Summer 2003 IP VTC Issues

SUMMARY

This document describes the analytical process V/Dops used to identify the causes for the failure of the July 18th HemOnc Internet Protocol Video-TeleConferencing (IP VTC) event. The author intends the audience to be data networking professionals.

SCCA

Our analysis identifies a bug in the software loaded on the SCCA's firewall which has induced intermittent (every 10-30 minutes) and unreliable IP VTC disconnects not only on July 18th but also across the last several years.

We speculate that wide-scale disruptions of the University of Washington's data network, inflicted by UW technical staff on July 18th in an effort to harden their data network against a recently discovered vulnerability, caused the bulk of the IP VTC disconnects which occurred during the HemOnc VTC on that date.

FHCRC

Our analysis also uncovered a bug in the software loaded on the Hutch's firewall which induced reliable (every 28 minutes) disconnects for IP VTC held at the Hutch. This bug had no influence on the July 18th HemOnc IP VTC, of course, but because its presence substantially delayed resolution of the SCCA-specific issue, we include a description of this analysis.

Resolution

As of Wednesday July 30th, we have fixed the IP VTC-related bug in the SCCA's firewall and have validated our fix by sustaining a handful of SCCA <–> UW IP VTC for an hour each with no disruptions to service. As of August 19th, we have not fixed the IP VTC-related bug in the Hutch's firewall.

OVERVIEW

History

Hutch AV has employed ISDN-based Polycom video teleconference units for years and has recently (Summer 2001) begun experimenting with IP-based Polycom units. Specifically, these IP-based units have been used to service a twice/month HemOnc call (SCCA / UW) and, as of Spring 2003, a once/week CPRP seminar (Seattle / New Mexico). Hutch AV operates from two locations: the Fred Hutchinson Cancer Research Center and the Seattle Cancer Care Alliance -- Outpatient Clinic. Both facilities are located on South East Lake Union.

Sometimes, IP-based VTC initiated from the SCCA to the UW terminated unexpectedly after varying amounts of time, typically 10 – 30 minutes. AV staff correlated fewer disconnects with locating the Polycom unit on G3 and more frequent disconnects with locating the unit on G1. VTC calls initiated from anywhere outside the SCCA to Polycom units located inside the SCCA consistently failed to make the initial connection.

IP-based VTC initiated from within the Hutch to New Mexico worked reliably.

On July 18th, the HemOnc call failed every few minutes; after ~40 minutes, the participants abandoned their effort to establish this call.

Technical Details

The SCCA network lives within 140.107.240.0/24 and sits behind a (highly-available) pair of Cisco PIX 535 running PIX OS 5.3(1). FHCRC lives within the remainder of 140.107.0.0/16 and sits behind a (highly-available) pair of Nokia IP740s running Checkpoint NG FP3 Hotfix 2. The two companies share a common DMZ (206.253.211.192/27) which provides connectivity to the Internet (via a T3 to the InterNAP), the GigaPOP (via a gigabit ethernet link), and CHRMC (via a VPN crossing the GigaPOP).

At the access layer, both companies deploy Cisco Catalyst 4006s equipped with 10/100 line cards running over a modern Cat 5 cabling plant. At the distribution layer, both companies deploy Cisco Catalyst 6506s with MSFC1s. Past the access-layer, the entire packet infrastructure runs at Gigabit Ethernet over a modern fiber optic cable plant composed of both multi-mode and single-mode fiber.

We own two Polycom FX equipped with both BRI and Ethernet interfaces running v5.0 of the Polycom OS. We purchase our Polycom via a reseller, GBH Communications.

WHAT CHANGED

Hutch IT regularly changes the rule sets on both the PIX and the Nokia. On July 13th, Hutch IT rebooted both the PIX and the Nokia, during a planned maintenance window, for work unrelated to this issue. Recently, Hutch AV had upgraded the Polycom units from 4.x to 5.0 of the Polycom OS.

ANALYSIS

Data-Gathering

V/Dops staff spent two weeks gathering packet traces, employing test calls between Polycom units located at the Hutch, at the SCCA, and at the UW, and analyzing the results, employing an outside consultant (Mike Pennachi of Network Protocol Specialists) and tech support at Cisco, Polycom, and Nokia.

Calls placed from a Polycom within the Hutch to another Polycom within the Hutch did not display the unexpected disconnect. Same with calls placed from a Polycom within the SCCA to another Polycom within the SCCA. Calls placed between the two organizations terminated after varying amounts of time, ranging from 10-30 minutes.

Results

We discovered two bugs, one in the PIX OS and another in CheckPoint.

PIX OS

The PIX OS contained a bug in which it would sometimes identify the TCP-based management stream flowing between the two Polycoms as "half-open", despite the fact that neither side had initiated a TCP FIN. Having identified a session as "half-open", the PIX would then close that session (spoofing a TCP FIN the far side) after its half-open timer expired: 10 minutes, by default -- thereby tearing down the call. During our analysis, we reset this timer to infinity. On Sunday July 27, we upgraded the PIX to a newer version of its OS which contains a fix for this bug.

This is a known bug in the PIX OS[1] and is fixed in later revs of the OS.

As for IP VTC initiated from outside the SCCA, the PIX are configured, by SCCA policy, to deny all incoming efforts to establish an IP VTC session. Thus, all efforts to initiate VTC from the outside were failing. Here, the PIX were and are functioning as designed and configured.[2]

CHECKPOINT

Polycom uses a controversial but popular technique for keeping its TCP-based management session alive, the use of decremented TCP ACKs.[3] (See http://www.thinkingsecure.com/docs/TCPIP-Illustrated-1/tcp_keep.htm for the relevant section from Stevens' TCP-IP text). One or the other of the Polycoms (we have not developed a model to understand which) sends such an ACK every ~28 minutes. CheckPoint blocks the ACK and spoofs a TCP RST to each side of the conversation, thereby tearing down the call.

This is a known bug[4] in CheckPoint and is fixed with CheckPoint NG AI, which we plan to install on the Hutch firewalls at an as yet unspecified date.

CONFIRMATION

V/Dops staff confirmed that calls from the SCCA to the UW were functioning normally on August 7th, with additional tests from both G1 and G3 on August 12th.

[1] cscdw55700

[2] SCCA IT may at some point choose to change this policy.

[3] More commonly, TCP implementations will send a garbage byte of data, incrementing the ACK count by one. And of course, per Stevens, the entire concept of implementing the Keep-Alive functionality at the TCP layer remains a subject of research and discussion, with some groups recommending that Keep-Alive be implemented at the application layer rather than at the TCP layer.

[4] sk19254

Speculation

The inquiring reader may wonder why IP-based VTC worked at all, prior to upgrading the software in the PIX and the Nokia.

We speculate that the PIX bug was unreliable, affecting some VTC and not others -- this lead AV staff to correlate failures with location (G1 failed, G3 succeeded), when in fact failures were actually being caused by transitory changes within the PIX, during which this bug appeared and disappeared. We speculate that the July 13th reboot of the PIX exacerbated this bug, leading to a more reliable of pattern of call interruptions.

We speculated that the same was happening with the CheckPoint bug … but then we started querying the user community. Turns out only one IP VTC has been occurring from the Hutch, and after consulting with the users in question, we discovered that the 28 minute issue occurred reliably for them … they took this as normal behavior and hadn't mentioned it to AV staff.

We speculate that the version of CheckPoint we are running interprets the decremented ACK which Polycom uses as the signature of sequence-number guessing, i.e. as a man-in-the-middle attack; this would explain its subsequent TCP RST spoofing.

As for the meltdown of the HemOnc call on July 18th, we note that on that date the UW disrupted their data network substantially, installing a software upgrade in their routers in an effort to harden them against a recently discovered security vulnerability. We speculate that this event combined with the PIX bug created the July 18th HemOnc VTC meltdown.

COMPLETING THE STORY

Key Points

- The PIX bug, and possibly other issues, have intermittently disconnected the HemOnc VTC for several years, but not reliably enough for staff to gather traces of the experience.

- End-users at the Hutch did not complain about disconnects, leading us to erroneously believe that their calls were trouble-free.

- Both the SCCA and the Hutch firewalls contained bugs which disconnected IP VTC.

- The UW disrupted their data network on July 18th, leading to the aborted VTC on that day.

Chronology

In 2001, Hutch AV started deploying IP VTC in the SCCA. The PIX intermittently inflicted disconnects, but not reliably enough for staff to acquire packet traces. Late in Q2 2003, Hutch

AV deployed IP VTC to one user group at the Hutch … this group experienced reliable disconnects (28 minutes) but did not feed this data back to Hutch AV.

On July 18th, the HemOnc VTC failed spectacularly due to data network disruptions at the UW; Hutch AV opened a ticket with Center IT.

Wednesday July 30 identify and install a work-around for the bug in the PIX which was intermittently disconnecting calls. However, we use a Hutch – SCCA test bed, and the bug in the Hutch firewalls masks the fact that our fix has resolved the SCCA side of the issue. In fact, had we tried an SCCA – UW call, we would have had success. However, we believe, erroneously, that the Hutch firewalls are not contributing to the issue and therefore that we have not resolved the problem.

We validate our understanding of the PIX issue which we had identified with Cisco, receiving confirmation from Cisco's tech support that the version of the PIX OS we are running contains a bug which would account for the behavior we are seeing.

Thursday July 31
We spend this day continuing to analyze the issue, looking for a single cause to explain the behavior we were seeing. We discover the CheckPoint bug … but since we believe that the Hutch firewalls are functioning correctly, vis-à-vis IP VTC, we misinterpret what we are seeing. Unable to correlate what we are seeing with an SCCA-specific issue, we engage a consultant (Mike Pennachi of Network Protocol Specialists) to assist us. We attempt to engage Polycom tech support, via our reseller, GBH. We attempt to engage CheckPoint, via our reseller, Nokia.

Friday August 1
We gathered particular packet captures in preparation for Monday.

Monday August 4
We spent the day with Mike Pennachi, eventually identifying and accurately characterizing the CheckPoint issue. No substantive response from GBH or Nokia.

Tuesday August 5
We validate the model we had built on Monday, using additional tests and traces. Nokia tech support relays the information from CheckPoint that a newer version of the CheckPoint software fixes a known issue with H.323, the protocol used by our IP VTC. Nokia doesn't know what that issue is. Our analysis doesn't show an H.323 issue; rather, our analysis shows a TCP issue. As a result, we are unsure whether or not the CheckPoint patch addresses the issue we are seeing: (a) Perhaps CheckPoint is inaccurately characterizing the issue which this patch fixes, or (b) perhaps CheckPoint doesn't know about this bug. Still no response from Polycom – we notify our reseller (GBH) that we have discovered a cause for the issue and no longer need Polycom's assistance.[5]

[5] In other words, we gave up on Polycom.

Wednesday August 6
In an effort to engage Nokia, we gather precise "smoking gun" traces.

Thursday August 7
We tested our understanding by running two IP VTC from the SCCA to the UW: both functioned fine for an hour a piece.

What Remains

As of this writing, Nokia has not relayed to us a deeper understanding of the H.323 bug supposedly fixed in the latest rev of the CheckPoint software. Nor have we upgraded our Nokia firewalls. Until we upgrade to the latest rev of CheckPoint, we won't know whether or not this release fixes the TCP-oriented bug we've identified.

LESSONS LEARNED

Include Intermediate Techies and End-Users
Believing that end-users were not seeing the 28 minute disconnect issue on Hutch IP VTC was a piece of inaccurate information which lead us astray, burning the better part of a week, until the consultant put us back on track. Had we included Hutch AV staff and/or end-users in our analysis, this error would have come to light more quickly.[6]

Include intermediate techies and end-users regularly in discussion, as new information comes to light, and solicit comments and input.

Buy Tech Support Directly from the Manufacturer
Of the three technology companies involved in this case (Cisco, CheckPoint, Polycom), we purchase tech support directly only from Cisco; we acquire tech support for the other two products via resellers. Cisco answered our query within twenty-four hours of opening a case; neither Polycom (via GBH) nor CheckPoint (via Nokia) were able to engage this issue at the depth we needed. In my experience, this is the usual trouble with purchasing tech support from resellers – the intervening layer often gives me excellent "end-user" support but is unable to deliver support for deeply technical issues..

Investigate purchasing tech support contracts directly with CheckPoint and Polycom.[7]

Sometimes, the problem really is the firewall
The overwhelming majority of problems blamed on the firewall end up originating from either the client or the server, with the firewall functioning as designed and configured. However … every now and then, the issue really does sit within the firewall.

[6] The hard part in this is that intermediate techies and end-users may not be willing to allocate the time to spend the hours – or, in this case, weeks – needed to see the process through completion.

[7] Some manufacturers refuse to support end-users directly, requiring that the end-user pass first through a reseller. If Polycom or CheckPoint adhere to such a policy, we may find implementing this step difficult.

ACKNOWLEGEMENTS

Many thanks to Rande Gray (UW AV) for providing a test Polycom at the UW at short notice..
And thanks to various members of UW C&C and UW MCIS who patiently explained to us that no firewalls sit on their side of the equation.

TRACES

All traces taken with Network Associates Sniffer Portable and typically contained ~40,000 packets. Traces below have been filtered to include only the Polycom TCP-based management session (thus excluding the UDP-based H.323 streams and the TCP-based H.245 control stream).

Successful

Here is an example of a successful VTC session. Notice how the initiating Polycom, 140.107.241.49 (located in the SCCA) sets up a TCP-based management session in packets 1-3 with the classic TCP SYN-SYN-ACK three-way handshake to its partner, 140.107.249.43, also located in the SCCA. Notice also that the last ACK, during call-setup, from 140.107.241.49 occurs at packet 8 and declares to its partner that its receive buffer has reached byte 110903201 (`Seq = 110903201`).

The next time 140.107.241.49 sends a frame is in packet 9, approximately 28 minutes after the beginning of the call, and here it declares that its receive buffer has reached byte 1808404076 (`Seq = 1808404076`), i.e. one less than the byte count it declared in packet 8. Evidently, its partner understands this use of a decremented ACK, because a millisecond later, 140.107.249.49 replies with its own ACK in packet 10. And starting in packet 11, forty-four minutes (2651 seconds) after the call started, we see TCP tearing down the connection in a predictable fashion. (In fact, the human operator used the Polycom's interface to terminate the call.)

```
No. Time        Source              Destination          Protocol Info

  1 0.000000    140.107.241.49      140.107.249.43       TCP      3230 > 1720 [SYN] Seq=110903013 Ack=0 Win=23360 Len=0
  2 0.000809    140.107.249.43      140.107.241.49       TCP      1720 > 3230 [SYN, ACK] Seq=15755063 Ack=110903014 Win=23360 Len=0
  3 0.001948    140.107.241.49      140.107.249.43       TCP      3230 > 1720 [ACK] Seq=110903014 Ack=15755064 Win=23360 Len=0
  4 0.054827    140.107.241.49      140.107.249.43       Q.931    SETUP[Short Frame]
  5 0.085452    140.107.249.43      140.107.241.49       Q.931    ALERTING[Short Frame]
  6 0.347116    140.107.241.49      140.107.249.43       TCP      3230 > 1720 [ACK] Seq=110903201 Ack=15755170 Win=23254 Len=0
  7 0.467090    140.107.249.43      140.107.241.49       Q.931    CONNECT[Short Frame]
  8 0.747067    140.107.241.49      140.107.249.43       TCP      3230 > 1720 [ACK] Seq=110903201 Ack=15755315 Win=23109 Len=0
  9 1709.959049 140.107.249.43      140.107.241.49       TCP      1720 > 3230 [ACK] Seq=15755314 Ack=110903201 Win=23173 Len=0
 10 1709.960350 140.107.241.49      140.107.249.43       TCP      3230 > 1720 [ACK] Seq=110903201 Ack=15755315 Win=23109 Len=0
 11 2651.744799 140.107.241.49      140.107.249.43       Q.931    RELEASE COMPLETE
 12 2651.746262 140.107.241.49      140.107.249.43       TCP      3230 > 1720 [FIN, ACK] Seq=110903252 Ack=15755315 Win=23109 Len=0

 13 2651.747738 140.107.249.43      140.107.241.49       TCP      1720 > 3230 [ACK] Seq=15755315 Ack=110903253 Win=0 Len=0
 14 2652.196265 140.107.249.43      140.107.241.49       TCP      1720 > 3230 [FIN, ACK] Seq=15755315 Ack=110903253 Win=0 Len=0
 15 2652.198058 140.107.241.49      140.107.249.43       TCP      3230 > 1720 [ACK] Seq=110903253 Ack=15755316 Win=0 Len=0
```

PIX Bug

Here again we see the initiating Polycom, 140.107.241.49 (located in the SCCA), build a TCP connection with its partner, 140.107.70.55 (located in the Hutch) using the three-way TCP handshake in packets 1-3.

At about 17 minutes into the call (1064 seconds), in packet 9, the initiating Polycom sends management data to its partner. However, its partner ostensibly responds with a TCP RST (packet 10). I saw ostensibly because a trace taken by a sniffer located next to 140.107.241.49 does not contain this RST. Furthermore, notice that "Time To Live" (TTL) in packet 8 (and previous packets) is 55, whereas TTL in packet 10 is 58: this suggests either that the two packets traveled radically different paths (different by three routers) or that the originator of packet 10 is spoofing. In fact, consulting our network map, we can see that the box located three hops closer to the originating Polycom are the Cisco PIX, making them a likely suspect as the originator of this TCP RST.

```
    No. Time        Source                Destination           Protocol Info

      1 0.000000    140.107.241.49        140.107.70.55         TCP      3230 > 1720 [SYN] Seq=2142899995 Ack=0 Win=23360 Len=0
      2 0.000681    140.107.70.55         140.107.241.49        TCP      1720 > 3230 [SYN, ACK] Seq=18860277 Ack=2142899996 Win=23360 Len=0
      3 0.003470    140.107.241.49        140.107.70.55         TCP      3230 > 1720 [ACK] Seq=2142899996 Ack=18860278 Win=23360 Len=0
      4 0.051919    140.107.241.49        140.107.70.55         TCP      3230 > 1720 [PSH, ACK] Seq=2142899996 Ack=18860278 Win=23360 Len=187
      5 0.105136    140.107.70.55         140.107.241.49        TCP      1720 > 3230 [PSH, ACK] Seq=18860278 Ack=2142900183 Win=23173 Len=106
      6 0.333408    140.107.241.49        140.107.70.55         TCP      3230 > 1720 [ACK] Seq=2142900183 Ack=18860384 Win=23254 Len=0
      7 0.676084    140.107.70.55         140.107.241.49        TCP      1720 > 3230 [PSH, ACK] Seq=18860384 Ack=2142900183 Win=23173 Len=145
      8 0.933114    140.107.241.49        140.107.70.55         TCP      3230 > 1720 [ACK] Seq=2142900183 Ack=18860529 Win=23109 Len=0
      9 1064.209619 140.107.70.55         140.107.241.49        TCP      1720 > 3230 [PSH, ACK] Seq=18860529 Ack=2142900183 Win=23173 Len=54
     10 1064.209946 140.107.241.49        140.107.70.55         TCP      3230 > 1720 [RST] Seq=2142900183 Ack=18860529 Win=0 Len=0
     11 1064.210251 140.107.70.55         140.107.241.49        TCP      1720 > 3230 [FIN, ACK] Seq=18860583 Ack=2142900183 Win=23173 Len=0
     12 1064.210269 140.107.241.49        140.107.70.55         TCP      3230 > 1720 [RST] Seq=2142900183 Ack=18860583 Win=0 Len=0

Frame 8 (60 bytes on wire, 60 bytes captured)

Ethernet II, Src: 00:07:84:7a:ab:fc, Dst: 00:e0:db:01:94:b7
Internet Protocol, Src Addr: 140.107.241.49 (140.107.241.49), Dst Addr: 140.107.70.55 (140.107.70.55)
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
    Total Length: 40
    Identification: 0xd85c (55388)
    Flags: 0x00
    Fragment offset: 0
    Time to live: 55
    Protocol: TCP (0x06)
    Header checksum: 0x5b34 (correct)
    Source: 140.107.241.49 (140.107.241.49)
    Destination: 140.107.70.55 (140.107.70.55)
Transmission Control Protocol, Src Port: 3230 (3230), Dst Port: 1720 (1720), Seq: 2142900183, Ack: 18860529, Len: 0
```

```
Frame 10 (60 bytes on wire, 60 bytes captured)
Ethernet II, Src: 00:07:84:7a:ab:fc, Dst: 00:e0:db:01:94:b7
Internet Protocol, Src Addr: 140.107.241.49 (140.107.241.49), Dst Addr: 140.107.70.55 (140.107.70.55)
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
    Total Length: 40
    Identification: 0x5d87 (23943)
    Flags: 0x00
    Fragment offset: 0
    Time to live: 58
    Protocol: TCP (0x06)
    Header checksum: 0xd309 (correct)
    Source: 140.107.241.49 (140.107.241.49)
    Destination: 140.107.70.55 (140.107.70.55)
Transmission Control Protocol, Src Port: 3230 (3230), Dst Port: 1720 (1720), Seq: 2142900183, Ack: 18860529, Len: 0
```

CheckPoint Bug

Here again we see the initiating Polycom, 140.107.241.49 (located in the SCCA), build a TCP connection with its partner, 140.107.70.55 (located in the Hutch) using the three-way TCP handshake in packets 1-3. Notice that the last ACK, during call-setup, from 140.107.241.49 occurs at packet 8 and declares to its partner that its receive buffer has reached byte 179377714 (`Seq = 179377714`).

The next time 140.107.241.49 sends a frame is in packet 9, approximately 28 minutes after the beginning of the call, and here it declares that its receive buffer has reached byte 179377713 (`Seq = 179377713`), i.e. one less than the byte count it declared in packet 8. However, this time, its partner doesn't seem to understand this use of a decremented ACK and responds instead with a TCP RST, in packet 10. As with the PIX Bug case, a sniffer located next to 140.107.70.55 does not show this station producing the RST … though it does show 140.107.241.49 sending a RST. Once again, analysis of the TTL in the relevant packets (Packet 7 and Packet 10 in this trace and their equivalents in the trace taken by the sniffer next to 140.107.70.55[8]) suggests that an intermediate box has spoofed the TCP RSTs. Analysis of our network work and counting hops points to the Hutch firewalls as the source of these RSTs.

```
   No. Time        Source                Destination           Protocol Info

     1 0.000000    140.107.241.49        140.107.70.55         TCP      3230 > 1720 [SYN] Seq=179377526 Ack=0 Win=23360 Len=0
     2 0.001948    140.107.70.55         140.107.241.49        TCP      1720 > 3230 [SYN, ACK] Seq=141921921 Ack=179377527 Win=23360 Len=0
     3 0.002835    140.107.241.49        140.107.70.55         TCP      3230 > 1720 [ACK] Seq=179377527 Ack=141921922 Win=23360 Len=0
     4 0.052635    140.107.241.49        140.107.70.55         TCP      3230 > 1720 [PSH, ACK] Seq=179377527 Ack=141921922 Win=23360 Len=187
     5 0.119429    140.107.70.55         140.107.241.49        TCP      1720 > 3230 [PSH, ACK] Seq=141921922 Ack=179377714 Win=23173 Len=106
```

[8] This trace is not shown here.

```
      6 0.393842    140.107.241.49        140.107.70.55         TCP      3230 > 1720 [ACK] Seq=179377714 Ack=141922028 Win=23254 Len=0
      7 0.509399    140.107.70.55         140.107.241.49        TCP      1720 > 3230 [PSH, ACK] Seq=141922028 Ack=179377714 Win=23173 Len=145
      8 0.793823    140.107.241.49        140.107.70.55         TCP      3230 > 1720 [ACK] Seq=179377714 Ack=141922173 Win=23109 Len=0
      9 1738.600399 140.107.241.49        140.107.70.55         TCP      3230 > 1720 [ACK] Seq=179377713 Ack=141922173 Win=23109 Len=0
     10 1738.600405 140.107.70.55         140.107.241.49        TCP      1720 > 3230 [RST, ACK] Seq=141922173 Ack=179377713 Win=23109 Len=0
```

Frame 7 (199 bytes on wire, 199 bytes captured)

```
Ethernet II, Src: 00:02:7e:23:8b:c0, Dst: 00:e0:db:02:b5:78
Internet Protocol, Src Addr: 140.107.70.55 (140.107.70.55), Dst Addr: 140.107.241.49 (140.107.241.49)
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
    Total Length: 185
    Identification: 0x8db1 (36273)
    Flags: 0x00
    Fragment offset: 0
    Time to live: 55
    Protocol: TCP (0x06)
    Header checksum: 0xa54e (correct)
    Source: 140.107.70.55 (140.107.70.55)
    Destination: 140.107.241.49 (140.107.241.49)
Transmission Control Protocol, Src Port: 1720 (1720), Dst Port: 3230 (3230), Seq: 141922028, Ack: 179377714, Len: 145
```

Frame 10 (60 bytes on wire, 60 bytes captured)

```
Ethernet II, Src: 00:02:7e:23:8b:c0, Dst: 00:e0:db:02:b5:78
Internet Protocol, Src Addr: 140.107.70.55 (140.107.70.55), Dst Addr: 140.107.241.49 (140.107.241.49)
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
    Total Length: 40
    Identification: 0x5e7a (24186)
    Flags: 0x00
    Fragment offset: 0
    Time to live: 58
    Protocol: TCP (0x06)
    Header checksum: 0xd216 (correct)
    Source: 140.107.70.55 (140.107.70.55)
    Destination: 140.107.241.49 (140.107.241.49)
Transmission Control Protocol, Src Port: 1720 (1720), Dst Port: 3230 (3230), Seq: 141922173, Ack: 179377713, Len: 0
```