openChannel()`**
- The API is implemented synchronously.
- If there is no event, this API will wait for ***timeout*** millisecond.
- This API returns 0 for no event, n>0 for n events and -1 for failure.

```
typedef enum {
   AMT_READ_NONE      = 0,
   AMT_READ_IN        = 1,        /* there is data in buffer */
   AMT_READ_CLOSE     = 2,        /* amt channel close */
   AMT_READ_ERR       = 4,        /* unexpected error */
   AMT_READ_END
} amt_read_event_e;
```

AMT Gateway Library: **amt_recvfrom()**

```
/*
 * retrieve a packet from the channel the "handle" points to
 * return the size of packet; -1 for failure
 */
int amt_recvfrom(amt_handle_t handle,      /* the handle open with amt_openChannel() */
                 unsigned char *buf,       /* buffer */
                 int maxBufSize            /* max buffer size */
                 );
```

- Use this API to receive the packet.
- The API is none-blocked for packet available or not.
- ***amt_poll()*** can be used to check the packet availability

AMT Gateway Library: **amt_addRecvHook()**

```
/*
 * add packet receiving sink
 */
typedef void (*amt_recvSink_f)(amt_handle_t handle, void *buf, int size, void *param);
int amt_addRecvHook(amt_recvSink_f recvSinkfuc,    /* a sink function to receive packets */
                    void *param                     /* a parameter to pass from the sink function */
                    );
```

- Use this API to add a packet receiving sink function.
- The sink will be called once there is a packet available
- Parameter ***param*** is passed through the sink function.
- This API is exclusive with ***amt_poll***() and **`amt_recvfrom()`** which are disabled once this API is called. ***amt_reset***() will need to call to re-enable them.

AMT Gateway Library: amt_setTraceSink()/amt_setTraceLevel()

```
/*
 * set trace sink and level
 */
typedef void (*amt_traceSink_f)(int level, char *msg, int size);
int amt_setTraceSink(int level,                 /* 0-10 with 0 to turn off tracing and
                                                   10 to enable the max tracing */
                     amt_traceSink_f traceFunc /* a sink function to pass the traces */
                     );
int amt_setTraceLevel(int level                 /* 0-10 with 0 to turn off tracing and
                                                   10 to enable the max tracing */
                      );
```

- Use these two APIs to add a trace sink function and set trace level.
- If trace sink function is not set, the library will use the internal trace functions to print out trace.

Examine test source code
To understand how to use the library

Example 1: find the relay interface address

```
#include <stdio.h>
#include "amt.h"

int main(int argc, char *argv[])
{
    unsigned int anycast;
    if (argc < 2) {
          printf ("usage: %s anycast\n", argv[0]);
          return 1;
    }
    anycast = ntohl(inet_addr(argv[1]));

    // set trace level
    //amt_setTraceLevel(AMT_LEVEL_10);

    amt_init(anycast);
    while((amt_getState(NULL) &
          AMT_SSM_STATE_CONNECTED)!=AMT_SSM_STATE_CONNECTED) {
          sleep(1);
    }
    amt_reset();

    return 0;
}
```

Notes:

(1) This program will establish a AMT connection.

(2) By default, the trace will print out the relay address from anycast IP, for example, "*ex1 192.168.255.1"* will return

02:53:34.277 amt_impl.c:handleDiscoveryResp-- discovered relay address:**192.168.255.101**

(3) To see all the traces, uncomment *amt_setTraceLevel(AMT_LEVEL_10);*

Code location: …/amt_gw_lib_10/*examples*/ex1.c

```
> gcc -o ex1 ex1.c -I../include -L../lib/linux -lamt64 -lpthread
```

Example 2: Subscribe to a AMT channel

```
. . .
// subscribe to a AMT channel
{
  unsigned char buf[1500];
  unsigned int  group = ntohl(inet_addr("232.10.10.10"));
  unsigned int  srcIP =  ntohl(inet_addr("192.168.0.100"));

  amt_handle_t  channelID = amt_openChannel(anycast, group, srcIP,
                                            9010, AMT_CONNECT_REQ_RELAY);
  amt_read_event_t rs={channelID,0};
  while(1) {
     int n =  amt_poll(&rs,  1, 2000);
     if (n>0) {
         int  len = amt_recvfrom(channelID, buf, 1500);
         if (len > 0) {
             printf("received packet with len = %u\n", len);
         } else {
             printf("Oh, get errors with amt_recvfrom() !!, Exit\n");
             return 1;
         }
      } else if (n<0) {
         printf("Oh, get errors with amt_poll()!!, Exit\n");
         return 1;
      }
    }

 }

. . .
```

Notes

(1) This program will establish a AMT connection and read packets from the channel.

(2)Error check is not complete

(3)More detailed code could be found in …/amt_gw_lib_11/test/amtpoll.c

Code location: …/amt_gw_lib_10/examples/ex2.c

```
> gcc -o ex2 ex2.c -I../include -L../lib/linux -lamt64 -lpthread

>  ex2 192.168.255.1
```

Example 2: Subscribe to a AMT channel – complete code

```
#include <stdio.h>
#include "amt.h"

int main(int argc, char *argv[])
{
    unsigned int anycast;
    unsigned char   buf[1500];
    unsigned int group = ntohl(inet_addr("232.10.10.10"));
    unsigned int srcIP =  ntohl(inet_addr(" 192.168.0.100"));
    amt_handle_t channelID
    amt_read_event_t rs;

    if (argc < 2) {
          printf ("usage: %s anycast\n", argv[0]);
          return 1;
    }
    anycast = ntohl(inet_addr(argv[1]));

    // set trace level
    //amt_setTraceLevel(AMT_LEVEL_10);

    amt_init(anycast);
    while((amt_getState(NULL) &
          AMT_SSM_STATE_CONNECTED)!=AMT_SSM_STATE_CONNECTED) {
          sleep(1);
    }

    // subscribe a AMT channel
    channelID = amt_openChannel(anycast, group, srcIP, 9010,
                                AMT_CONNECT_REQ_RELAY);
  rs.handle = channelID;
  while(1) {
    int n =  amt_poll(&rs,  1, 2000);
     if (n>0) {
       int  len = amt_recvfrom(channelID, buf, 1500);
       if (len > 0) {
                 printf("receive packet with len = %u\n", len);
       } else {
       printf("Oh,get errors with amt_recvfrom()!!\n");
       return 1;
         }
     } else if (n<0) {
     printf("Oh, get errors with amt_poll()!!, Exit\n");
     return 1;
      }
    }
  }

  amt_reset();
   return 0;
}
```

Example 3: use amt_addRecvHook() to receive packets

```
static void packetRecv(amt_handle_t handle,
                       void *_buf,
                       int size,
                       void *param)
{
    printf("receive packet with len = %u from channel:%p\n len, handle);
}

...
// add a callback function to receive all packets
amt_addRecvHook(packetRecv, NULL);
...
```

Code location: …/amt_gw_lib_10/examples/ex3.c

```
> gcc -o ex3 ex3.c -I../include -L../lib/linux -lamt64 -lpthread
> ex3 192.168.255.1
```

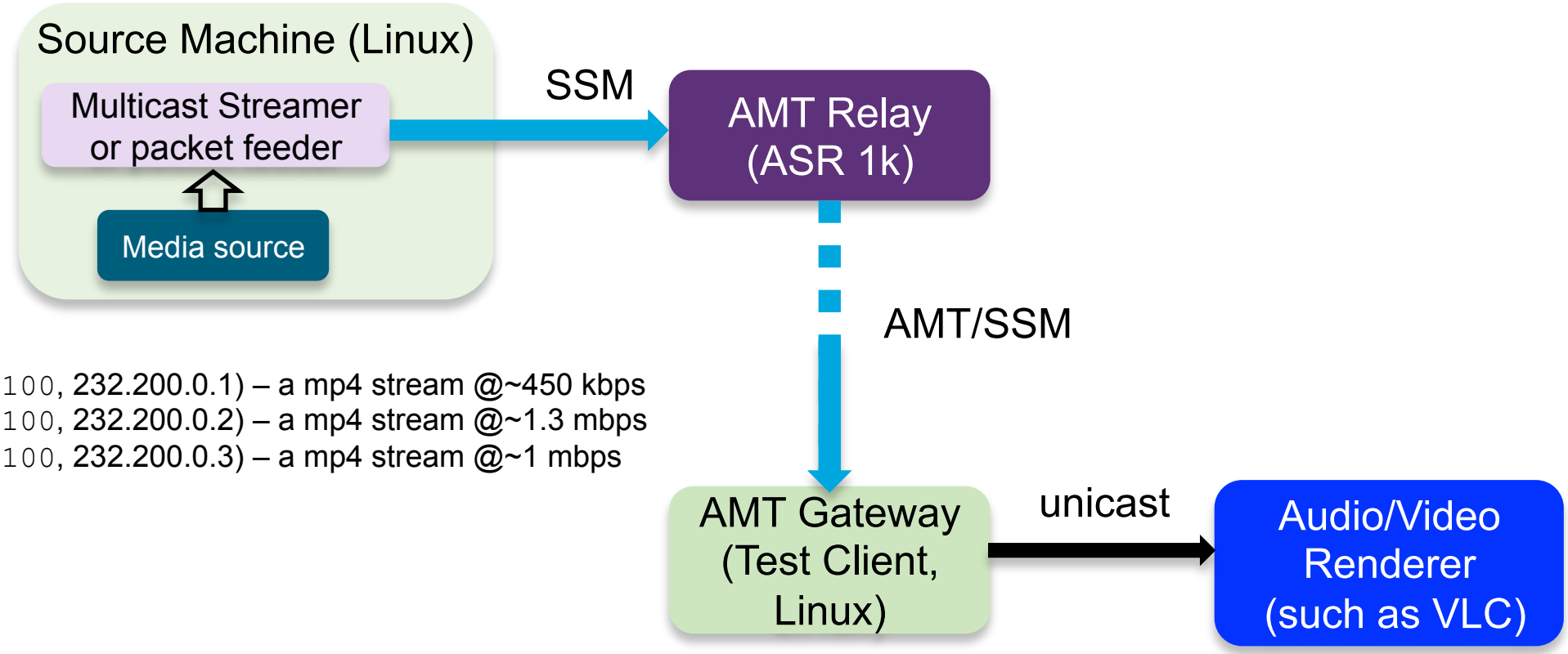
*More detail code could be found in …/*amt_gw_lib_11/test/amtsink.c

Example 3: use amt_addRecvHook() – complete code

```
#include <stdio.h>
#include "amt.h"

static void packetRecv(amt_handle_t handle,
                       void *_buf,
                       int size,
                       void *param)
{
    printf("ex3: receive packet with len = %u from
            channel:%p\n", size, handle);
}

int main(int argc, char *argv[])
{
    unsigned char buf[1500];
    unsigned int  group = ntohl(inet_addr("232.10.10.10"));
    unsigned int  srcIP = ntohl(inet_addr(" 192.168.0.100"));
    amt_handle_t  channelID;
    unsigned int  anycast;
    if (argc < 2) {
          printf ("usage: %s anycast\n", argv[0]);
          return 1;
    }
    anycast = ntohl(inet_addr(argv[1]));

    // set trace level
    //amt_setTraceLevel(AMT_LEVEL_10);

    // call the initial function.
    amt_init(anycast);

    // add a callback function to receive all packets
    amt_addRecvHook(packetRecv, NULL);

    // open a channel
    channelID = amt_openChannel(anycast,  group, srcIP,
                                9010, AMT_CONNECT_REQ_RELAY);

    pause();
    amt_reset();

    return 0;
}
```

Note that the error check is skipped.

Example 4: Receive a Video Stream and Render it



(1) (192.168.0.100, 232.200.0.1) – a mp4 stream @~450 kbps
(2) (192.168.0.100, 232.200.0.2) – a mp4 stream @~1.3 mbps
(3) (192.168.0.100, 232.200.0.3) – a mp4 stream @~1 mbps

Example 4: Media Streamer and Renderer

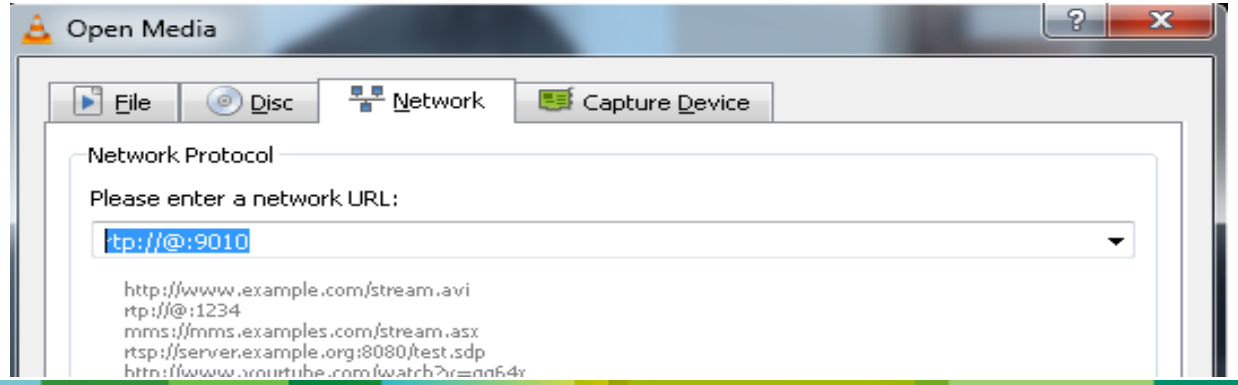
Source Streamer:

- **amtfeeder:** a streamer which reads the RTP packets pre-recorded from a live streaming session (such as VLC streaming) and sends out the rtp packets in exactly timing to SSM.

```
bram@testengine2:~/amt_gw_lib_11/test/linux$ ./amtfeeder -if em1 -ssm 232.10.10.10 -f ../media/elias-dumka.rtp
feed to: ssm (source IP=192.168.1.100 group IP=232.10.10.10 ) dstport=9010 len=64 interval=1000 (ms) srcMedia=../media/elia
s-dumka.rtp
total size=38873056
```

Rendering with VLC Media Player as an example:

- VLC: rtp://@:9010



Example 4: stream receiving code for Renderer

```
...

static void packetRecv(amt_handle_t handle,  void *_buf,  int size,
void *param)
{
    unsigned char *buf = (unsigned char *)_buf;
    unsigned int *renderIP = (unsigned int *)param;
    static ex_sock_t *pSock=NULL;
    unsigned short sport=0;

    if (pSock == NULL) { // to the render
          sport=0;
          pSock = ex_makeUDPSock(0, &sport, *renderIP, 9010);
          EX_ERR_CHECK(pSock != NULL, " ", _EMPTY);
    }
    ex_sendPacket(pSock,  buf, size);
}
...
```

Notes

(1) An AMT channel is created to receive RTP packets

(2) A socket is created to send the media stream to the renderer such as VLC Media Player.

(3) The code location: …/amt_gw_lib_11/examples/ex4.c

(4) `gcc -o ex4 ex4.c -I../include -I../ex_sock -L../lib/linux -lamt64 -lpthread`

(5) ex4 `192.168.255.1 192.168.1.x`

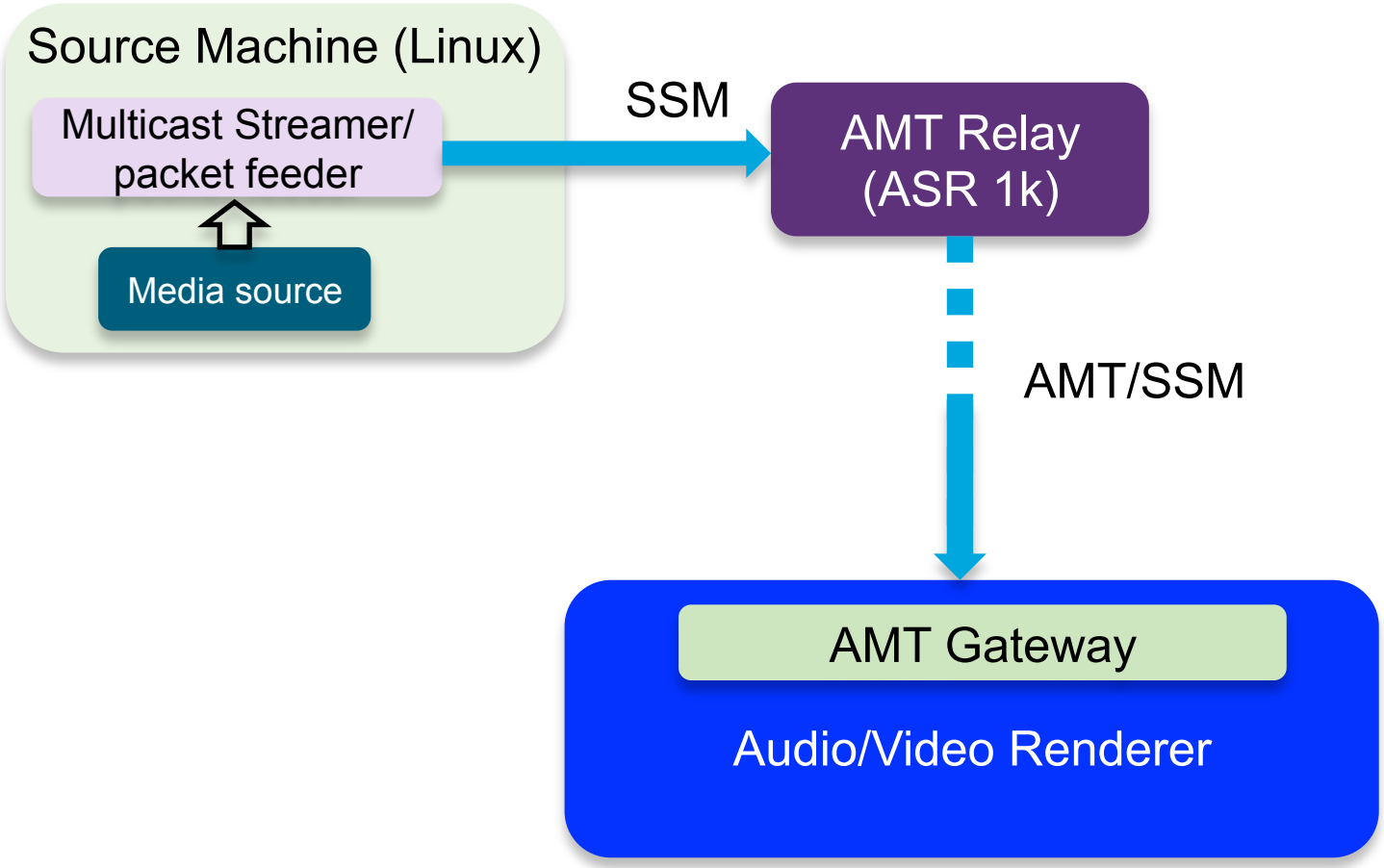
(192.168.1.x is your renderer IP)

(6) The full feature code could be found in …/amt_gw_lib_11/test/amtstream.c

Example 4: complete code

```
#include <stdio.h>
#include <sys/time.h>
#include <netinet/in.h>
#include "amt.h"
#include "ex_sock.h"

#define _EMPTY
static void packetRecv(amt_handle_t handle,  void *_buf,  int
size,  void *param)
{
    unsigned char *buf = (unsigned char *)_buf;
    unsigned int *renderIP = (unsigned int *)param;
    static ex_sock_t *pSock=NULL;
    unsigned short sport=0;

    if (pSock == NULL) { // to the render
          sport=0;
          pSock = ex_makeUDPSock(0, &sport, *renderIP, 9010);
          EX_ERR_CHECK(pSock != NULL, " ", _EMPTY);
    }
    ex_sendPacket(pSock,  buf, size);
}

int main(int argc, char *argv[])
{
    unsigned char   buf[1500];
    unsigned int group = ntohl(inet_addr("232.10.10.10"));
    unsigned int srcIP =  ntohl(inet_addr(" 192.168.0.100"));
    amt_handle_t channelID;
    unsigned int anycast, renderIP;
    if (argc < 3) {
      printf ("usage: %s anycast renderIP\n", argv[0]);
      return 1;
    }
    anycast = ntohl(inet_addr(argv[1]));
    renderIP = ntohl(inet_addr(argv[2]));

    // set trace level
    //amt_setTraceLevel(AMT_LEVEL_10);

    // call the initial function.
    amt_init(anycast);

    // add a callback function to receive all packets
    amt_addRecvHook(packetRecv, &renderIP);

    // open a channel
    channelID = amt_openChannel(anycast, group, srcIP,
                                          9010,
AMT_CONNECT_REQ_RELAY);

    pause();
    amt_reset();

    return 0;
}
```

Further work on audio/video renderer



What we covered

- Cisco drives effort to have open source AMT gateway library available to the public.
- The initial version is available from the github.com under Cisco fork and the library will be updated time by time.
- Introduction to the AMT lib and detail APIs
- In addition to the AMT lib, there are 4 examples of code in the repository to demonstrate how to use the library.
- 5 test programs with the complete source code to check relay setup and AMT channels are included in the repository.

Thank you.