SIP and Transport Protocols

Cristian Constantin
cco@iptel.org
Andrei Pelinescu-Onciul
andrei@iptel.org

This document is for informational purposes only, and Tekelec reserves the right to change any aspect of the products, features or functionality described in this document without notice. Please contact Tekelec for additional information and updates.
SIP / User Datagram Protocol (UDP)

- Fast and simple (best effort)
- Application level: atomic reads/writes on the sockets
- High throughput at a cost: congestion
- Congestion: infrastructure cannot support the amount of traffic; two types: application congestion / network congestion
- No explicit congestion control & avoidance mechanism in SIP/UDP; application has to take care of it
SIP / User Datagram Protocol (UDP) – cont.

- Application congestion in an Active/Stand-by failover
- Traffic rate during failover is close to the engineered cps
- Newly active server is experiencing congestion for several seconds due to retransmission spikes
SIP / Transmission Control Protocol (TCP)

- TCP offers a lot more than UDP: congestion control, retransmission, error control.

- However, TCP is a stream-oriented protocol used for reliable transfer of chunks of data from host A to host B; TCP was not meant for real-time signalling.

- Disadvantages for SIP: continuous flow of data (no message boundaries), application layer synchronization/serialization of reads/writes, usually no fine grain configuration of internal timers.
SIP / Stream Control Transmission Protocol (SCTP)

- SCTP is the Swiss army knife of transport protocols
- UDP-like features: message boundary preservation, unordered message delivery, one-to-many sockets at the application level.
- TCP-like features: positive (selective) acknowledgment, retransmission of lost data, windowed flow control, congestion control, one-to-one sockets at the application level
SIP / Stream Control Transmission Protocol (SCTP) – cont.

- **SCTP unique features:**
  - multihoming
  - multiple streams per connection
  - built-in heartbeats
  - most of the protocol parameters configurable per system and per socket (association)
  - exposes asynchronously its internal states/events to the application level through the use of notifications

- **Useful for SIP:**
  - message boundary preservation
  - fine tuning of the timer values
  - Multipath / transport layer failure detection per path
  - asynchronous notifications of socket events
Pitfalls

- the SCTP socket API is a moving target still under development,
- due to novelty, the level of complexity of some of the SCTP stack implementations is inversely proportional with the time spent on testing them
- sometimes their performance in terms of throughput is not on a par with the one offered by TCP.
UDP vs TCP vs SCTP

• Configuration:
  • Hardware: Intel XEON, 16 way, 2.53 GHz, 24 GB memory (8 GB used by SER), Gb network cards
  • SER / Linux CentOS
  • The test bed is emulating proxy to proxy signalling
  • small number of sip nodes, small number of tcp connections/sctp associations
  • a lot of calls coming from/go ing to the same node
  • SER is just routing the calls using prefix based routing (max. 30 prefixes)

• Tested call scenarios:
  • Transaction replied by SER directly: INVITE / 404
  • End to end transaction: INVITE / 100 / 180 / 200
  • Call consists of: INVITE / 100 / 180 / 200 / ACK / BYE / 200; it is both initiated and terminated by UAC, ringing time: 12s, call duration: 120s in average (35s – 205s)
UDP vs TCP vs SCTP – cont.

- Max throughput: UDP, directly replied transaction: 27K tps
- Calls on UDP (using raw sockets for send): 8K cps; active-standby failover produces much higher spikes (around 20K)
- Calls on TCP: 10K cps
- Calls on SCTP: 2.5K cps; most reliable active-standby failover (no call loss)
UDP vs TCP vs SCTP – issues

- **Timer process congestion**
  - there are: multiple udp/tcp/sctp processes, only two timer processes
  - at high call rate the timer processes cannot cope with all the events generated by traffic processed in the udp/tcp/sctp processes

- **SCTP relatively poor throughput**
  - Linux kernel issue
  - One-to-many sockets do not scale properly with the number of readers/writers (synchronization bottleneck)

- **Active/standby failover with TCP**
  - Depends on how quickly the peers detect the failure and reconnect
  - Worse than SCTP