SCTP TML Implementation

Forwarding and Control Element Separation WG

Kentaro Ogawa <ogawa.kentaro@lab.ntt.co.jp>
Jamal Hadi Salim <hadi@znyx.com>
SCTP TML Channels

<table>
<thead>
<tr>
<th>Channel Type</th>
<th>Priority Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reliable, High Priority</td>
<td></td>
</tr>
<tr>
<td>Semi-reliable, Medium Priority</td>
<td></td>
</tr>
<tr>
<td>Unreliable, Low Priority</td>
<td></td>
</tr>
</tbody>
</table>

Diagram:
- CE (Central Element)
- FE (Forwarding Element)

July 2008
IETF ForCES WG
Why Multiple SCTP sockets

• Alternative is multiple SCTP Streams
  – Essentially we could have a single socket with reliable, semi and unreliable packets
  – Problem is HOL in case a reliable packet sits infront of unreliable packet
    • Done in SCTP to avoid reordering ...

• There is some ongoing work from Michael Tuxen to allow for stream prioritization
  – But it is not standardized yet
    • Therefore we can not recommend it
Channel 1: High Prio, Reliable

• Used for
  – Configuration from CE to FE and responses from FE to CE
  – Query from CE to FE and responses from FE to CE
  – Some class of events
    • High priority alarms
Channel 2: Medium Prio, Semi-reliable

- SCTP allows you to semantically say
  - “Please send this message but obsolete it if you are unable to deliver it in 100ms”

- Used for
  - Events that are obsoleted over time
Channel 3: Low Prio, Unreliable

- Used for redirects from FE to CE
  - Some control protocols are reliable end to end
  - Some control protocols prefer obselence of messages over retransmissions
- Can be used for some other FE events that we can afford to loose because we can recover
  - Example some counters emitted synchronously
Implementation example
TML Parameterization

```xml
<FEM_CONFIG>
  ....
  <TML>
    <DEFAULT_TML>sctp</DEFAULT_TML>
  </TML>
  <CE_CONFIG>
    <CE>
      <PID>0x40000001</PID>
      <ADDRESS>169.254.100.1</ADDRESS>
      <HPORT>6700</HPORT>
      <MPORT>6701</MPORT>
      <LPORT>6702</LPORT>
    </CE>
    <CE>
      <PID>0x40000002</PID>
      <ADDRESS>169.254.100.2</ADDRESS>
      <ADDRESS>169.254.100.1</ADDRESS>
      <HPORT>6700</HPORT>
      <MPORT>6701</MPORT>
      <LPORT>6702</LPORT>
    </CE>
  </CE_CONFIG>
  ....
</FEM_CONFIG>
```
struct tml_target {
  char name[TML_NAME_SIZE]; //name of TML
  uint8_t version; //version of TML
  // PL invoker passes callback function to receive msgs
  int (*open)(... (*listen_func)(int, int, void *, void *), void *arg);
  int (*close)(unsigned long);
  // send packet via TML
  int (*send)(unsigned long, void *, int);
  // config/query things about TML
  int (*ioctl)(unsigned long, void *, void *);
};
PL-TML Bootstrap

• PL boots up and gets the TML name from xEM config
• PL scans for TML by name in libpath
• TML found
  – Load callback structure
  – Ready to use
• TML not found
  – Exit
Remote TML Bootstrapping

• PL calls TML open()
  – TML reads its xEM config parameters and connects via three sockets
• On success PL gets a filedesc
  – PL uses filedesc for ForCES communication
    • TML send() api calls
• On failure to connect to all endpoints an error code/filedesc is returned
Misc TML-PL API

- close() used to close connection between PL-TML
- ioctl() to issue control to the TML
  - example map PL message type to channel
- callback function passed in open() used to invoke PL from TML
  - arriving packets
  - events