Improving TCP/IP Security Through Randomization Without Sacrificing Interoperability

Michael J. Silbersack

5/13/2006

http://www.silby.com/bsdcan06/

What does that title mean?

- TCP was not designed with an eye towards security
- There are many attacks against TCP which can be prevented without resorting to encryption or keyed hashes
- Sometimes the obvious fix to a TCP security problem leads to interoperability problems

Topics to discuss

- IP ID values
- Ephemeral Port Randomization
- TCP Initial Sequence Numbers
- TCP Timestamps

IP ID Values

- IP ID values are used for the purpose of IP fragment reassembly
- If IP ID values are repeated too quickly, two different packets can be reassembled together, creating a corrupt packet
- Operating systems traditionally use a single system-wide counter which increments by one for each packet sent
- This leaks information about a host's level of traffic and a host's identity

IP ID Fixes

- Use a ID value of 0 on fragments with the DF (don't fragment) bit set
  - Tried by Linux, some firewalls / NAT machines were found to strip DF bits, causing a stream of fragmented packets that all had the same ID value
- Store per-IP state and use a separate counter for each IP (Linux)
- Use a LCG to generate psuedo-random ID values that have a relatively long time between repeats (OpenBSD)

IP ID fixes - simpler

- At EuroBSDCon I stated that the danger of quickly repeated ID values has been overstated
- I have been informed that empirical tests have shown that the TCP/UDP checksum is not strong enough to detect all cases of corruption if two packets with the same IP ID are reassembled together.

TCP Connections: A Quick Review

- A TCP connection is identified by a 4-tuple:
  - Source IP
  - Source Port
  - Destination IP
  - Destination Port
- The destination port is usually a well known port such as port 80 on a web server
- The source port is usually chosen from the ephemeral port range by the operating system

TCP Sequence Numbers

- TCP uses 32-bit sequence numbers to track how much data has been transmitted
  - The SYN and FIN flags also count as a byte in the data stream
- Each direction's sequence number is independent, and is chosen by the operating system at that end of the connection
- A sliding window is used, typically around 32K in size. Packets with sequence numbers that fall into this window are accepted.

A Sample Connection

```
IP 10.1.1.9.65500 > 10.1.1.237.80: S 2766364594:2766364594(0) win 65535 <mss
     1460,sackOK,wscale 1,timestamp 146016542 0>
IP 10.1.1.237.80 > 10.1.1.9.65500: S 4027082585:4027082585(0) ack 2766364595
     win 5792 <mss 1460,sackOK,timestamp 80799562 146016542,wscale 2>
IP 10.1.1.9.65500 > 10.1.1.237.80: . ack 4027082586 win 33304
     <timestamp 146016542 80799562>
IP 10.1.1.9.65500 > 10.1.1.237.80: P 2766364595:2766364664(69) ack 4027082586
      win 33304 <timestamp 146016542 80799562>
IP 10.1.1.237.80 > 10.1.1.9.65500: . ack 2766364664 win 1448 <timestamp
     80799563 146016542>
IP 10.1.1.237.80 > 10.1.1.9.65500: P 4027082586:4027083050(464) ack
     2766364664 win 1448 <timestamp 80799565 146016542>
IP 10.1.1.9.65500 > 10.1.1.237.80: F 2766364664:2766364664(0) ack 4027083050
     win 33304 <timestamp 146016542 80799565>
IP 10.1.1.237.80 > 10.1.1.9.65500: F 4027083050:4027083050(0) ack 2766364665
     win 1448 <timestamp 80799566 146016542>
IP 10.1.1.9.65500 > 10.1.1.237.80: . ack 4027083051 win 33303
     <timestamp 146016542 80799566>
```

Cases of Security Breaking Interoperability

- Implementation of OpenBSD ISN scheme in FreeBSD
- Implementation of zeroed IP ID values in Linux
- Implementation of port randomization in FreeBSD

Ephemeral Port Randomization

- Ephemeral ports have traditionally been allocated in a sequential fashion, making it easy for an attacker to figure out the next port to be used
  - One positive property of this behavior is that the period of time before ephemeral port reuse was maximized
- Ephemeral port randomization makes spoofing attacks more difficult, nearly 2^16 times more difficult if a large ephemeral port space is used
  - But as a result, ports can be reused a few milliseconds later

Port Randomization Problems

- After FreeBSD enabled port randomization, one user with a FreeBSD machine running squid in front of a FreeBSD machine running Apache started to notice that some connections were failing
- Disabling port randomization solved the problem for him
- One of the failure cases was caught; a port was being reused in 3ms

One Troubled Connection

```
23.606609 Client > Server: S 1670850402:1670850402(0)
23.606730 Server > Client: S 1392685077:1392685077(0) ack 1670850403
23.606742 Client > Server: . ack 1392685078
23.606751 Client > Server: P 1670850403:1670850611(208) ack 1392685078
23.609936 Server > Client: . 1392685078:1392686526(1448) ack 1670850611
23.609938 Server > Client: P 1392686526:1392687580(1054) ack 1670850611
23.609939 Server > Client: F 1392687580:1392687580(0) ack 1670850611
23.609957 Client > Server: . ack 1392687580
23.609960 Client > Server: . ack 1392687581
23.609995 Client > Server: F 1670850611:1670850611(0) ack 1392687581
23.610440 Server > Client: . ack 1670850612
23.641734 Client > Server: S 1670903298:1670903298(0)
23.641931 Server > Client: R 1392687581:1392687581(0) ack 1670850612
23.641939 Server > Client: R 0:0(0) ack 1670903299
```

Port Randomization Problems Continued

- The glitch is a bug in the FreeBSD TCP stack – but it is one that would never happen without port randomization
  - May be due to the sequence number landing within the previous connection's window; a check that should not be running for TIME_WAIT sockets
- Do other operating systems have lingering bugs like this that port randomization will expose?
- For now, FreeBSD turns off port randomization when the connection rate exceeds a certain threshold

Classes of Initial Sequence Numbers

- Time based – specified in RFC 793
  - Compatible, but very insecure
- Random Positive Increments
  - Compatible, with slightly better security
- Random
  - Secure, but incompatible
- RFC 1948
  - A good compromise between the two

IP Spoofing

- An exact guess at the ISN in a SYN-ACK allows you to spoof a connection
- As you can only send data, this can be used to attack rsh/rlogin with IP-based authentication
  - Sending anonymous e-mails and other types of attacks should be possible as well
- This attack was easy when time-based sequence numbers were used
- Random positive increments make this attack more difficult, but not impossible

Connection corruption

- Attacks well described in “Slipping in the Window” by Paul Watson
- The following attacks work because TCP stacks generally accept packets that have a seq # value that is anywhere in the sliding window
  - RST attacks
  - SYN attacks
  - Data injection attacks

How to defeat these attacks

- Ensure that the sequence numbers of each connection are entirely independent of one another
  - Attackers will have to spoof the entire sequence space
- Implement the countermeasures described in tcpsecure so that not just any sequence number in the window is accepted

Interoperability concerns

- Initial sequence numbers can be randomized...
  - Except when the same 4-tuple is reused within a short period of time
- Theoretical reasoning: If the same 4-tuple is reused and the same sequence space is overlapped, old duplicate packets may corrupt the connection
- Practical reason: TIME_WAIT socket recycling rules

The Time Wait State

- During a normal TCP socket close, the side of the connection that starts to close the connection will enter the time wait state for two minutes (RFC 793)
- The purpose of the time wait state is to ignore any old (or duplicate) packets still in the network
- BSD-derived TCP/IP stacks will recycle a TIME_WAIT socket only if the ISN in the SYN packet is greater than the sequence number at the end of the previous connection

Empirical TIME_WAIT recycling results

- In order to verify the monotonically increasing sequence number requirement, a FreeBSD machine was modified so that it would generate monotonically *decreasing* sequence numbers
- The results showed types of behavior that were not expected

Empirical TIME_WAIT results

- Cisco IOS 12.3: All connections accepted
- FreeBSD: All connections delayed / failed
- Linux 2.6.11-FC4: All connections accepted due to timestamp heuristic & tcpsecure behavior also implemented
- NetBSD 2.0.2: tcpsecure behavior
- OpenBSD 3.7: tcpsecure behavior
- Windows XP SP2: All connections delayed / failed

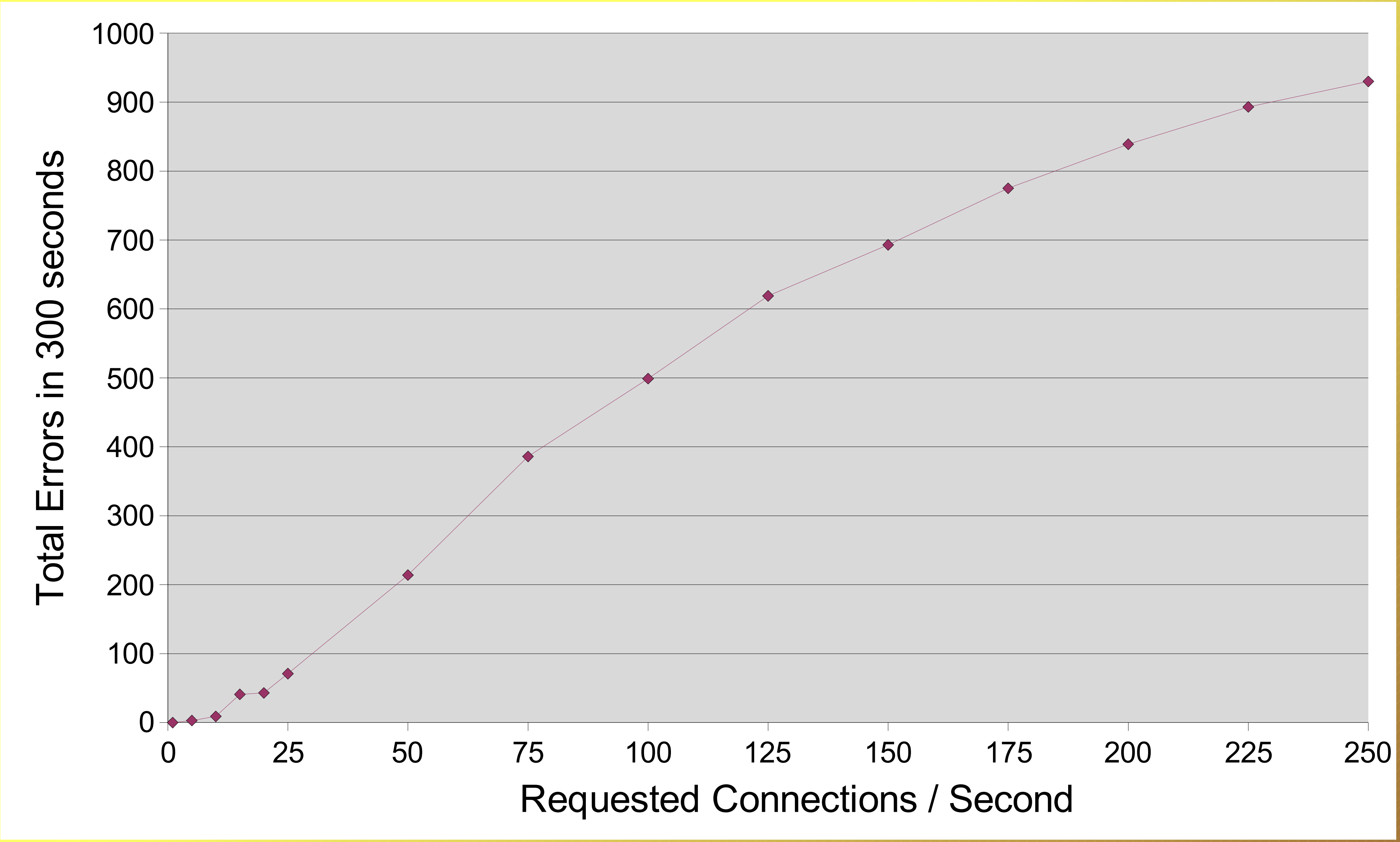
The tcpsecure Behavior

```
59.515622 IP Server > Client: F 993959099:993959099(0) ack 4086058688
59.515742 IP Client > Server: . ack 993959100
65.657308 IP Client > Server: S 4078507753:4078507753(0)
65.657610 IP Server > Client: . ack 4086058688
65.657741 IP Client > Server: R 4086058688:4086058688(0)
68.655831 IP Client > Server: S 4078507753:4078507753(0)
68.655914 IP Server > Client: S 2006422470:2006422470(0)
```

Connection Failures due to randomized SYN ISNs: test setup

- Server: FreeBSD 4.11
- Client: FreeBSD 6.1 with Randomized ISN patch, ephemeral port range 1024-65535
- Test tool: modified netrate, 25 threads, 5 second connection timeout
  - In reality, most of these connections would never have completed, but letting that happen would cause all 25 threads to get stuck waiting at times

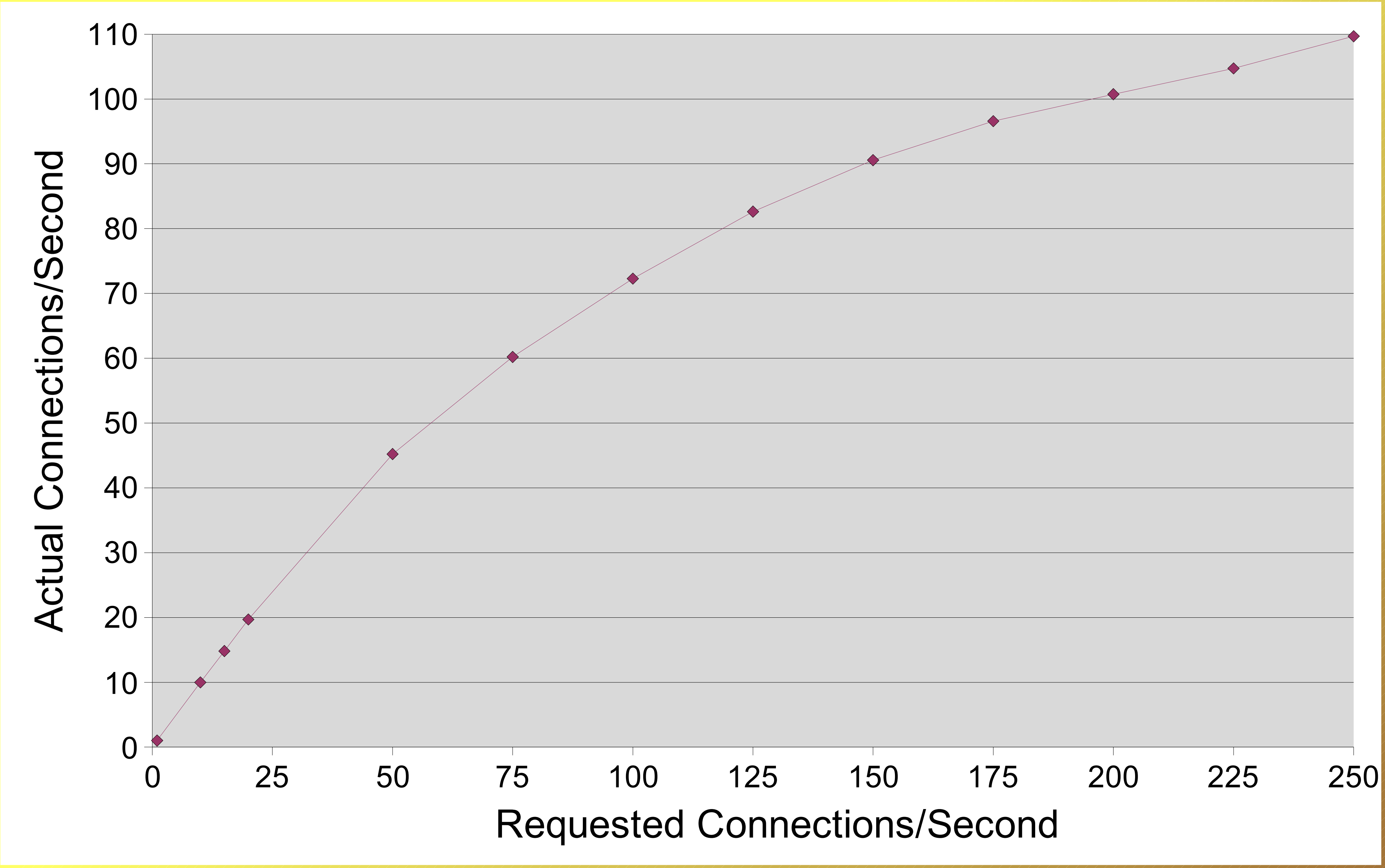
Connection Failures due to randomized SYN ISNs: results



Connection rate falloff due to randomized SYN ISNs: test setup

- Client: FreeBSD 6.1 with Randomized ISN patch, ephemeral port range 1024-65535
- Server: OpenBSD 3.8 Server
- Test tool: modified netrate, 25 threads, 50 second connection timeout
  - No errors were reported during this test
- Caveat: Test not run with normal sequence numbers

Connection rate falloff due to randomized SYN ISNs: results



Who is negatively affected by randomized SYN ISNs?

- Benchmarking tools
- Front-end web caches
- NAT boxes who have clients who all visit the same sites

RFC 1948

- Steven Bellovin describes a near-perfect solution to this problem in RFC 1948
- A system-wide secret is generated and stored at boot time
- A system-wide time counter is incremented at a constant rate
- Initial sequence numbers are generated as follows:
- ISN = time + MD5(srcip, srcport, dstip, dstport, secret)

One Flaw In RFC 1948

- For a certain tuple, sequence numbers are fully predictable until the system reboots
- Example:
  - A SMTP server uses RFC 1948 for all ISNs
  - Spammer uses an AOL account to connect to that SMTP server, records ISN values
  - Spammer can now spoof connections from that AOL IP to the SMTP server until it reboots
- If the hash is rekeyed, then monotonicity is broken – so we can't fix it that way

TCP Security / Interoperability Summary

- For security purposes, sequence numbers must be unrelated to the sequence numbers of any other connection
- For interoperability purposes, ISNs in SYN packets must be monotonically increasing
  - If this principle is violated, connection establishment may stall whenever a TCP connection is reused
  - If port randomization is used, port reuse may be a common occurrence

A Sequence Number Survey

- Many ISN surveys have been done, but they generally do not consider
  - How RFC 1948 works
  - That OSes may generate SYN and SYN-ACK packets in different manners
- This survey focuses on a small range of ephemeral ports and watches how they behave

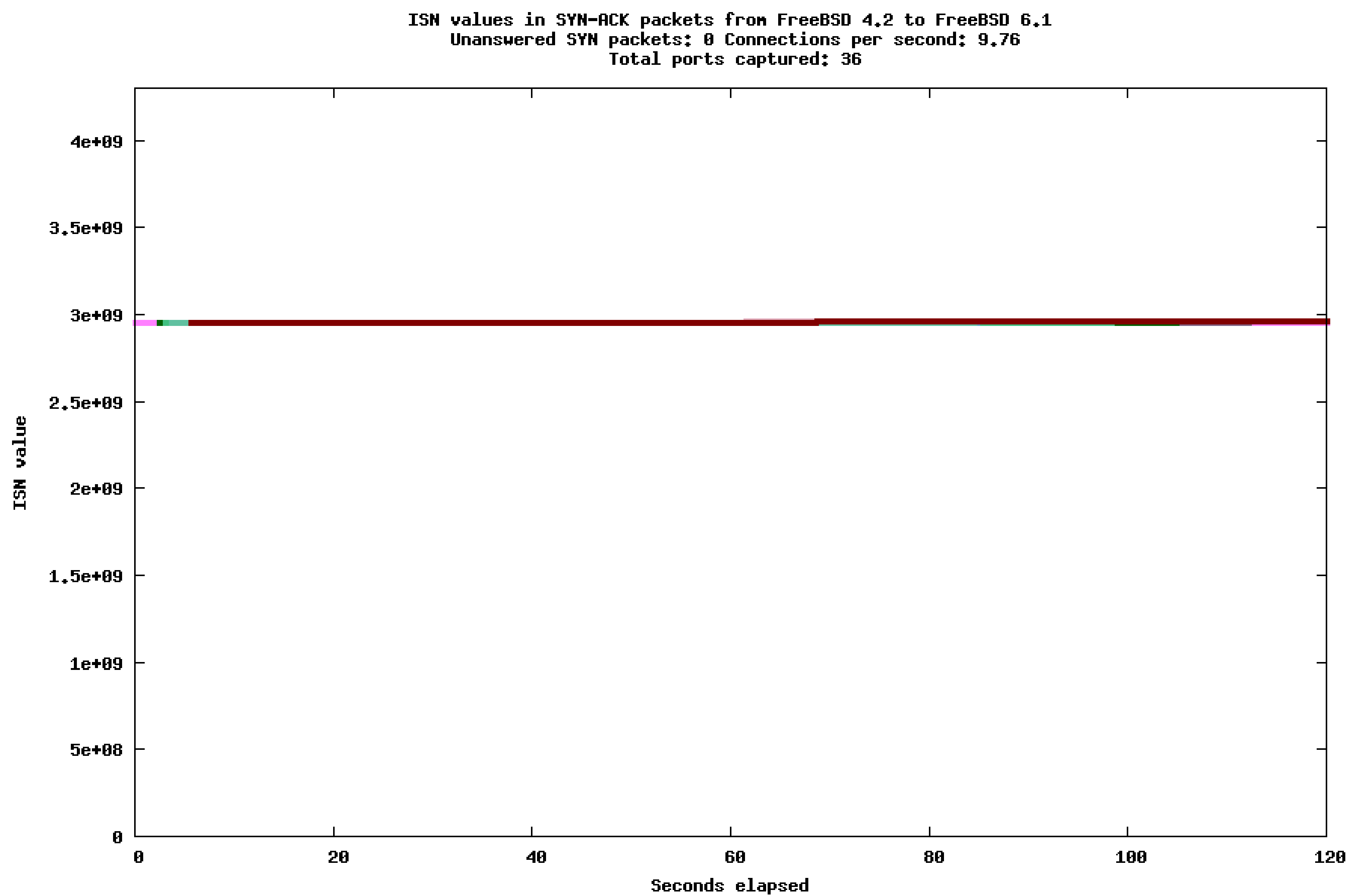
The Graphs

- The graphs you're about to see were generated by running a http benchmark utility against a web server
- Tests were run in both directions so that the ISN values in SYN and SYN-ACK packets could both be observed
- Each line is a series of initial sequence numbers captured in SYN / SYN-ACK packets for a certain sip:sport:dip:dport tuple

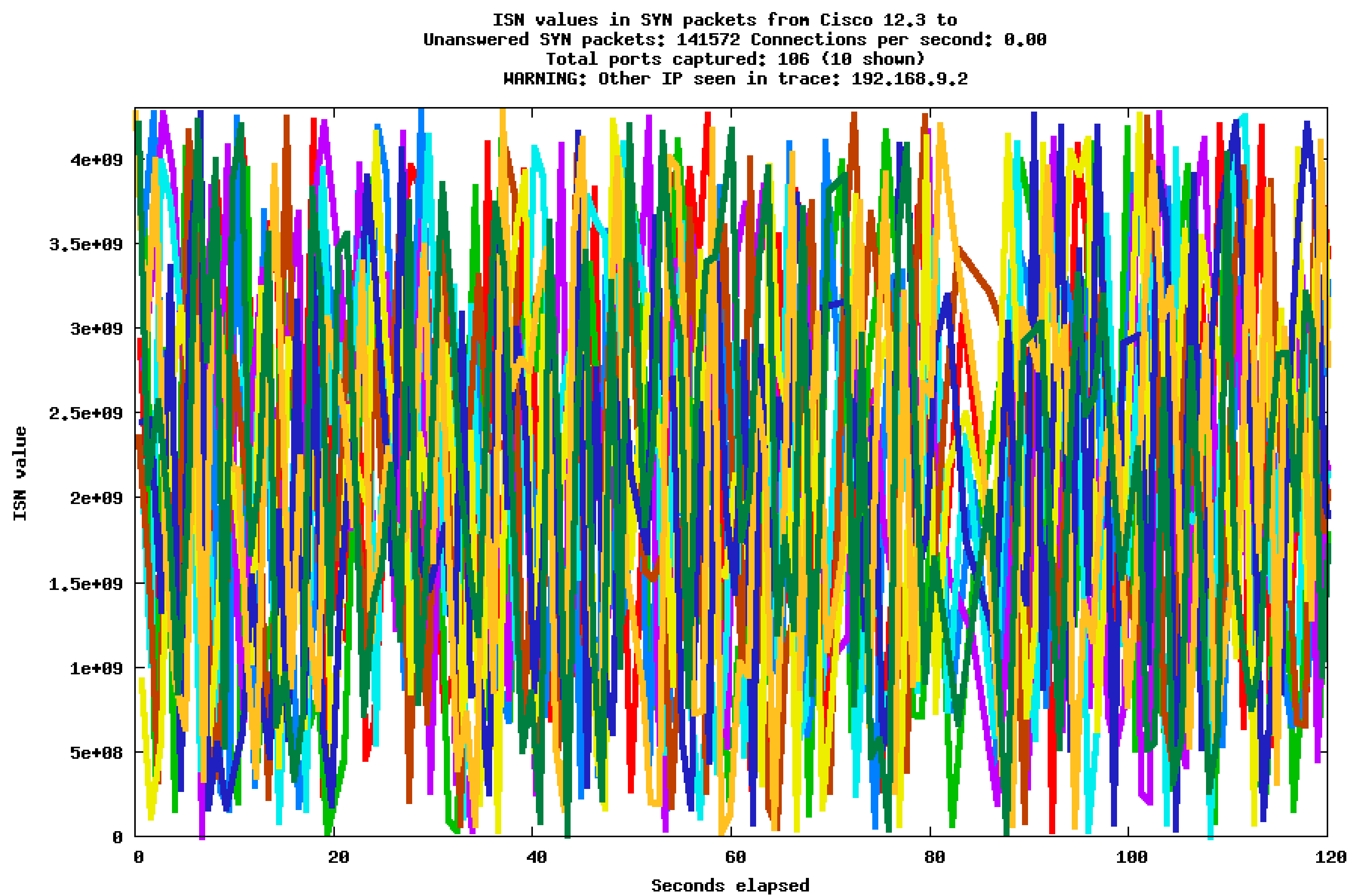
The Graphs (continued)

- Caveat 1: I used different http test tools, and didn't keep the connection rate the same during each test. This should not affect the results...
  - Except for random positive increments, which would change their slope based on the connection rate
- Caveat 2: For OSes that I do not have the source code to, the algorithm could be different than it appears to be.

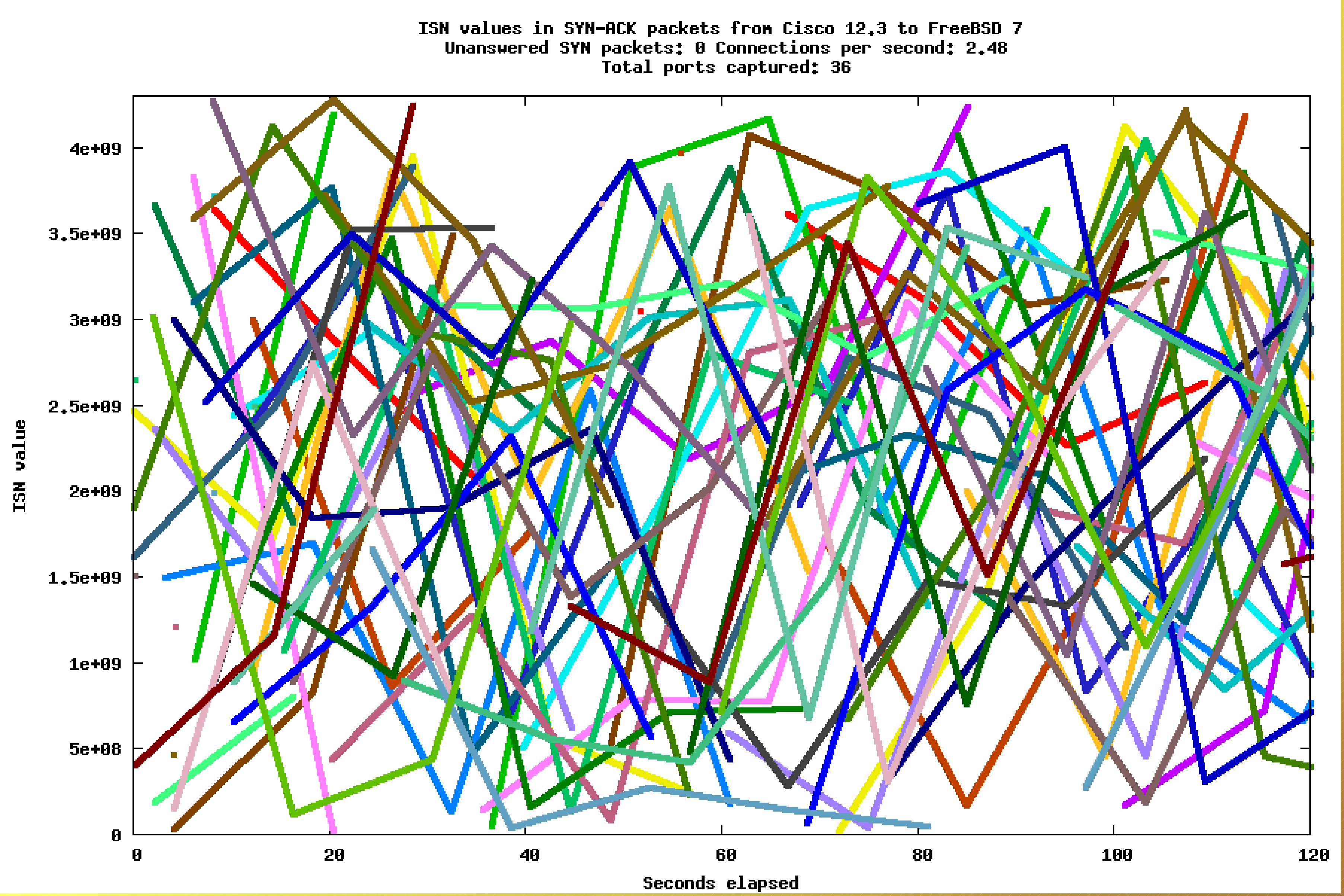
FreeBSD 4.2


ISN values in SYN-ACK packets from FreeBSD 4.2 to FreeBSD 6.1
Unanswered SYN packets: 0 Connections per second: 9.76
Total ports captured: 36
ISN value
4e+09
3.5e+09
3e+09
2.5e+09
2e+09
1.5e+09
1e+09
5e+08
0
0
20
40
60
80
100
120
Seconds elapsed

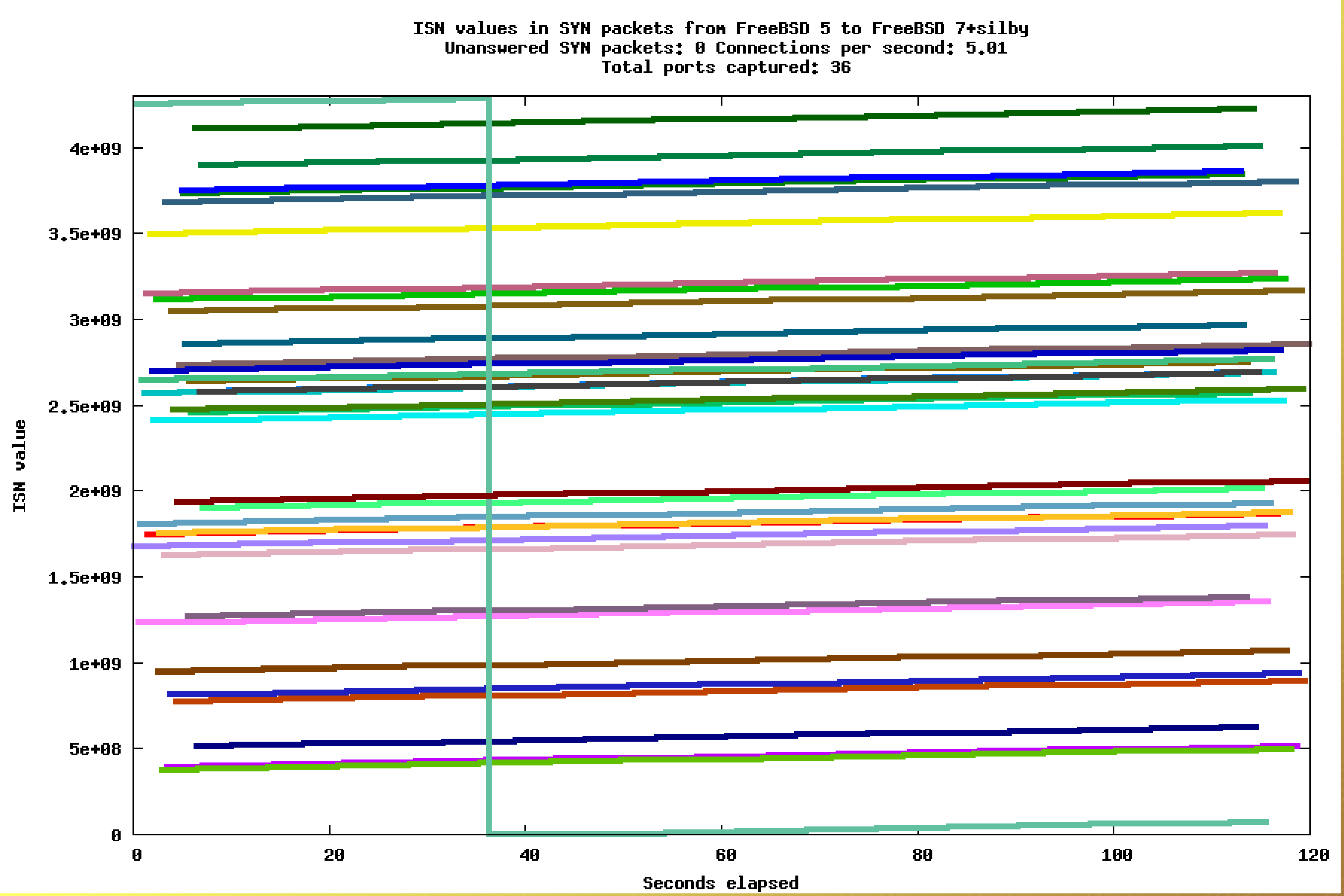
Cisco IOS 12.3 SYN



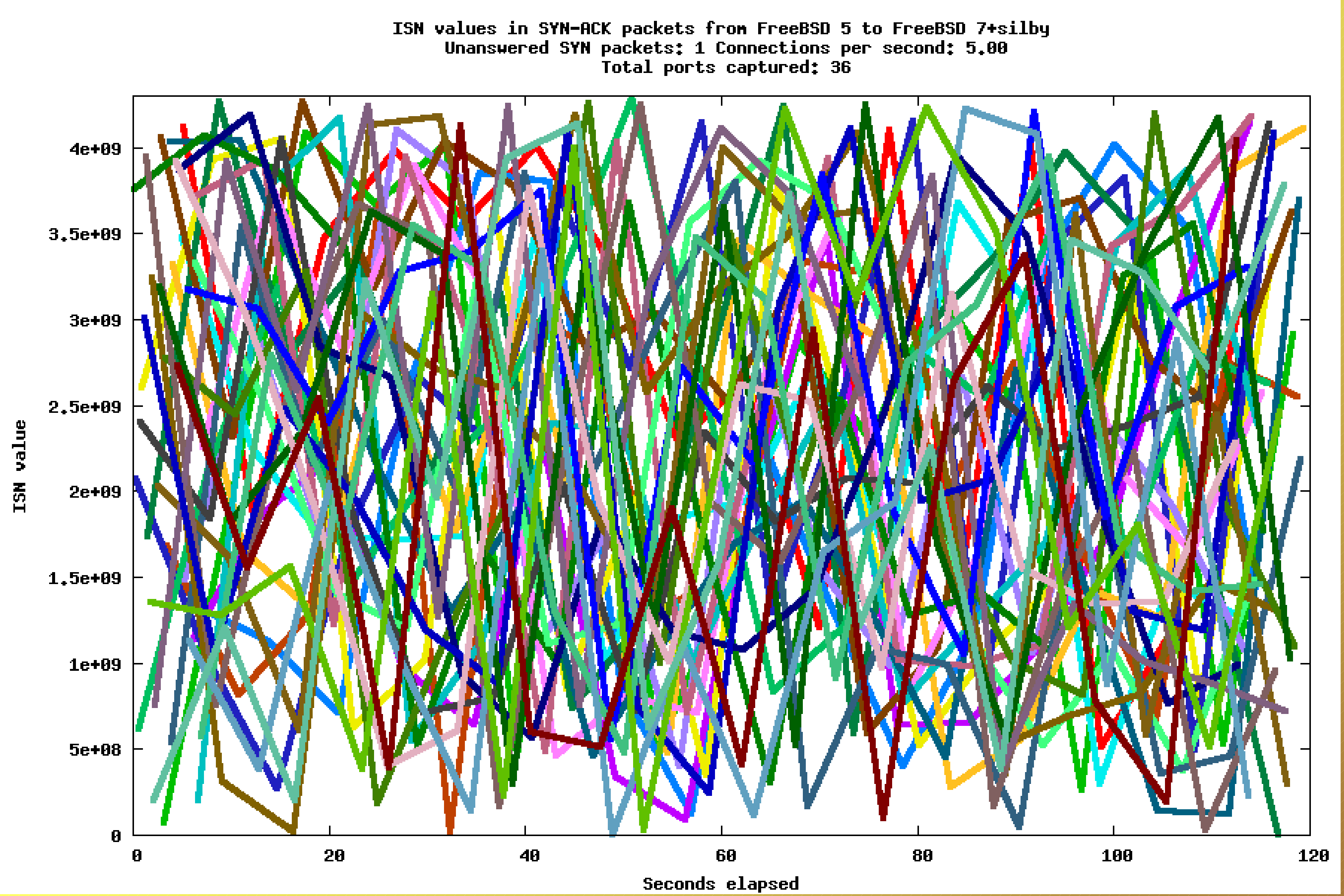
Cisco IOS 12.3 SYN-ACK



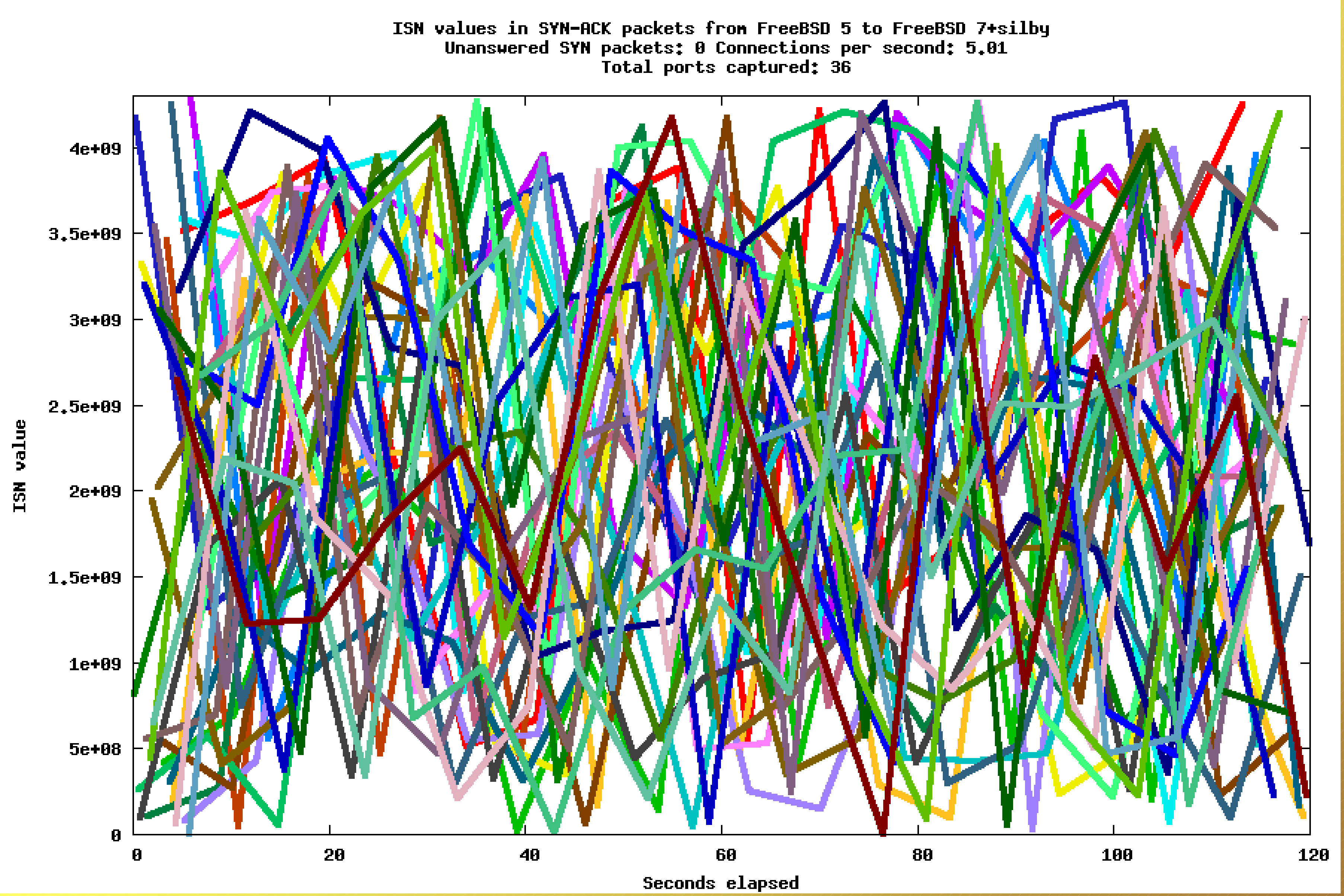
FreeBSD SYN


ISN values in SYN packets from FreeBSD 5 to FreeBSD 7+silby
Unanswered SYN packets: 0 Connections per second: 5.01
Total ports captured: 36
ISN value
4e+09
3.5e+09
3e+09
2.5e+09
2e+09
1.5e+09
1e+09
5e+08
0
0
20
40
60
80
100
120
Seconds elapsed

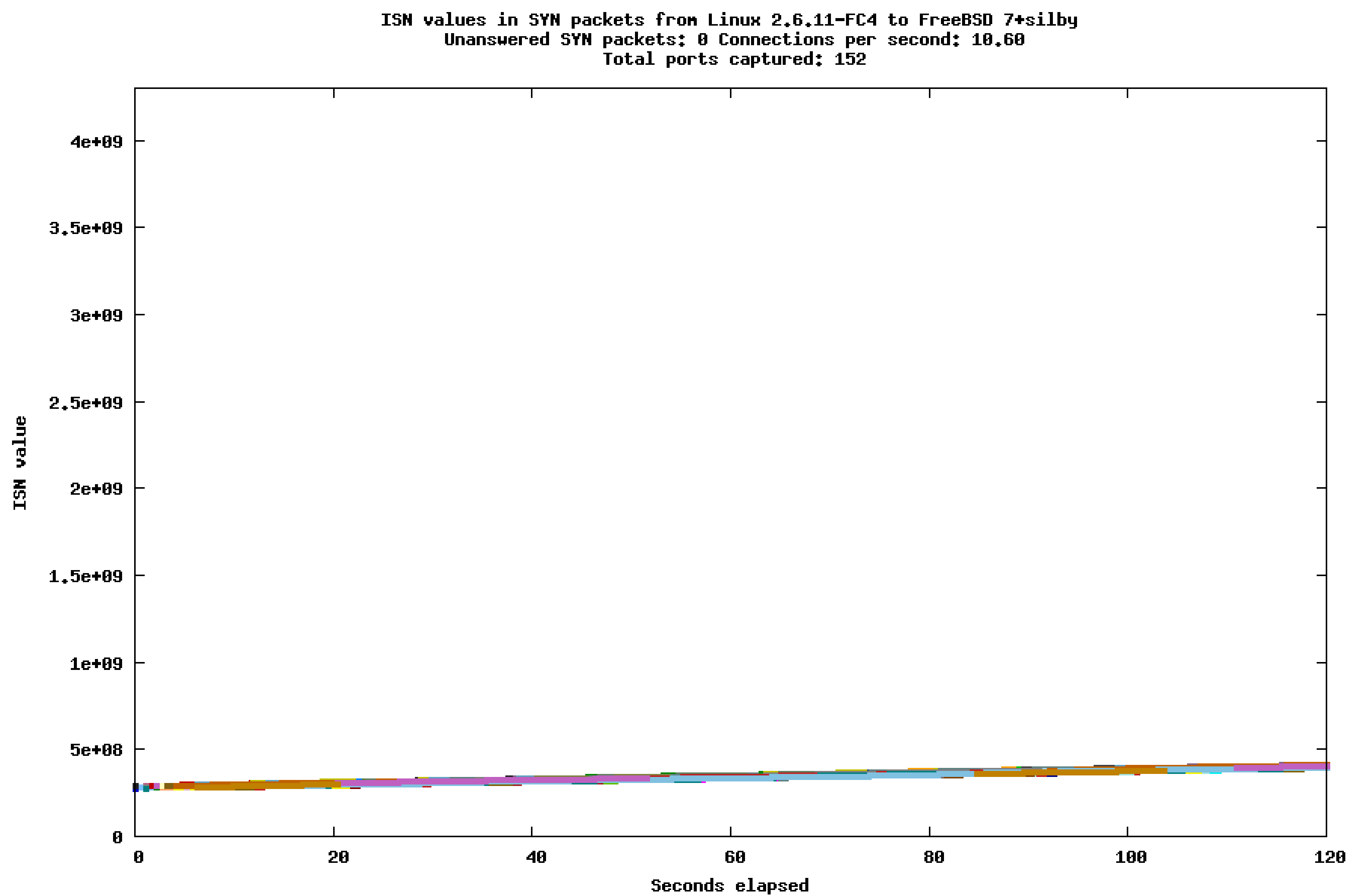
FreeBSD SYN-ACK (no cookies)



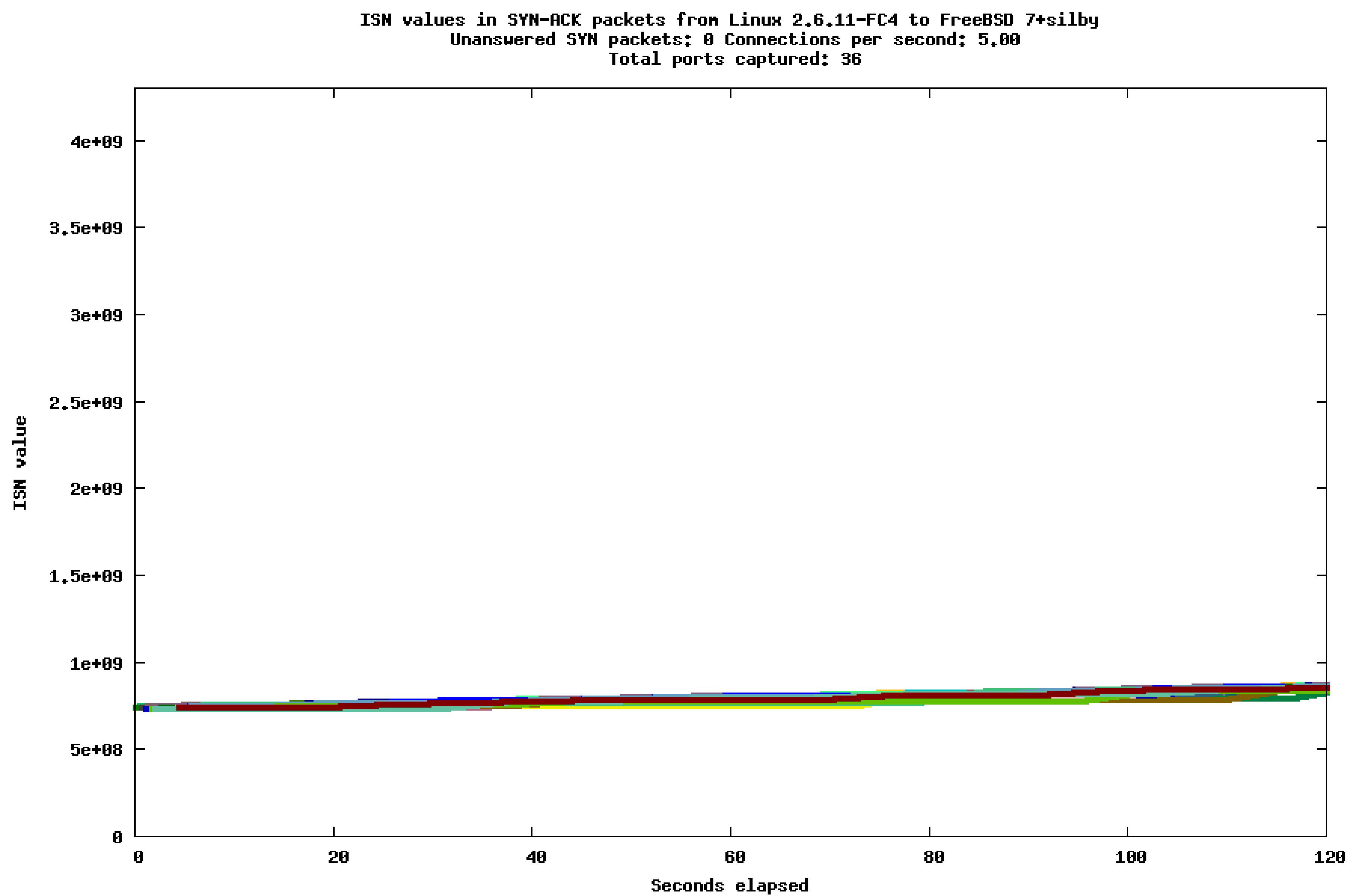
FreeBSD SYN-ACK (cookies!)



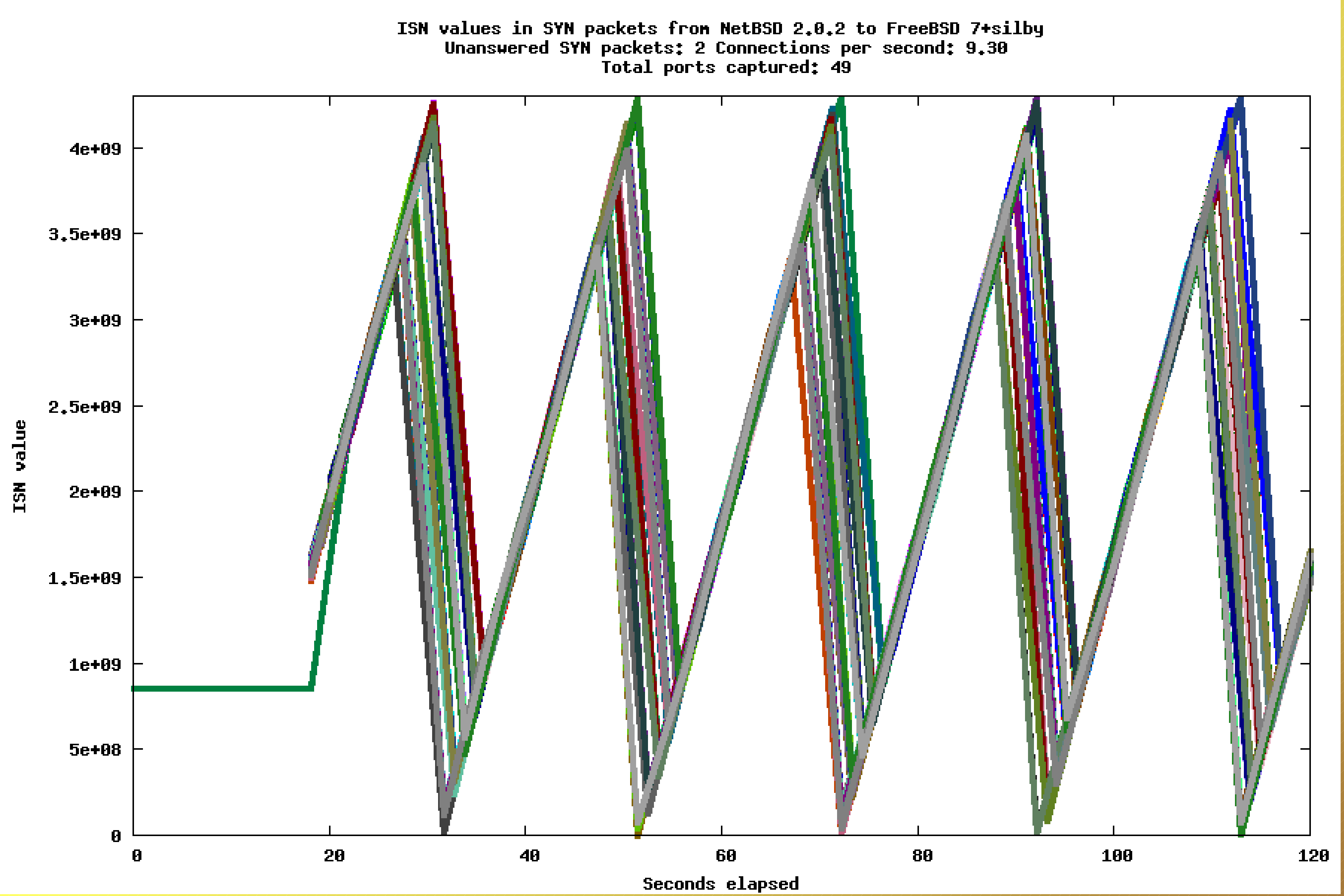
Linux 2.6.11-FC4 SYN



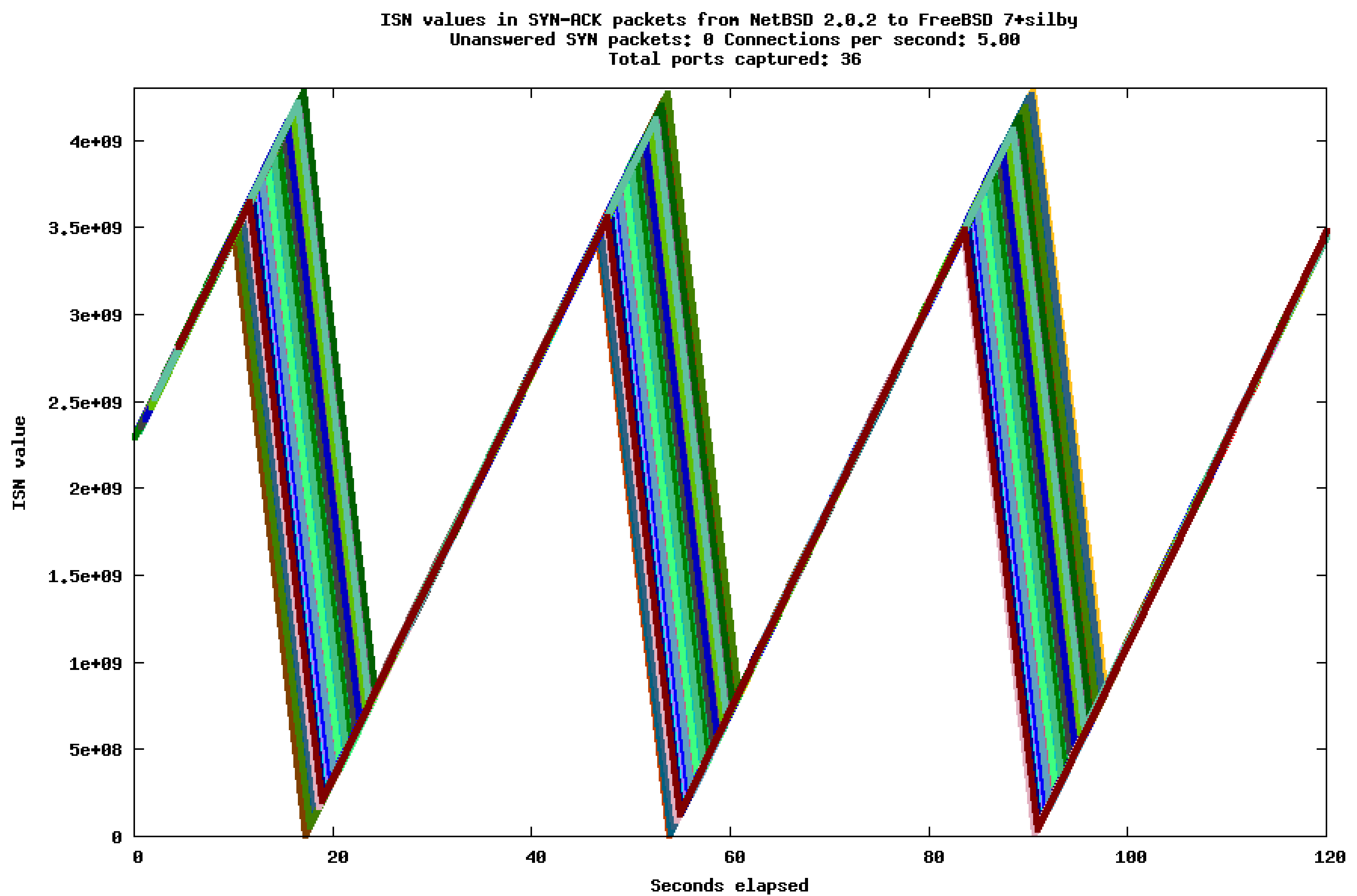
Linux 2.6.11-FC4 SYN-ACK



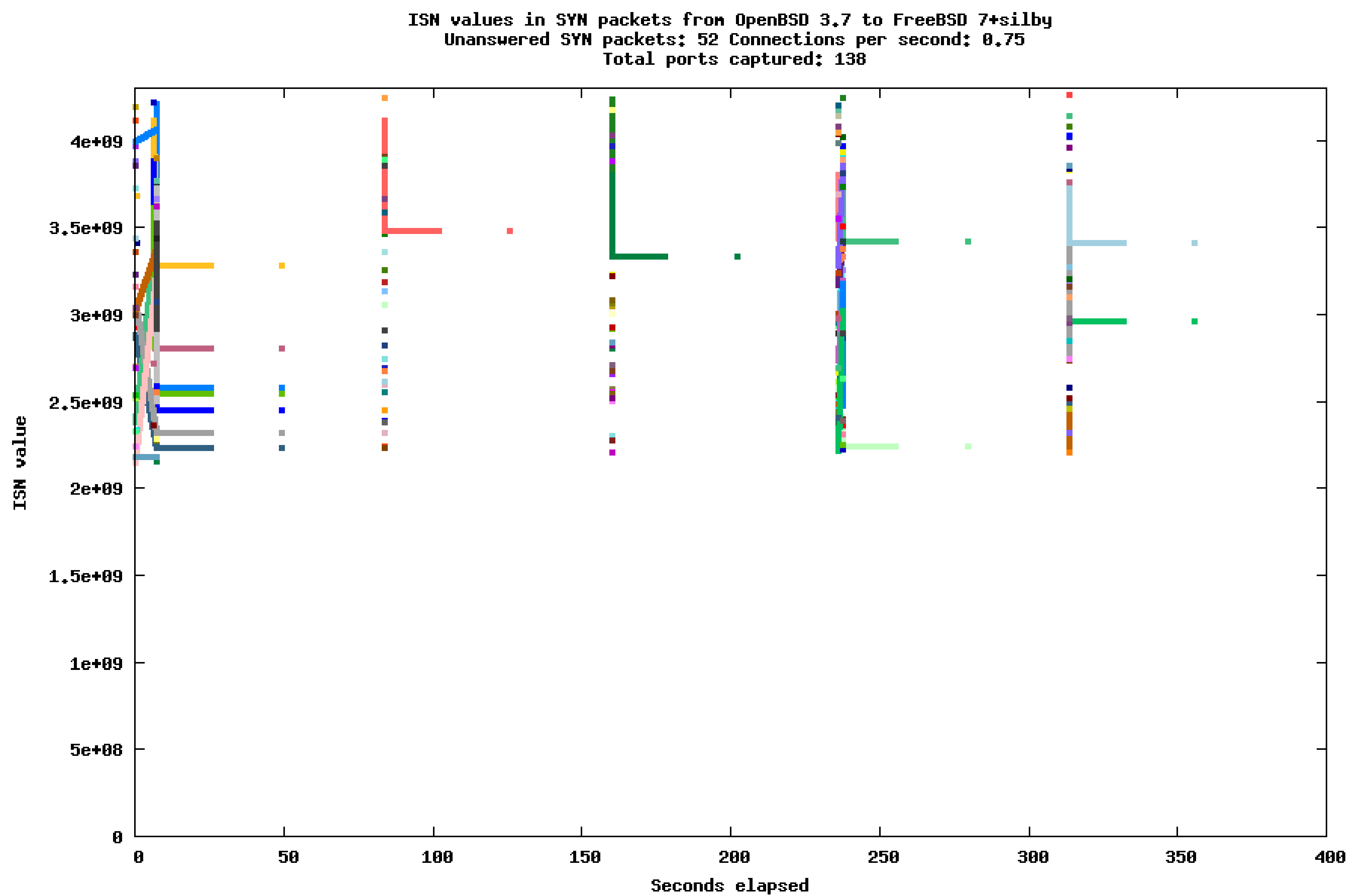
NetBSD 2.0.2 SYN



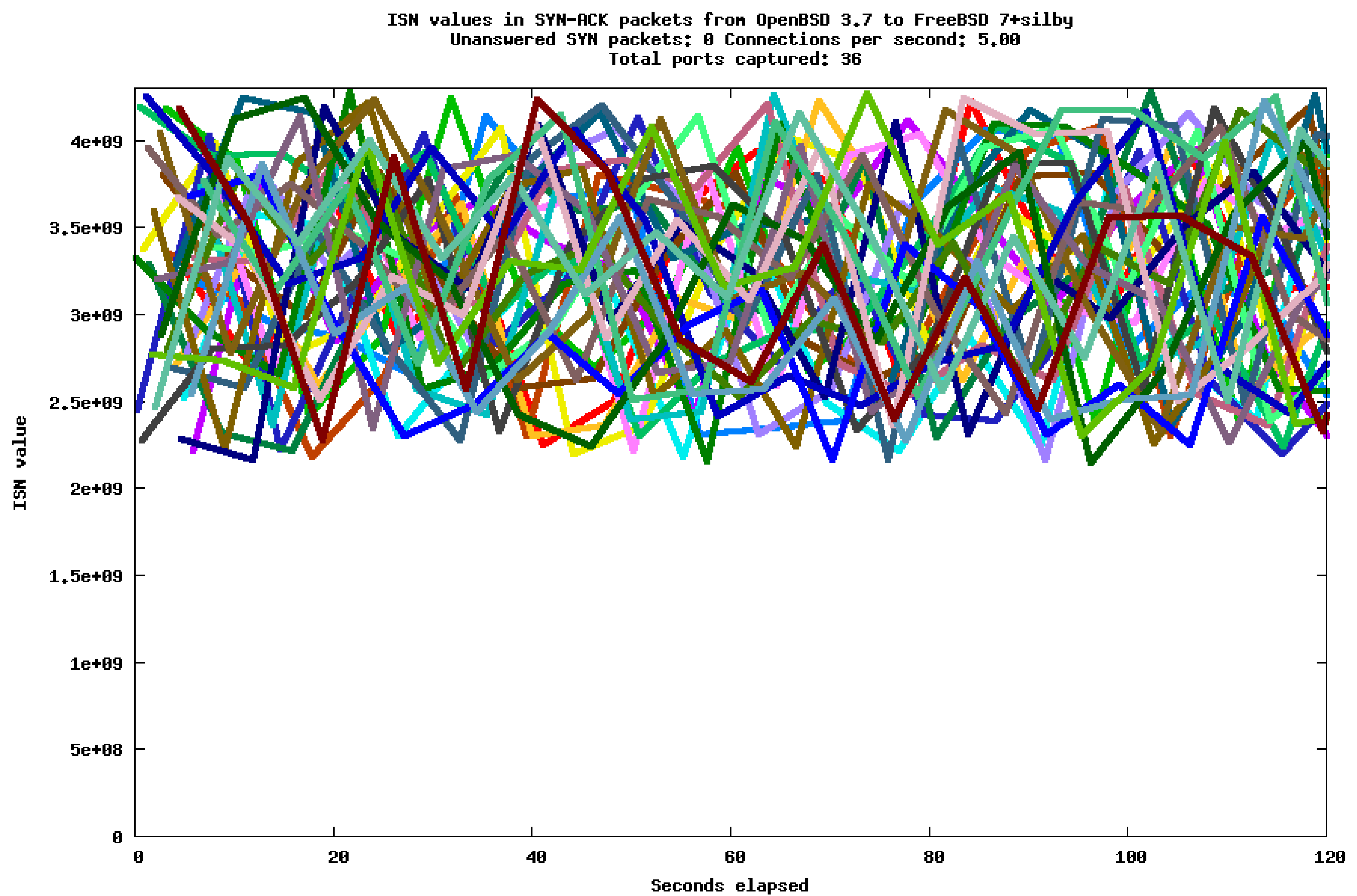
NetBSD 2.0.2 SYN-ACK



OpenBSD 3.7 SYN



OpenBSD 3.7 SYN-ACK



OpenBSD's algorithm

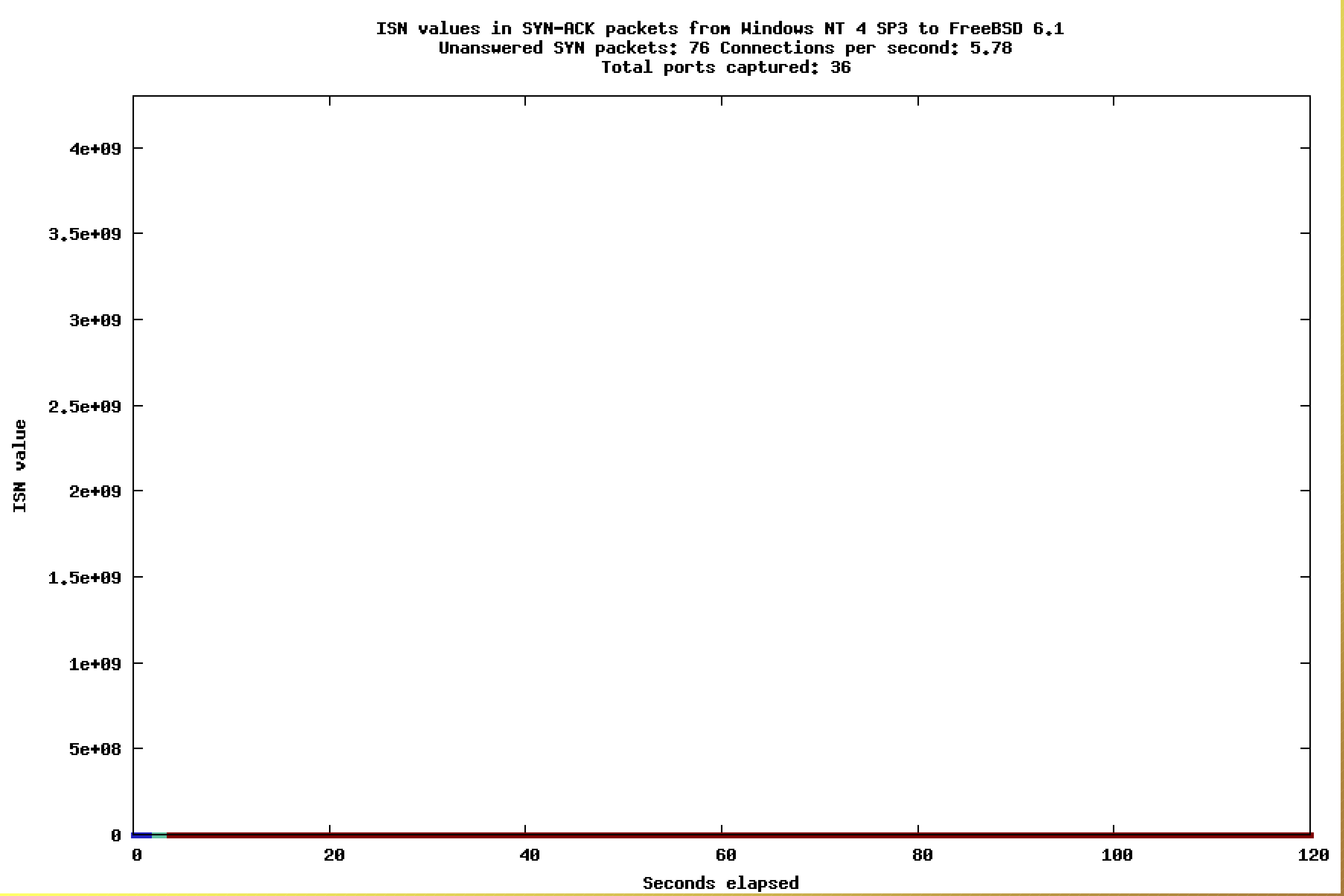
ISN = ((LCG(t)) << 16) + R(t)

LCG(t) = a 15 bit output from a LCG
R(t) = a 15-bit random number generator

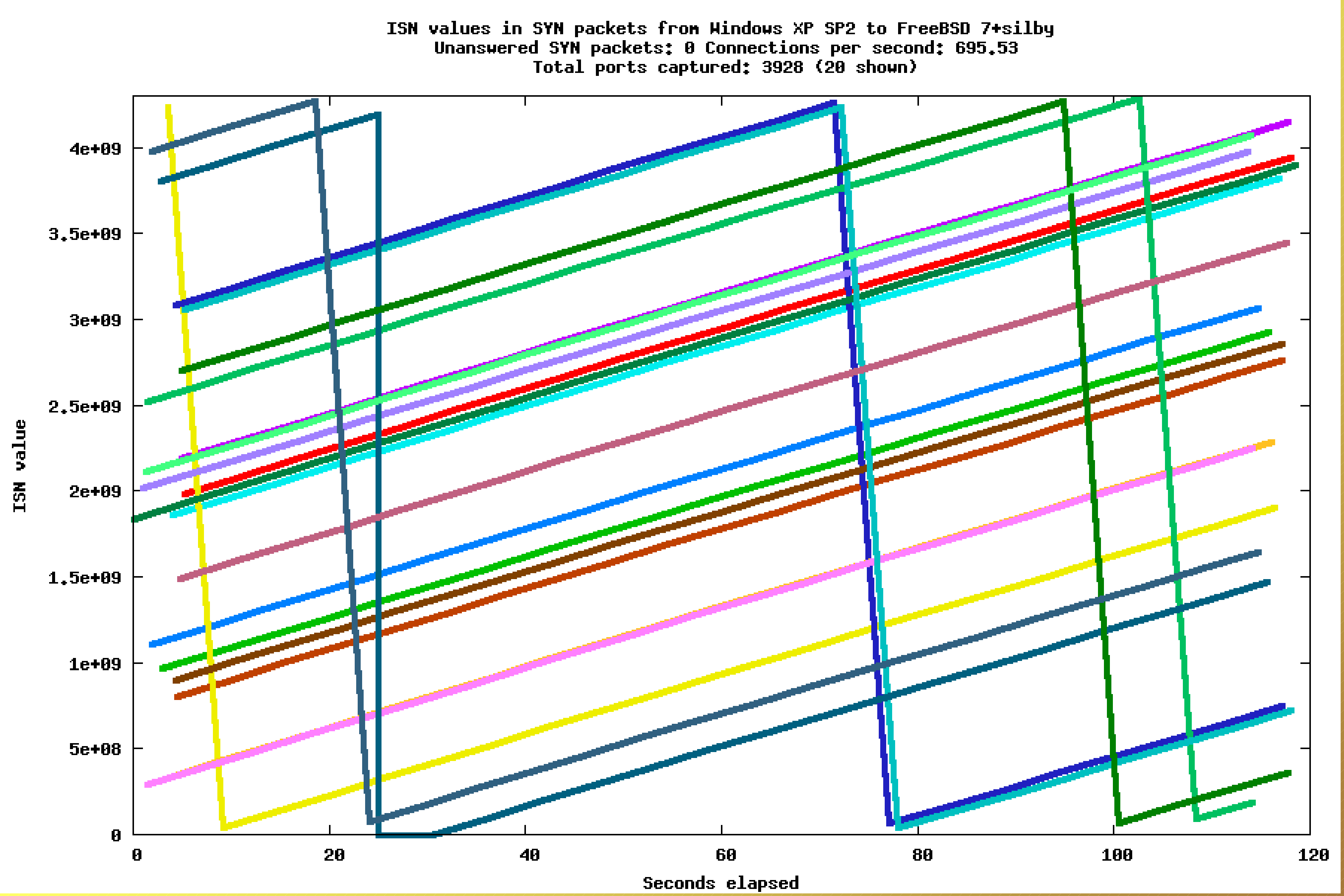
Bit 31 is toggled when the LCG is reset,
Bit 15 is always zero

The LCG is reset every 30000 uses or every 7200 seconds, whichever comes first

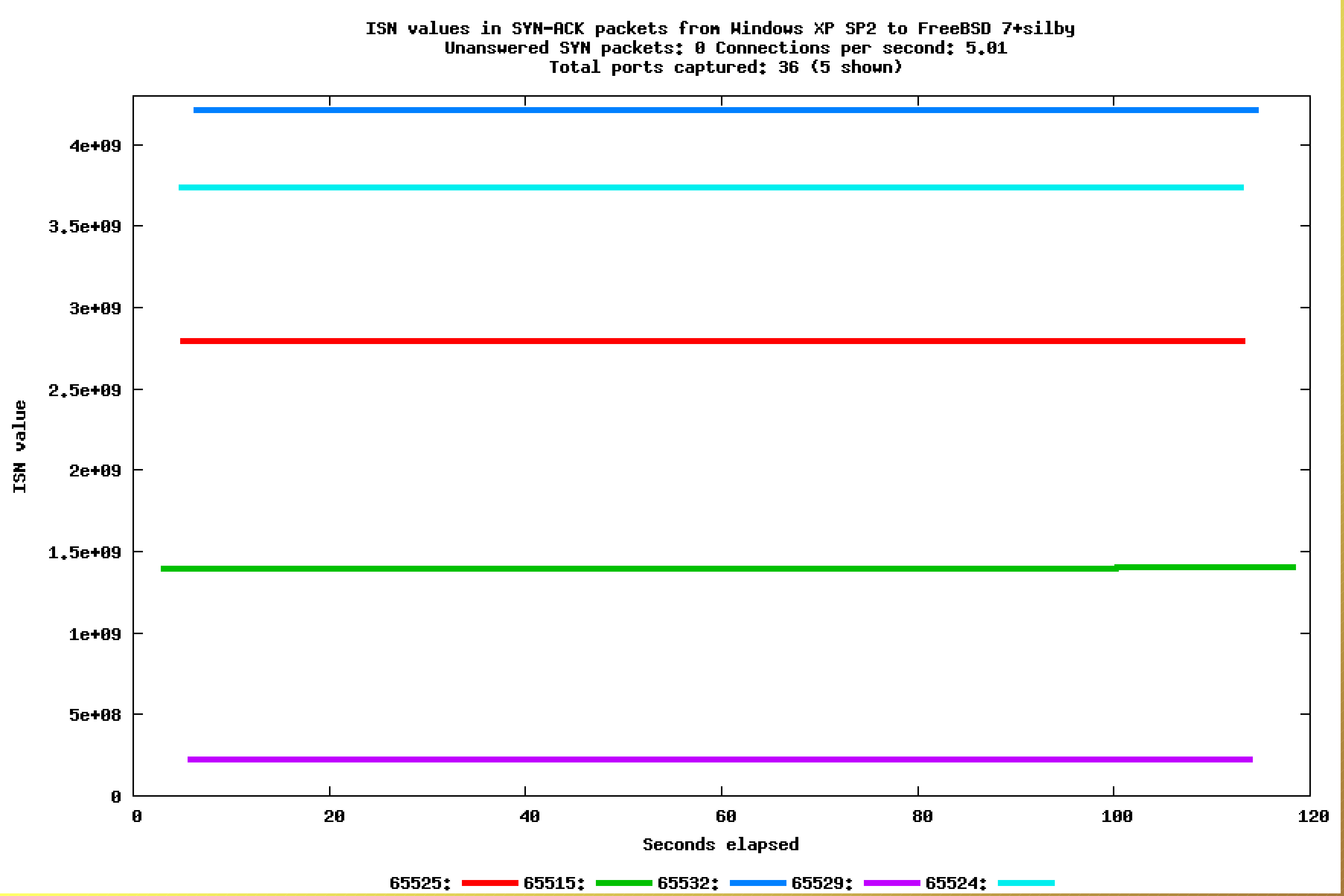
Windows NT 4 SP3



Windows XP SP2 SYN



Windows XP SP2 SYN-ACK



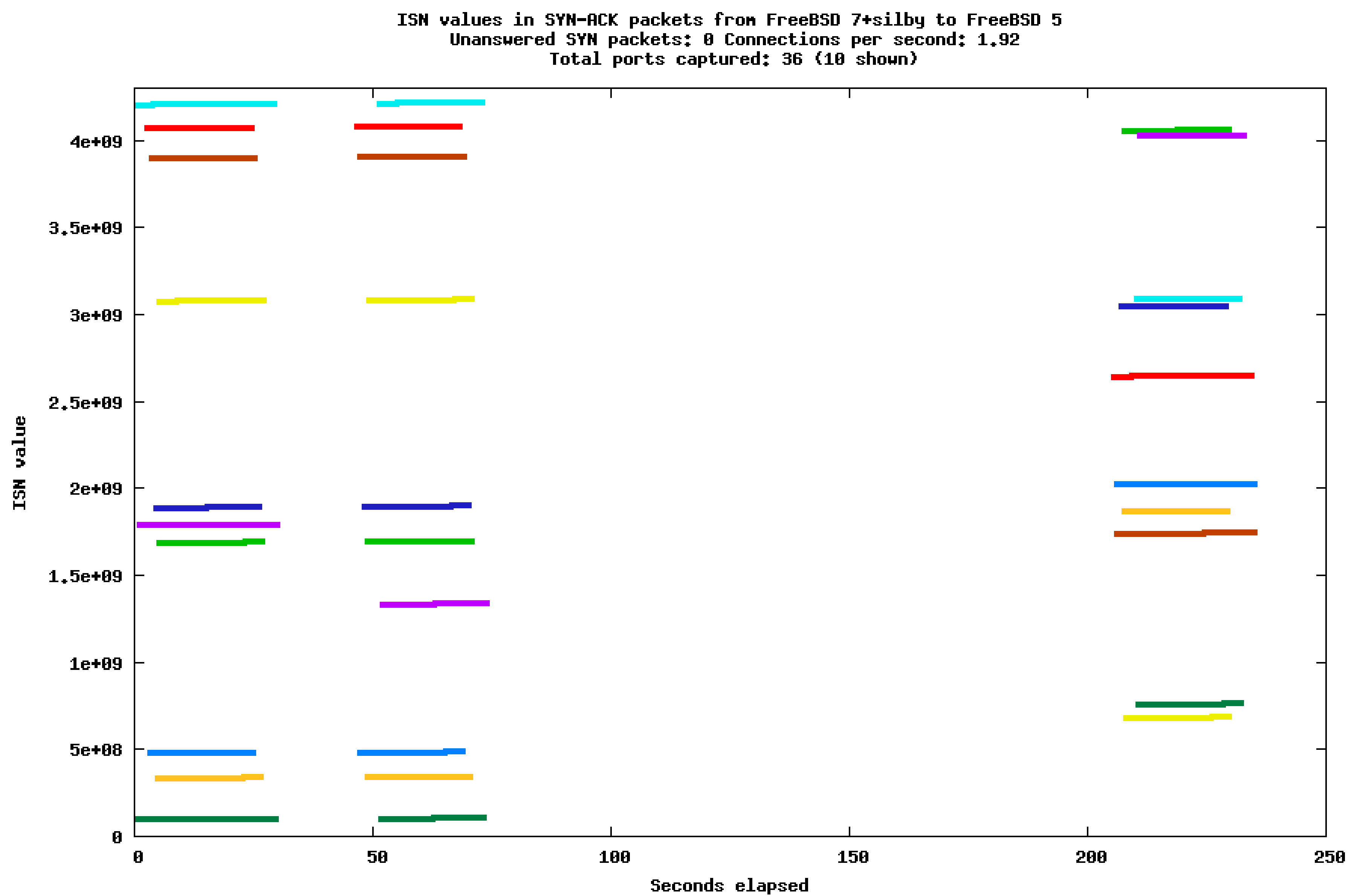
ISN Summary

- No two OSes are the same
  - Why?
- The FreeBSD way best meets the conflicting requirements of security and interoperability, but it is not perfect

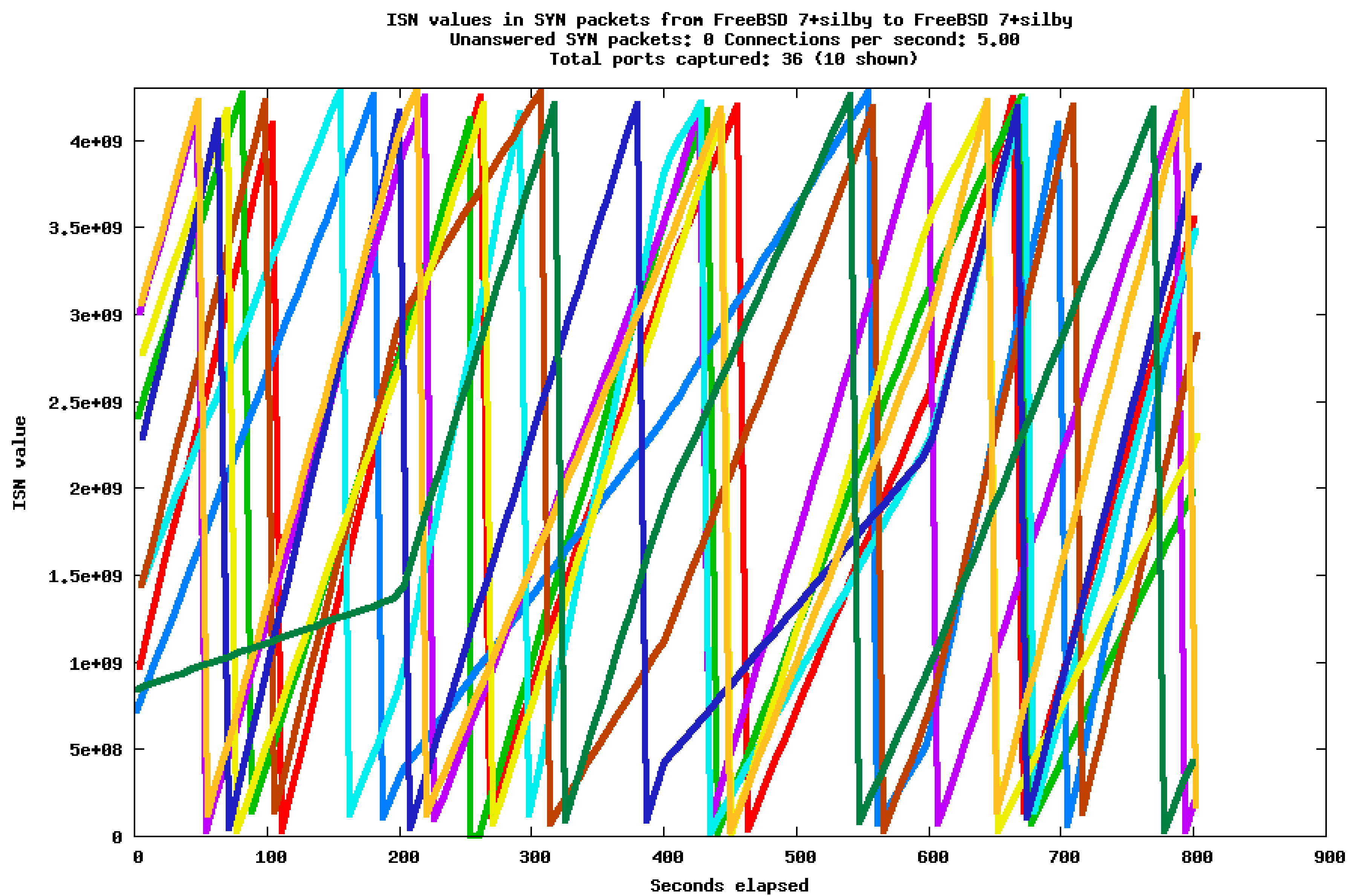
Improving the FreeBSD algorithm

- Flaws in the FreeBSD algorithm:
  - As the ISN values in SYN-ACK packets are randomized, there exists the possibility that the same sequence space will be used and a duplicate packet from the previous incarnation of the connection will cause problems
  - The RFC 1948 generated values in SYN packets exhibit the inherent weakness in RFC 1948

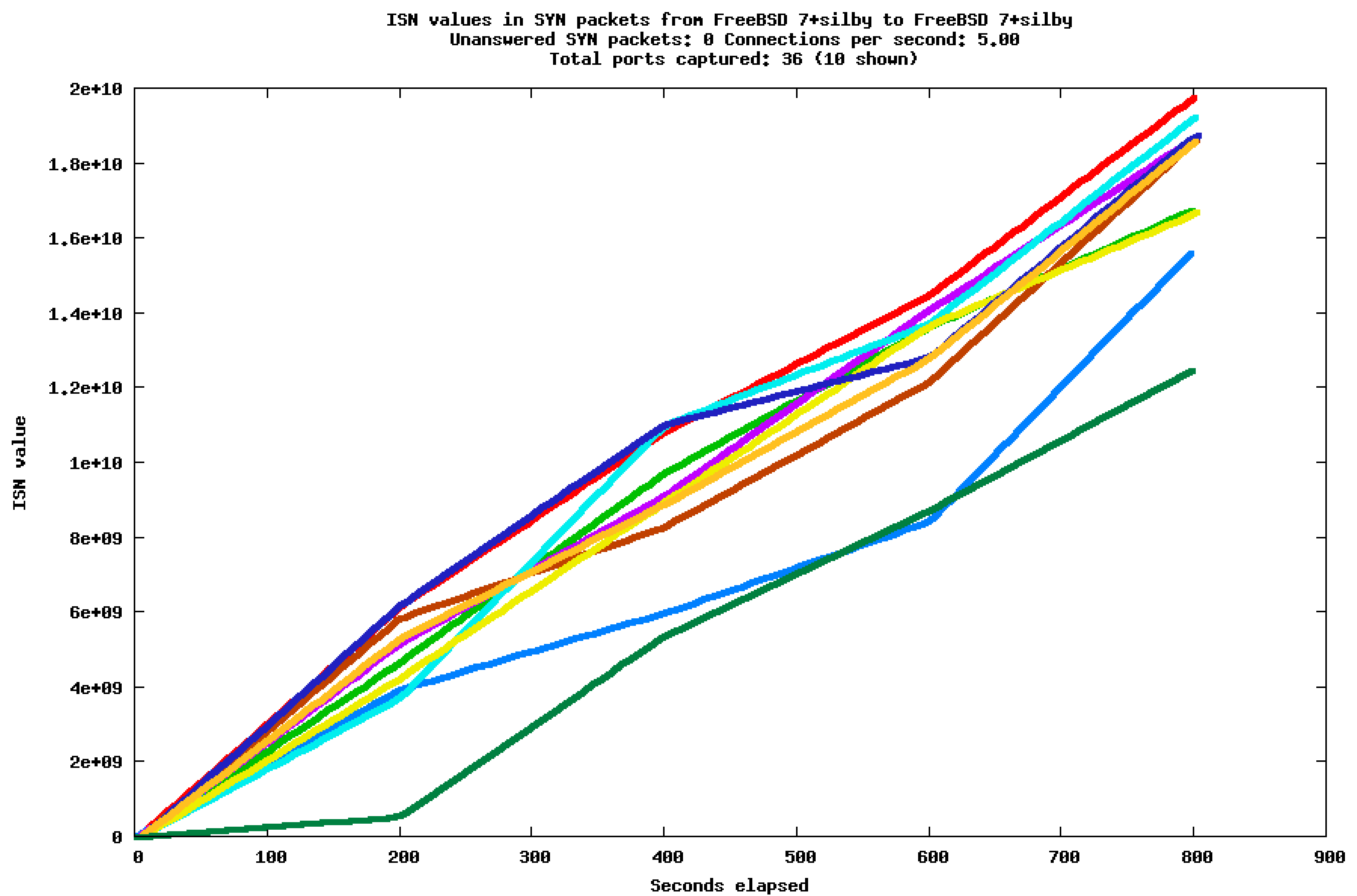
Improving FreeBSD SYN-ACK ISNs



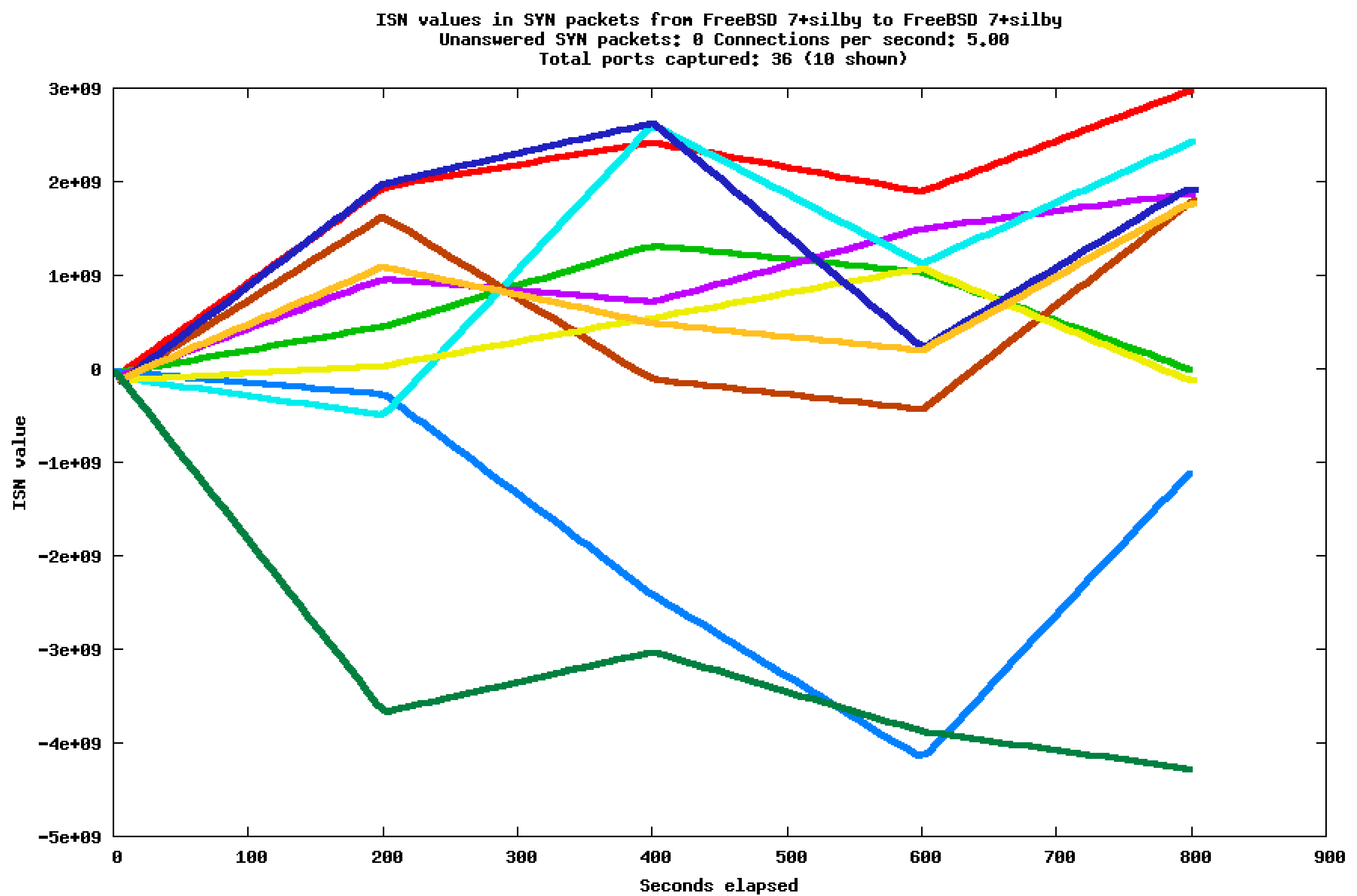
The dual-hash RFC 1948 variant



Another view of dual hashing



A View With Time Removed



TCP Timestamps

- The TCP Timestamp option was introduced in RFC 1323
- Timestamps serve two main purposes:
  - To allow for more accurate RTT calculations
  - For Protection Against Wrapped Sequence numbers (PAWS)
- All popular Operating Systems implement Timestamps, although Windows does not like to use them by default.

Timestamp Information Leakage

- Using a system-wide timestamp counter reveals a host's uptime
- Using a system-wide timestamp counter reveals which connections from a NAT machine originate from the same machine behind NAT.

Quick Fixes to Timestamps

- NetBSD: Start each connection's timestamp at zero
- OpenBSD: Start each connection's timestamp randomized
- The problem:
  - Timestamps are no longer useful for the purposes of PAWS
  - Linux makes the (reasonable) assumption that timestamps are monotonic over connection recycling in a few places

A Better Improvement For Timestamps

- Use the RFC 1948 algorithm, but use only the two IP addresses and the system-wide secret as input.
- Preserves PAWS usage
- Generally obscures uptime
- Does not solve the NAT issue entirely
- Allows for an important security improvement (next slide)

RFC 1948 Timestamp Security

- When timestamps are generated using RFC 1948, they will be predictable only on a per-IP basis.
- Hosts can check 32-bit timestamps as well as 32-bit sequence numbers
- Assume that a 16-bit sliding window of acceptable timestamps is used
- Spoofing packets is now 2^16 times as difficult
- Such a verification algorithm will still work if the other host does not use RFC 1948 timestamps, it will just not improve security.

Summary

- Security and Interoperability can coexist
- Significant testing is necessary to make this happen
- Interoperability is more important than security to some vendors

Questions?