TLS Crypto Offload to Network Devices

Boris Pismenny

Netdev conference, Tokyo, October 2016
Motivation

- Encrypted communication is becoming ubiquitous
- How CPU intensive is crypto?
- Compared 2 TLS cipher suites AES128-GCM vs. NULL using OpenSSL single stream

<table>
<thead>
<tr>
<th>Cipher suite (symmetric crypto-digest)</th>
<th>CPU time (user/system/usage)</th>
<th>Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES128-GCM</td>
<td>0.54 sec/0.78 sec/94%</td>
<td>6.48 Gbit/s</td>
</tr>
<tr>
<td>NULL</td>
<td>0.08 sec/0.61 sec/81%</td>
<td>10.4 Gbit/s</td>
</tr>
</tbody>
</table>

85% reduced CPU
Existing Solutions: TOE + TLS

- Not as robust and well tested as the software TCP stack running on the host
- Difficult to maintain
- Likely to have inferior congestion control and retransmission polices
- Lags behind state of the art software implementation
Existing Solutions: TOE + TLS

- Not as robust and well tested as the software TCP stack running on the host
- Difficult to maintain
- Likely to have inferior congestion control and retransmission policies
- Lags behind state of the art software implementation
Existing Solutions: Memory To Memory Crypto

- Inefficient for packet processing
  - Multiple PCI crossings
  - Additional processing overhead
    - Prepare crypto descriptors
    - Handle crypto completion interrupt

- Adds significant latency

**Crypto Offload via PCIe:**

[Diagram showing HOST, Crypto, and NIC connections]
Existing Solutions: Memory To Memory Crypto

- Inefficient for packet processing
  - Multiple PCI crossings
  - Additional processing overhead
    - Prepare crypto descriptors
    - Handle crypto completion interrupts
- Adds significant latency

Crypto Offload via PCIe:
TLS Offload To Network Devices

- NIC holds the crypto state
- NIC encrypts / decrypts packets on the fly
- Cipher text is never held in RAM
- Single PCIe round-trip
- Reduce latency

**Crypto Offload To NIC:**

![Diagram showing host and NIC+crypto]
**User Space:**
Application Data

**KTLS:**
Fragment \((2^{14})\)

**KTLS:**
Encrypt & Authenticate

**KTLS:**
TLS Records

**TCP:**
Segment (MSS)

---

**H** – TLS Record Header

**T** – TLS Record Authentication Tag
Software Stack

- Integrates with Kernel TLS
  - Added AF_KTLS “offload” socket option

- Offload symmetric crypto
  - AES-GCM

- Software stack unchanged (even simplified)
  - kTLS (without crypto)
  - TCP/IP
  - Congestion control
  - Memory management
TLS Crypto Offload vs. Other Protocols

- Ideally, packets would be processed independently:
  - IPsec
  - DTLS
  - QUIC

- However, in TLS *each record* is processed independently.

- Intermediate record state must be tracked by hardware
  - Used by subsequent packets that are part of a previous record

<table>
<thead>
<tr>
<th>TLS Records:</th>
<th>TLS Record 1</th>
<th>TLS Record 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP Packets:</td>
<td>P1</td>
<td>P2</td>
</tr>
<tr>
<td></td>
<td>P3</td>
<td></td>
</tr>
</tbody>
</table>
Driver Fast Path

1) Check packet belongs to offloaded socket skb->sk->sk_offloaded - OK
2) Check packet TCP sequence number against expected TCP sequence number - OK
3) Encrypt and authenticate packet in hardware

**TLS Records:**

| TLS Record 1 | TLS Record 2 | TLS Record 3 |

**TCP Packets:**

| P1 | P2 | P3 | P4 | P5 | P6 | P7 |
Driver Slow Path – Resync Flow

1) Check packet belongs to offloaded socket `skb->sk->sk_offloaded`
2) Check packet TCP sequence number against expected TCP sequence number – **Wrong!**
   2.1) **Resync**: Fix hardware TLS context
3) Encrypt and authenticate packet in hardware

**TLS Records:**

<table>
<thead>
<tr>
<th>TLS Record 1</th>
<th>TLS Record 2</th>
<th>TLS Record 3</th>
</tr>
</thead>
</table>

**TCP Packets:**

<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
<th>P6</th>
<th>P7</th>
</tr>
</thead>
</table>
Resync Flow Requirements

- Software needs to keep track of TLS records
- Mapping from SKB to its first TLS record
- TLS records may be released only after the last SKB that overlaps it is ACKed
  - Prevent releasing of partially acknowledged TLS record

<table>
<thead>
<tr>
<th>TLS Records:</th>
<th>TCP Packets:</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLS Record 1</td>
<td>P1 P2 P3 P4 P5 P6 P7</td>
</tr>
<tr>
<td>TLS Record 2</td>
<td></td>
</tr>
<tr>
<td>TLS Record 3</td>
<td></td>
</tr>
</tbody>
</table>
Resync Flow

GET TLS RECORD

KTLS

TCP

WRITE QUEUE

SKB 1

SKB 2

TLS RECORD 1

TLS RECORD 2

SH_INFO

SH_INFO

DRIVER
Proposed Solution

**kTLS:**
- Holds a reference on TLS record data
- Holds a mapping from TCP sequence number to TLS record data
- Expose mapping to offloading device drivers

**TCP:**
- Call kTLS on tcp_clean_rtx_queue() to release acknowledged TLS records

**Driver:**
- On Resync query kTLS mapping according to TCP sequence number
### Preliminary Performance Results

- **TLS single stream AES128-GCM using raw recv (no decrypt) with small TLS records (1457B)**

<table>
<thead>
<tr>
<th>Metric</th>
<th>TCP</th>
<th>GNU TLS</th>
<th>kTLS*</th>
<th>kTLS + Offload</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throughput</td>
<td>10-23Gbps</td>
<td>5.1Gbps</td>
<td>4.3Gbps</td>
<td>8.8Gbps</td>
</tr>
<tr>
<td>CPU</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
</tbody>
</table>

*kTLS version from 3 months ago*
Discussion Items

- Routing changes - plaintext packets must be routed to the offloading device
  - Bind to device
  - Software fallback

- New protocol family AF_KTLS_OFFLOAD
  - In addition to AF_KTLS or another socket option?

- Original TCP socket use
  - Requires flushing kTLS socket

- Memory accounting
  - TLS record accounting

- Zero copy sendfile + crypto offload
  - Retransmission + Stale data
Zero copy sendfile?

TLS HEADER → SKB

Page cache → SKB

Dummy Authentication tag → SKB
Zero copy sendfile?

**TLS HEADER** | **Plaintext** | **Dummy Authentication tag**

**TLS HEADER** | **Cipher text** | **Authentication tag**

Dropped

**TLS HEADER** | **Cipher text** | **Cipher text** | **Authentication tag**
Zero copy sendfile?

TLS HEADER | Plaintext | Modified | Dummy Authentication tag

TLS HEADER | Cipher text

Retransmission

TLS HEADER | Cipher text | Authentication tag
Zero copy sendfile?

The diagram illustrates the process of sending data over a TLS (Transport Layer Security) connection. It shows the transmission of plaintext and ciphertext through TLS headers and authentication tags.

1. **TLS HEADER** is followed by **Plaintext** which is modified and then transmits the **Dummy Authentication tag**.
2. The **Cipher text** is transmitted followed by a **Retransmission** indicating an authentication failure.
3. The **Cipher text** is then transmitted again with an **Authentication tag**.
Thank You