
# Ourmon and Network Anomaly Detection

Jim Binkley
jrb@cs.pdx.edu
Portland State University
Computer Science

# Outline

- intro to ourmon, a network monitoring system
- network control and anomaly detection
  - a TCP attack
  - a UDP attack
- Gigabit Ethernet - flow measurement
  - what happens when you receive 1,488,000 64 byte packets a second?
- conclusions

# ourmon introduction

- ourmon is a network monitoring system
  - with some similarities/differences to
  - traditional SNMP RMON II
    - name is a take off on this  (ourmon is not rmon)
  - Linux ntop
- we deployed it in the PSU DMZ a number of years ago (2001)
  - first emphasis on RMON like stats
    - how many packets, how much TCP vs UDP, etc.
  - recent emphasis on detection of network anomalies

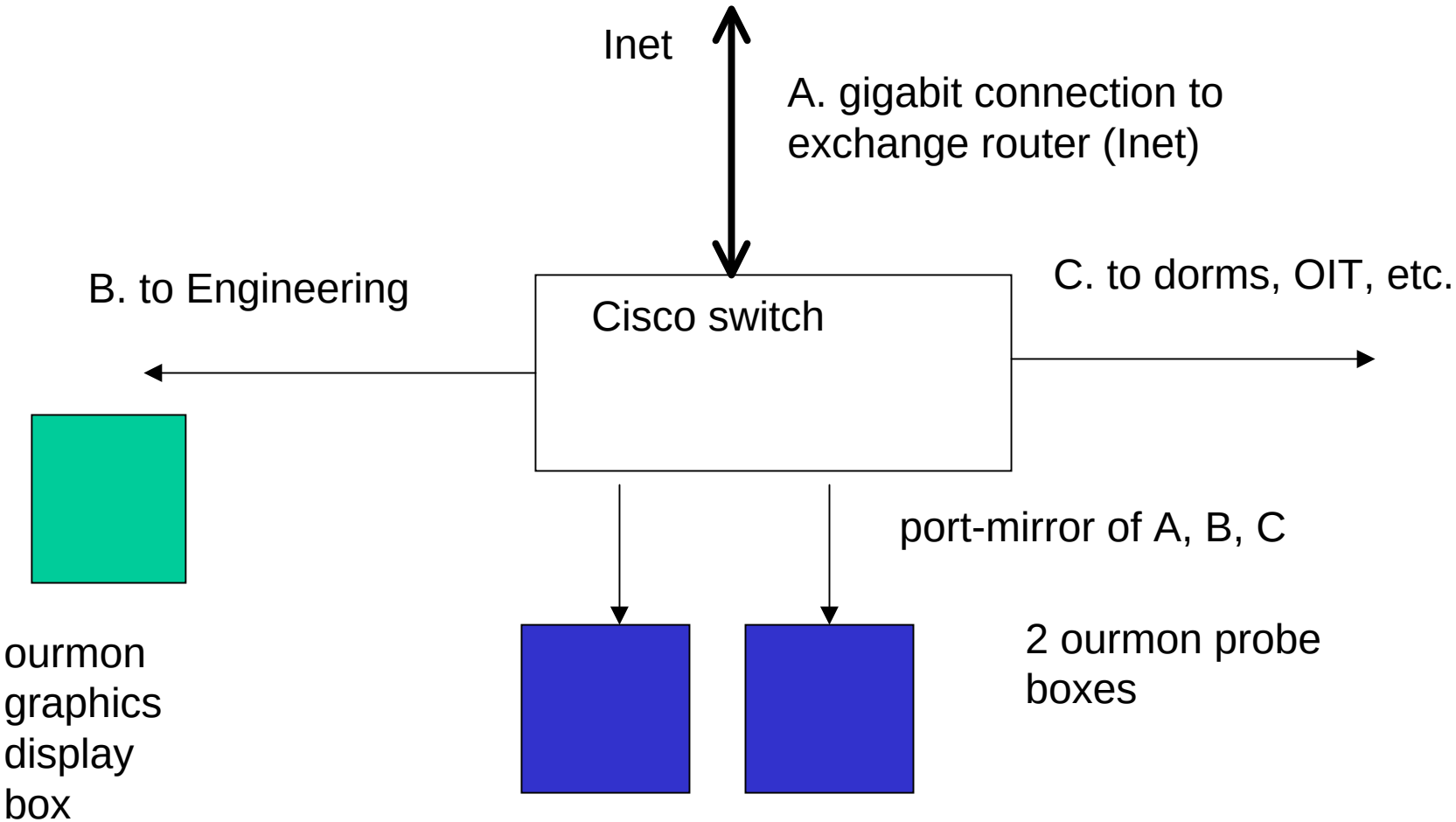
# PSU network

- Gigabit Ethernet backbone including GE connection to Inet1 and Inet2
  - I2 is from University of Washington (OC-3)
  - I1 is from State of Oregon university net (NERO)
- 350 Ethernet switches at PSU
  - 10000 live ports, 5-6k hosts
  - 4 logical networks: resnet, OIT, CECS, 802.11 (pubnet)
- 10 Cisco routers in DMZ
- ourmon shows 15-30k packets per second in DMZ

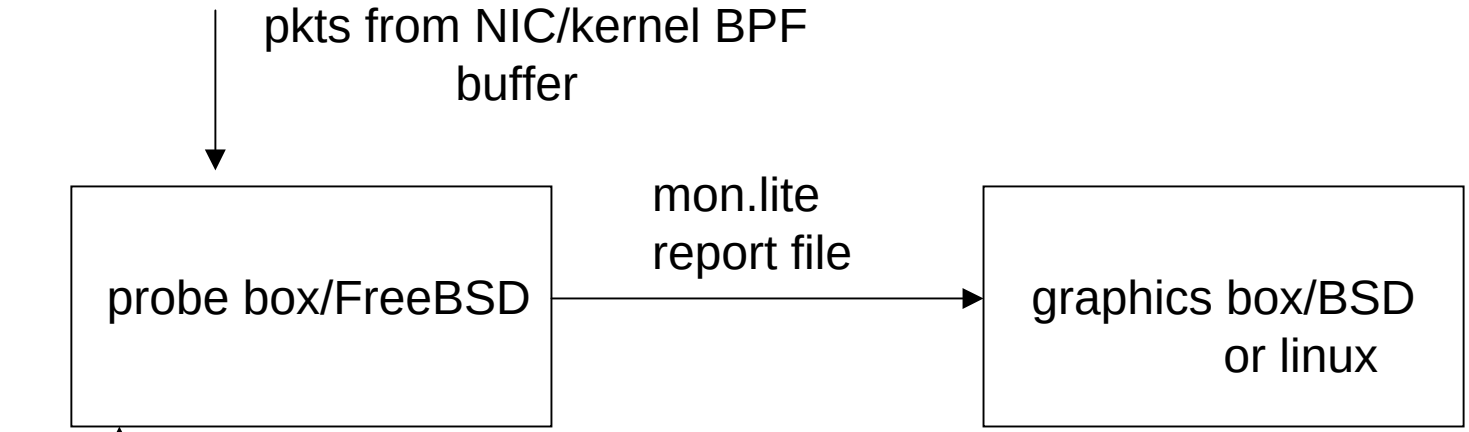
# ourmon architectural overview

- a simple 2-system distributed architecture
  - front-end probe (can easily divide load up for more)
  - back-end graphics/report processor
- front-end depends on Ethernet switch port-mirroring
  - like Snort
- does NOT use ASN.1/SNMP
- summarizes/condenses data for back-end
- cp summary file via out of band technique
  - micro_httpd/wget, or scp, or rsync, or whatever

# ourmon current deployment in PSU DMZ



# ourmon architectural breakdown




runtime:
1. N BPF expressions
2. + topn (hash table) of flows and other interesting topn like things +
3. some hardwired C filters

outputs:
1. bpf graphs using RRDTOOL (stripcharts)
2. hardwired graphs using RRDTOOL
3. topn histograms
   + syslog
   + hourly/daily reports

# the front-end probe

- written in C
- input file: ourmon.conf
  - 1-6 BPF expressions may be grouped in a named graph, and count either packets or bytes
  - some hardwired filters written in C
  - topn filters (generates lists, #1, #2, ... #N)
  - all filters specified by name, which is used in the backend to make distinct files
- output file: mon.lite
  - summarization of stats
  - ASCII, but very small (current 5k)

# the front-end probe

- typically use 7-8 megabyte kernel BPF buffer
- we only look at traditional 68 byte snap size
  - a la tcpdump
  - meaning HEADERS only, not data
- at this point due to hash tuning we rarely drop packets
  - barring massive syn attacks
- front-end basically is 2-stage
  - gather packets and count according to filter type
  - write report at 30-second alarm period

# ourmon.conf filter types

- **1. hardwired filters are specified as:**
- *fixed_ipproto  # tcp/udp/icmp/other pkts*
- *packet capture filter cannot be removed*
- **2. 1 user-mode bpf filter (configurable)**
- *bpf “ports” “ssh” “tcp port 22”*
- *bpf-next “p2p” “port 1214 or port 6881 or ...”*
- *bpf-next “web” “tcp port 80 or tcp port 443”*
- *bpf-next “ftp” “tcp port 20 or tcp port 21”*
- **3. topN filter is just**
- *topn_ip 9*

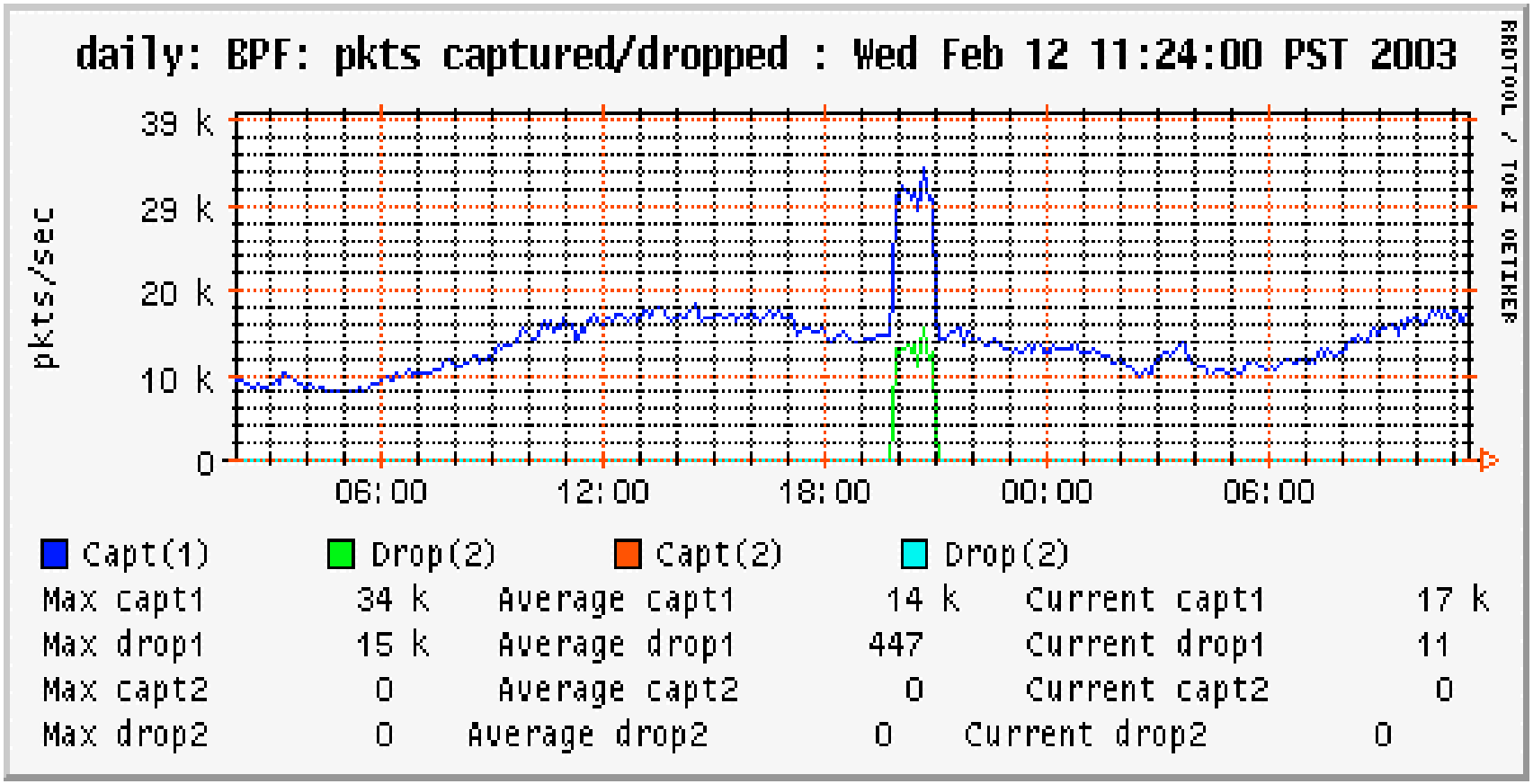
# mon.lite output file roughly like this:

- pkts: caught:670744 : drops:0:
- fixed_ipproto: tcp:363040008 : udp:18202658 : icmp:191109 : xtra:1230670:
- bpf:ports:0:5:ssh:6063805:p2p:75721940:web:102989812:ftp:7948:email:1175965:xtra:0
- topn_ip : 55216 : 131.252.117.82.3112->193.189.190.96.1540(tcp): 10338270 :  etc

# back-end does graphics

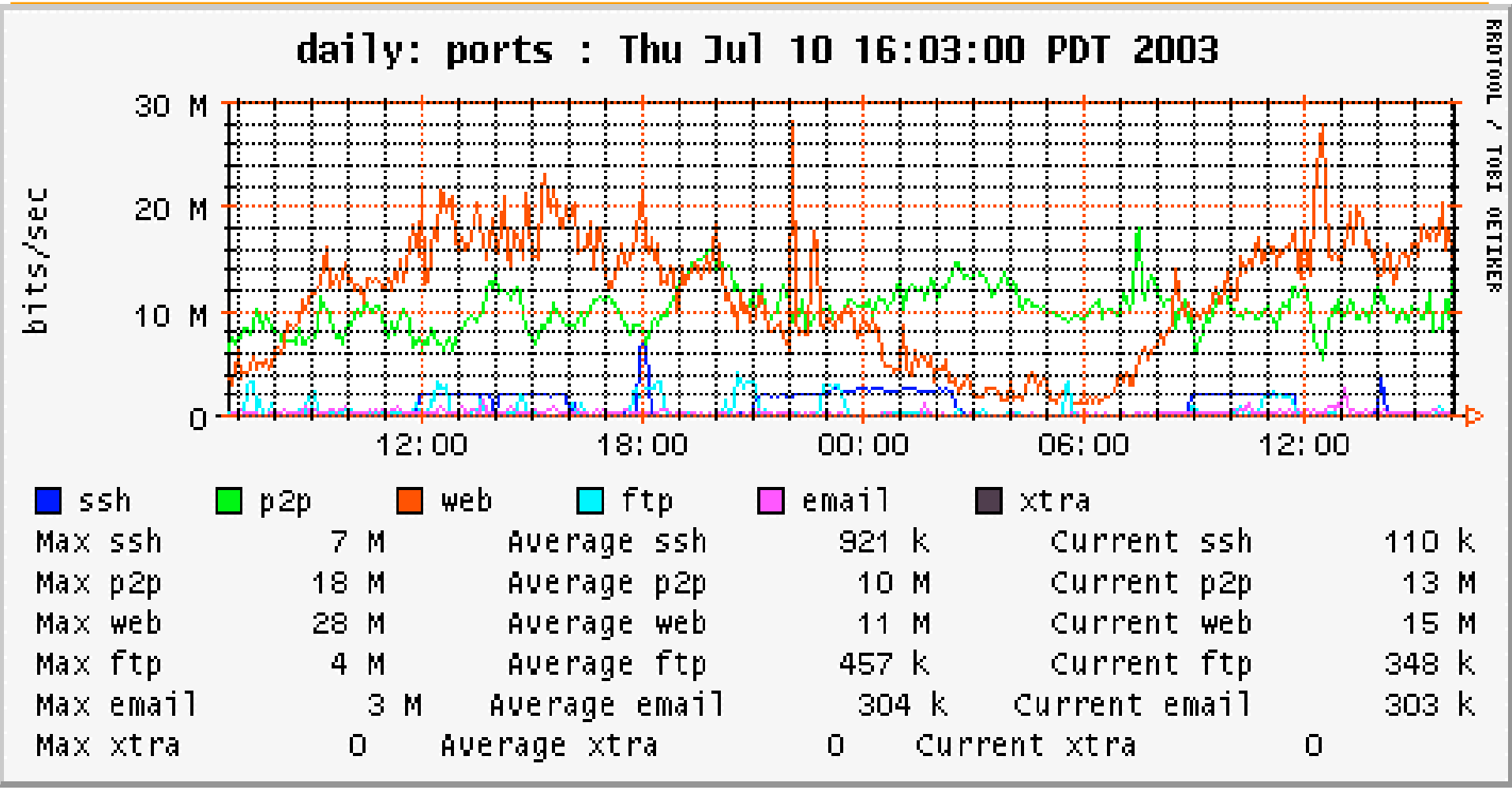
- written in perl
- uses Tobias Oetiker RRDTOOL for some graphs
  - as used in cricket/mrtg, other apps popular with network engineers
  - easy to baseline 1-year of data
  - logs (rrd database) has fixed size at creation
- top N uses histogram (our program) plus UNIX syslog
  - plus perl reports for topn data
  - we keep 1 week of data

# hardwired-filter #1: bpf counts/drops




this happens to be yet another SQL slammer attack.
front-end stressed as it lost packets due to the attack.

# 2. bpf filter output example




note: xtra means any remainder and is turned off in this graph.
note: 5 bpf filters mapped to one graph

# 3. topN example (histogram)

# ourmon has taught us a few hard facts about the PSU net

- P2P never sleeps (although it does go up in the evening)
  - Internet2 wanted apps.  It got **bittorrent.**
- PSU traffic is mostly TCP traffic
- web and P2P are the top apps
  - bittorrent/edonkey
- PSU's security officer spends a great deal of time chasing multimedia violations ...
  - Sorry students: Disney doesn't like it when you serve up Shrek
  - PSU is a net exporter of Inet bits

# current PSU dmz ourmon probe

- has about 60 BPF expressions grouped in 16 graphs
  - many are subnet specific (e.g., watch the dorms)
  - some are not (watch tcp control expressions)
- about 7 hardwired graphs
  - including a count of flow expressions IP/TCP/UDP/ICMP, and a count of topn hash inserts
- topn graphs include:
  - TCP syn'ners, IP flows (TCP/UDP/ICMP), top ports, ICMP error generators, UDP weighted errors
  - 1 ip src to many ip dst scans, 1 ip to many L4 ports

# ourmon and intrusion detection

- obviously it can be an anomaly detector
- McHugh/Gates paraphrase: Locality is a paradigm for thinking about normal behavior and “Outsider” threat
  - or **insider threat** if you are at a university with dorms
- thesis:  anomaly detection focused on
  - 1. network control packets; e.g., TCP syns/fins/rsts
  - 2. errors such as ICMP packets
  - 3. meta-data such as flow counts, # of hash inserts
- seems to be useful for scanner/worm finding

# inspired by noticing this ...

mon.lite file (reconstructed), Oct 1: 2003

topn_ip: 163000:
topn_tcp: 50000
topn_udp: 13000
topn_icmp: **100000** ← oops ...

normal icmp flow count: 1000/30 seconds

**We should have been graphing the meta-data (the flow counts).**
**Widespread Nachi/Welchia worm infection in PSU dorms**

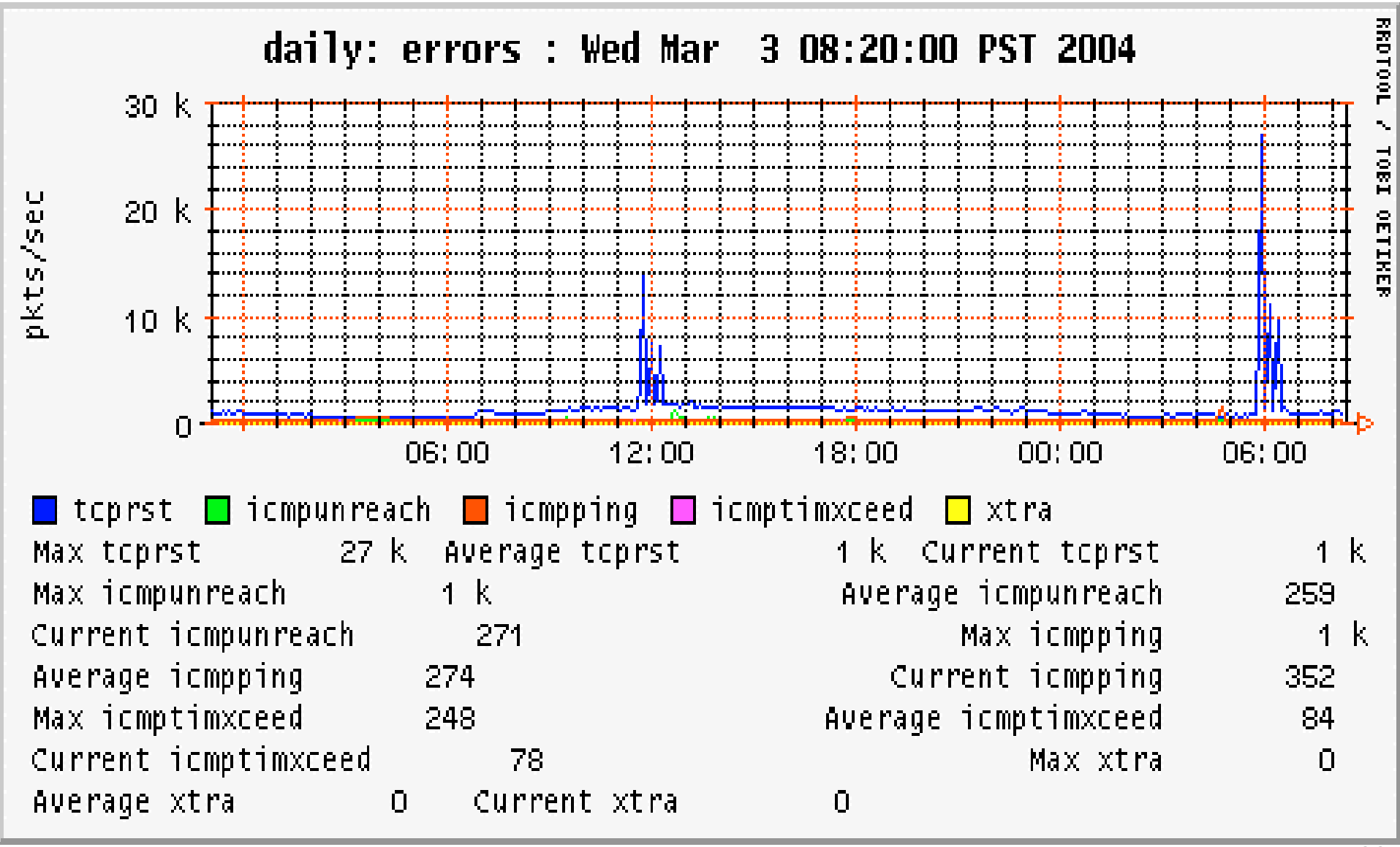
# actions taken as a result:

- we use the BPF/RRDTOOL to graph:
  - 1. network “errors” TCP resets and ICMP errors
  - 2. we graph TCP syns/resets/fins
  - 3. we graph ICMP unreachables (admin prohibit, host unreachable etc).
- we have RRDTOOL graphs for flow meta-data:
  - topN flow counts
  - topN hash inserts
- we have a new topn syns and others
  - sorts by SYNS, shows FINS/RESETS
- RRDTOOL graph for syn scanner ip count

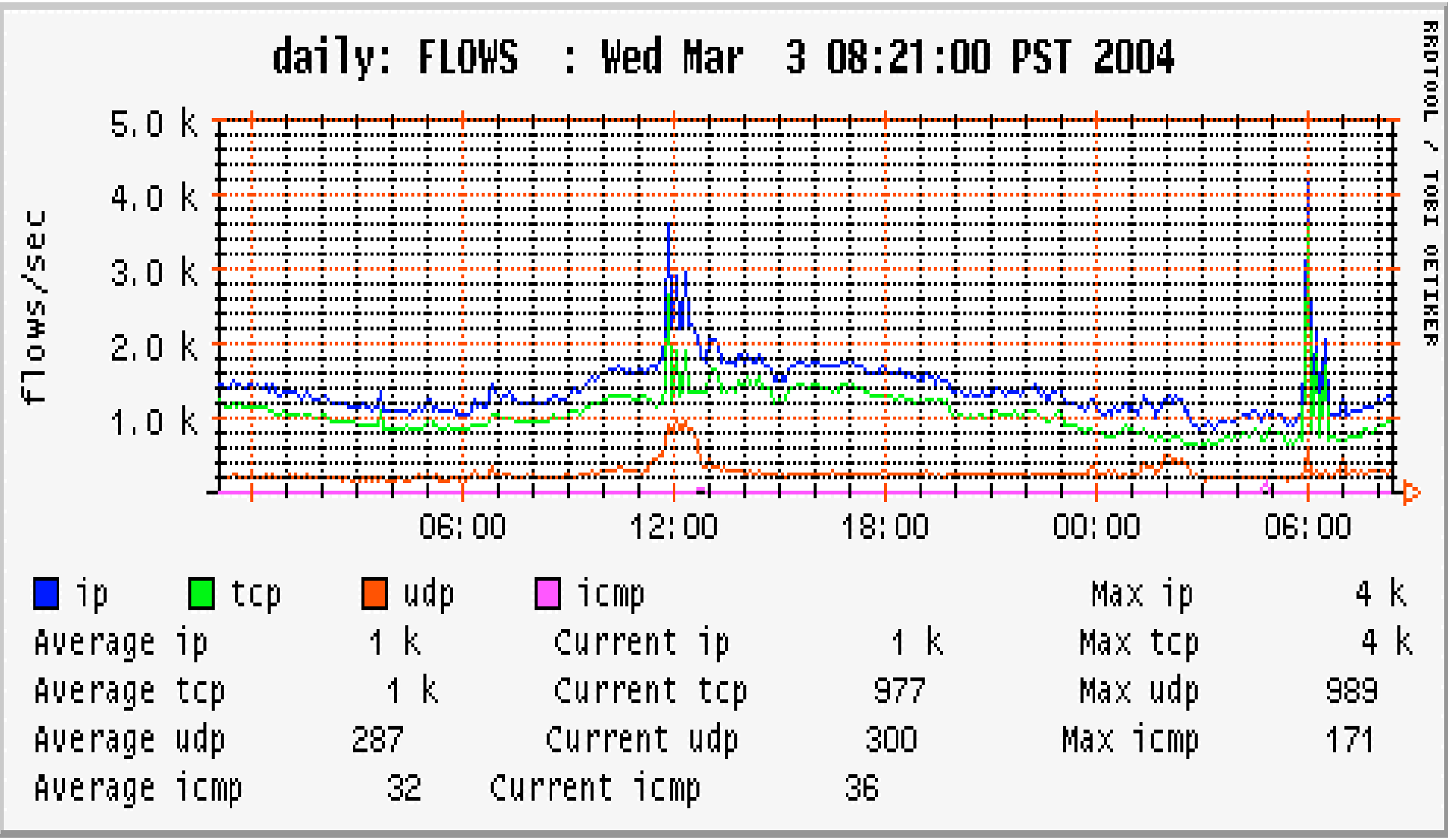
# daily topn reports are useful

- top N syn reports show us the cumulative
  - synners over time
  - if many syns, few fins, few resets
    - almost certainly a scanner/worm (or trinity?)
  - many syns, same amount of fins, may be a P2P app
- ICMP error stats
  - show up both top TCP and UDP scanning hosts
  - especially in cumulative report logs
- both of the above reports show MANY infected systems (and a few that are not)

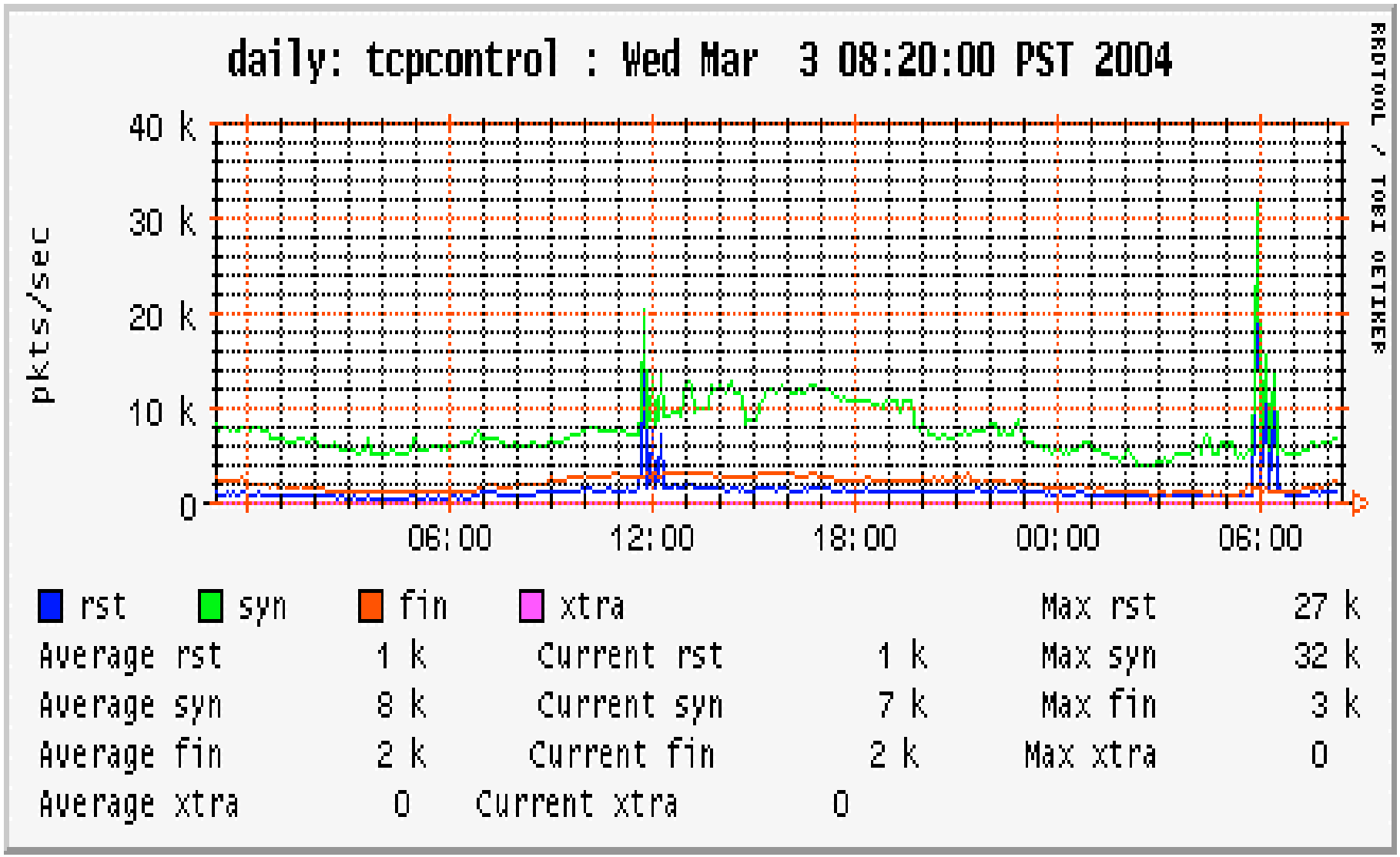
# 6:00 am TCP attack - BPF net errors



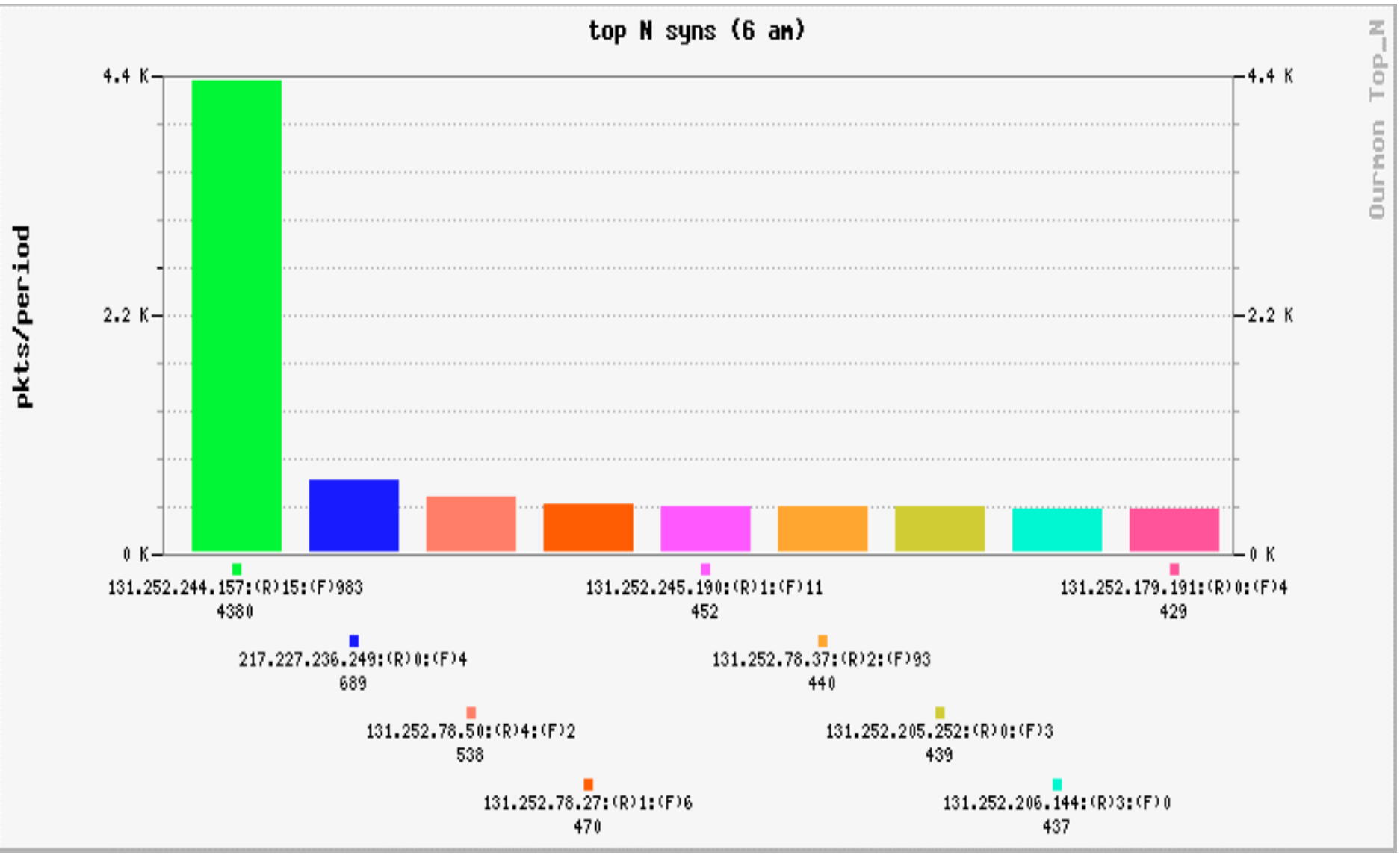
# topn RRD flow count graph



# bpf TCP control



# 6 am TCP top syn (this filter is useful...)



# topn syn syslog sort

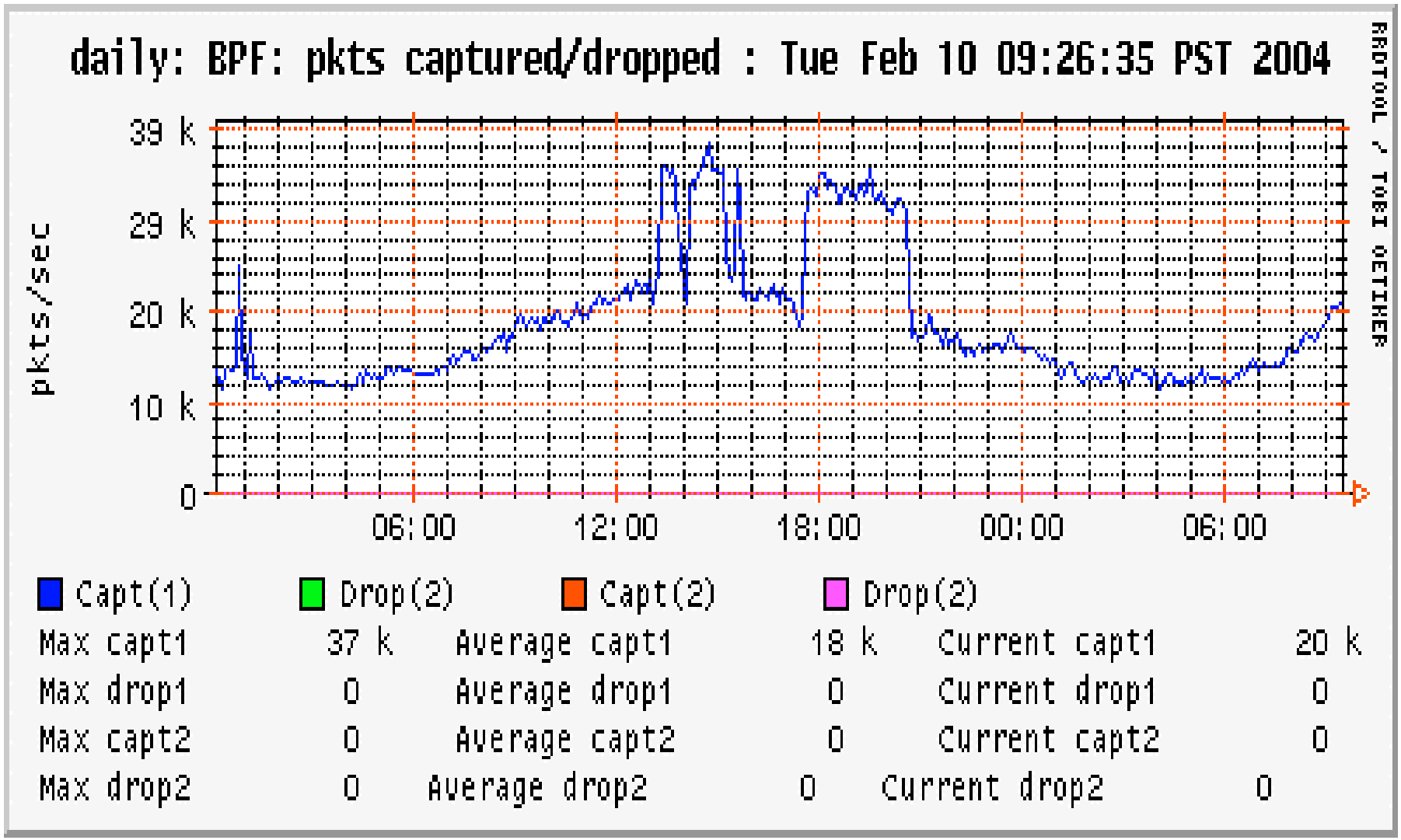
start log time : instances: DNS/ip : syns/fins/resets total counts end log time

---

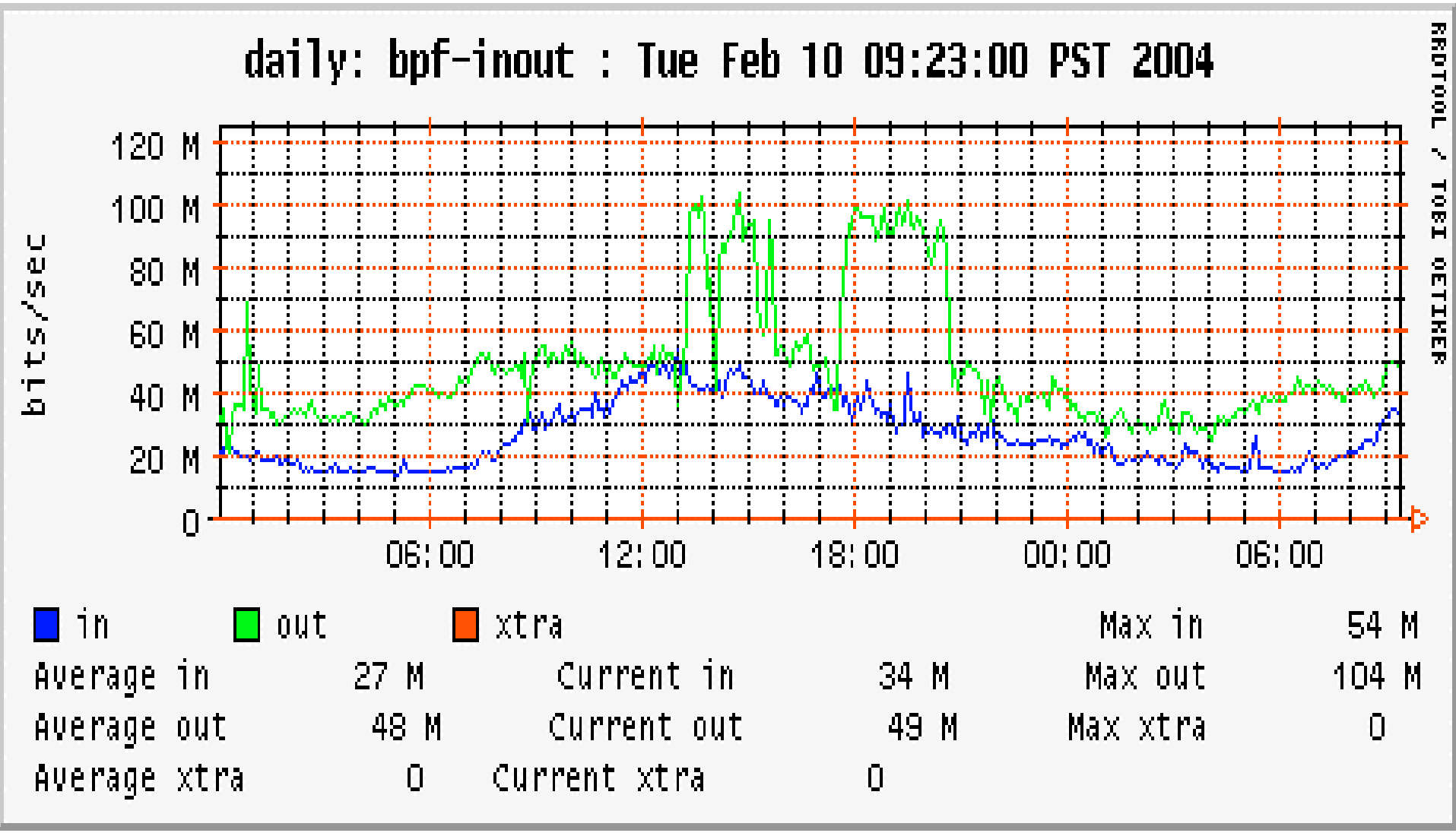
---

Wed Mar 3 00:01:04 2004: 777: host-78-50.dhcp.pdx.edu:401550:2131:2983
Wed Mar 3 07:32:36 2004
Wed Mar 3 00:01:04 2004: 890: host-206-144.resnet.pdx.edu:378865:1356:4755
Wed Mar 3 08:01:03 2004
Wed Mar 3 00:01:04 2004: 876: host-245-190.resnet.pdx.edu:376983:1919:8041
Wed Mar 3 08:01:03 2004
**Wed Mar 3 00:01:04 2004: 674: host-244-157.resnet.pdx.edu:348895: :8468:29627**
Wed Mar 3 08:01:03 2004

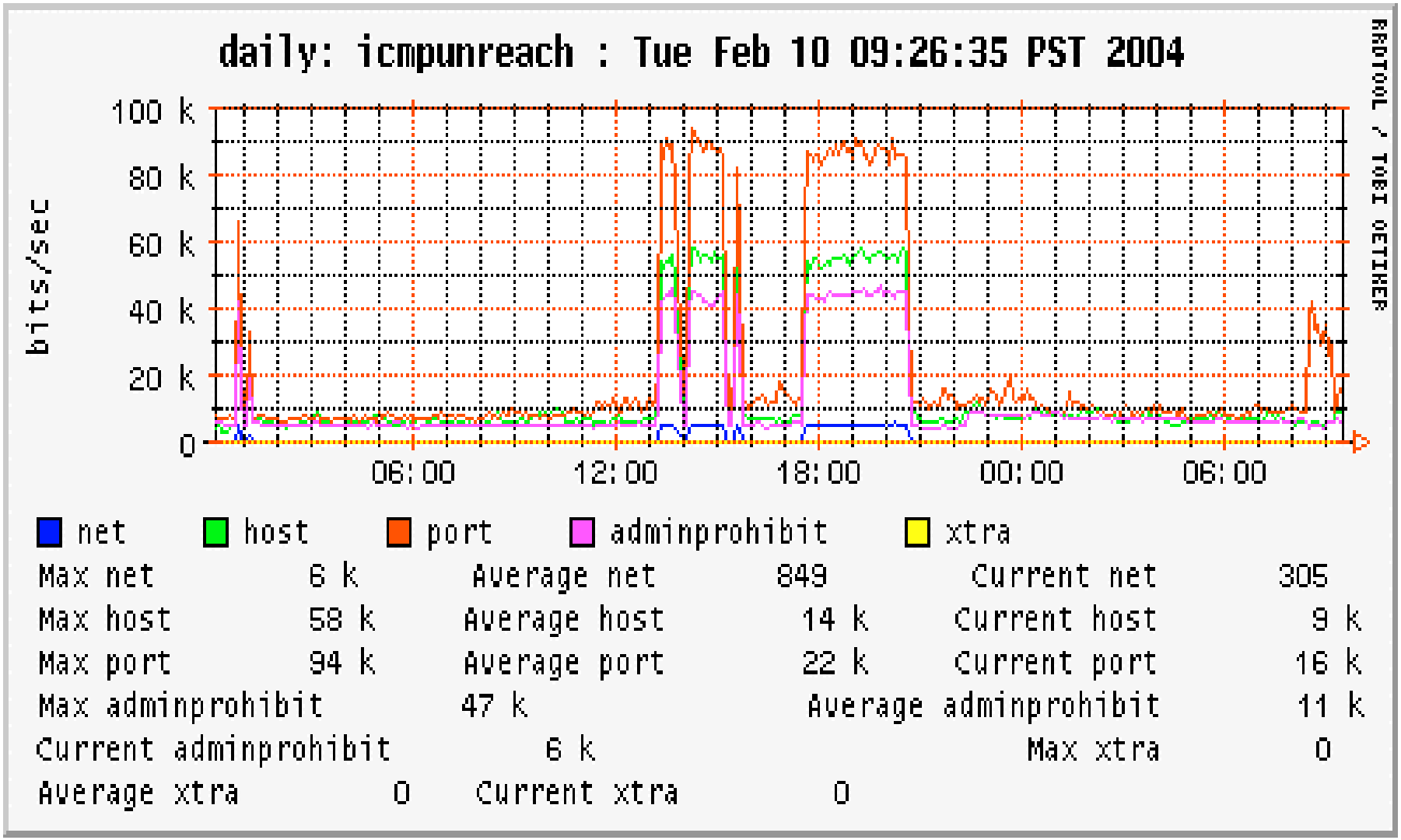
# 1st graph you see in the morning:



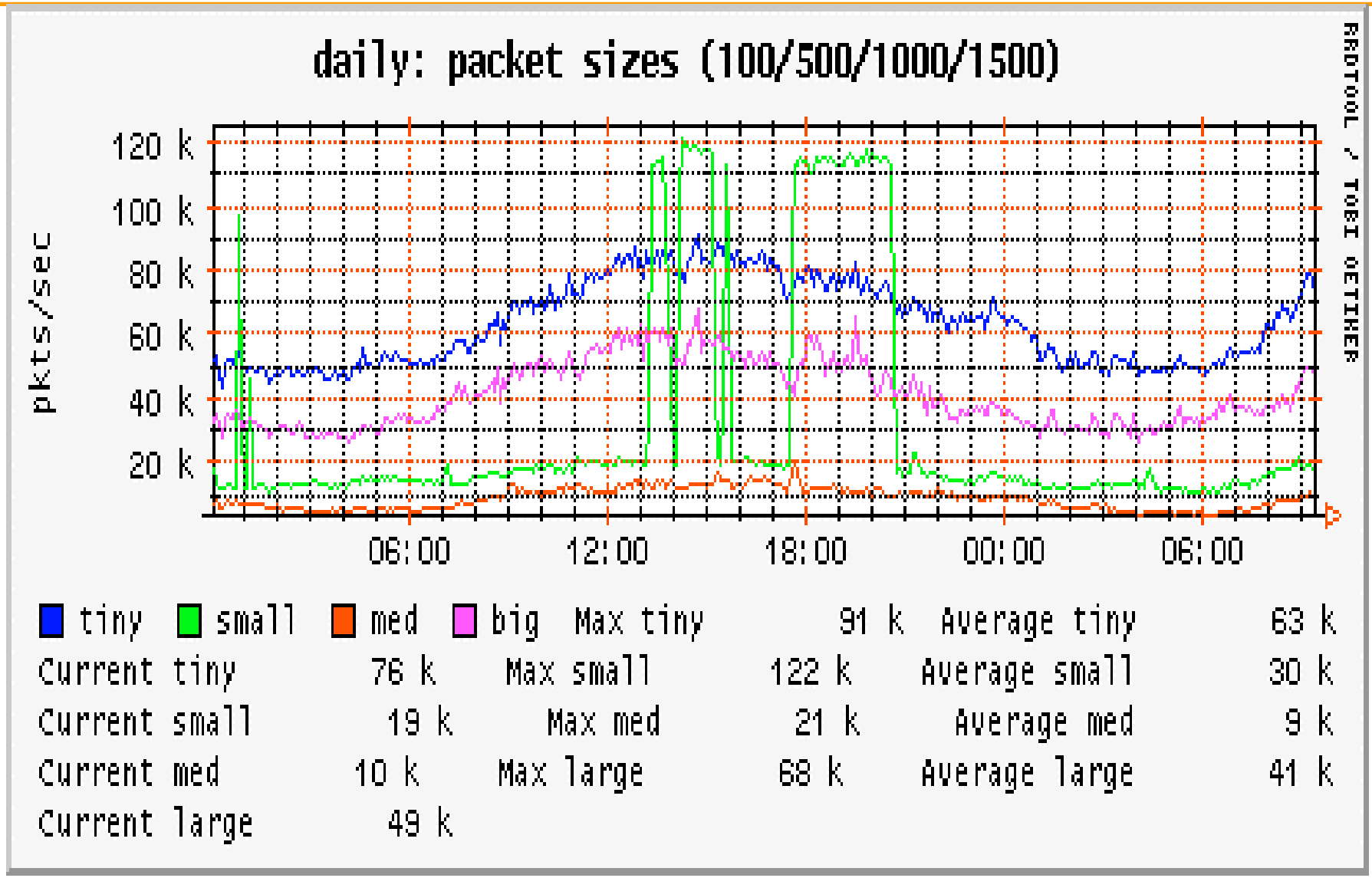
# BPF: in or out?


daily: bpf-inout : Tue Feb 10 09:23:00 PST 2004
RRDTOOL / TOBI OETIKER
bits/sec
120 M
100 M
80 M
60 M
40 M
20 M
0
06:00
12:00
18:00
00:00
06:00
in
out
xtra
Max in 54 M
Average in 27 M
Current in 34 M
Max out 104 M
Average out 48 M
Current out 49 M
Max xtra 0
Average xtra 0
Current xtra 0

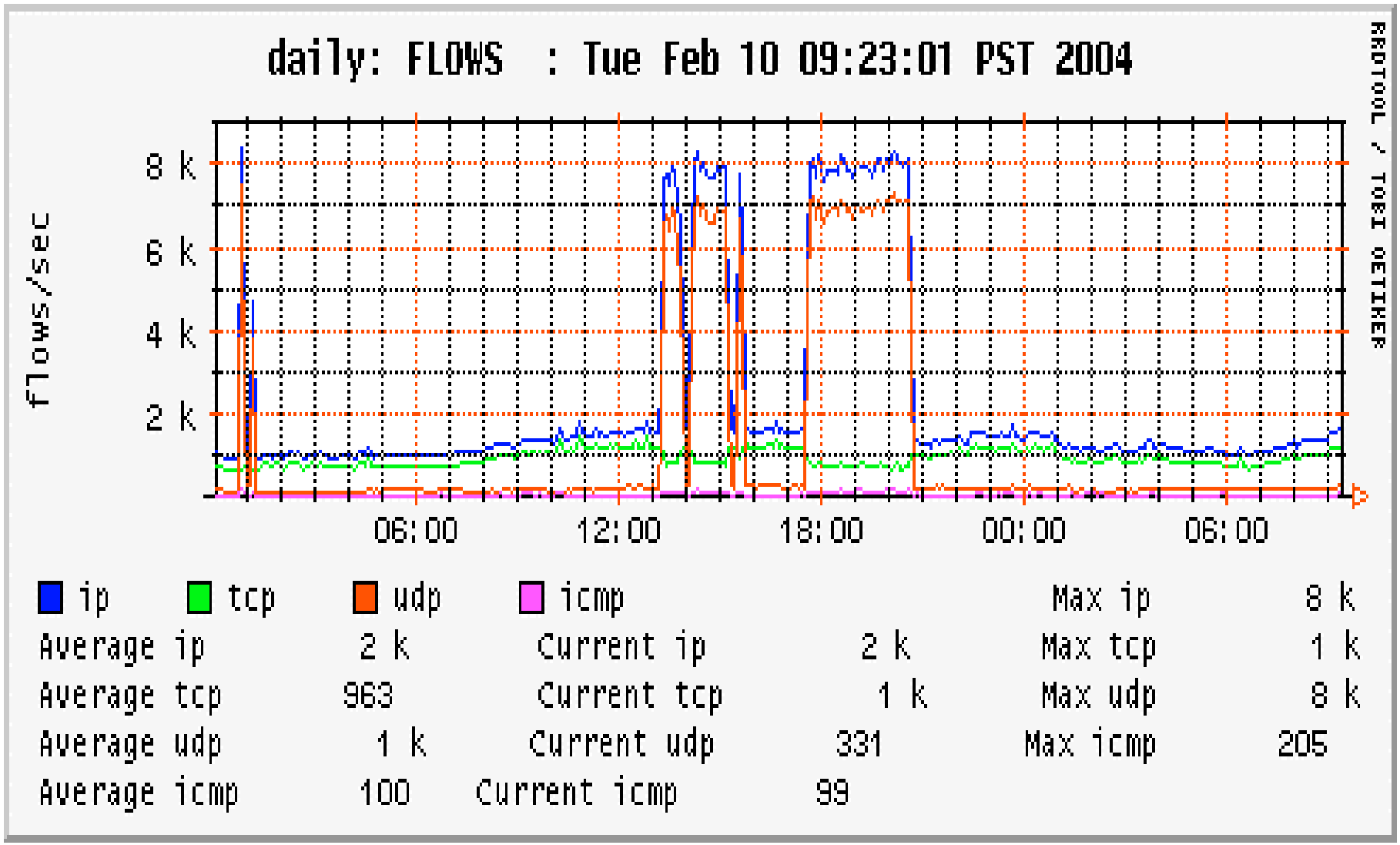
# BPF ICMP unreachables



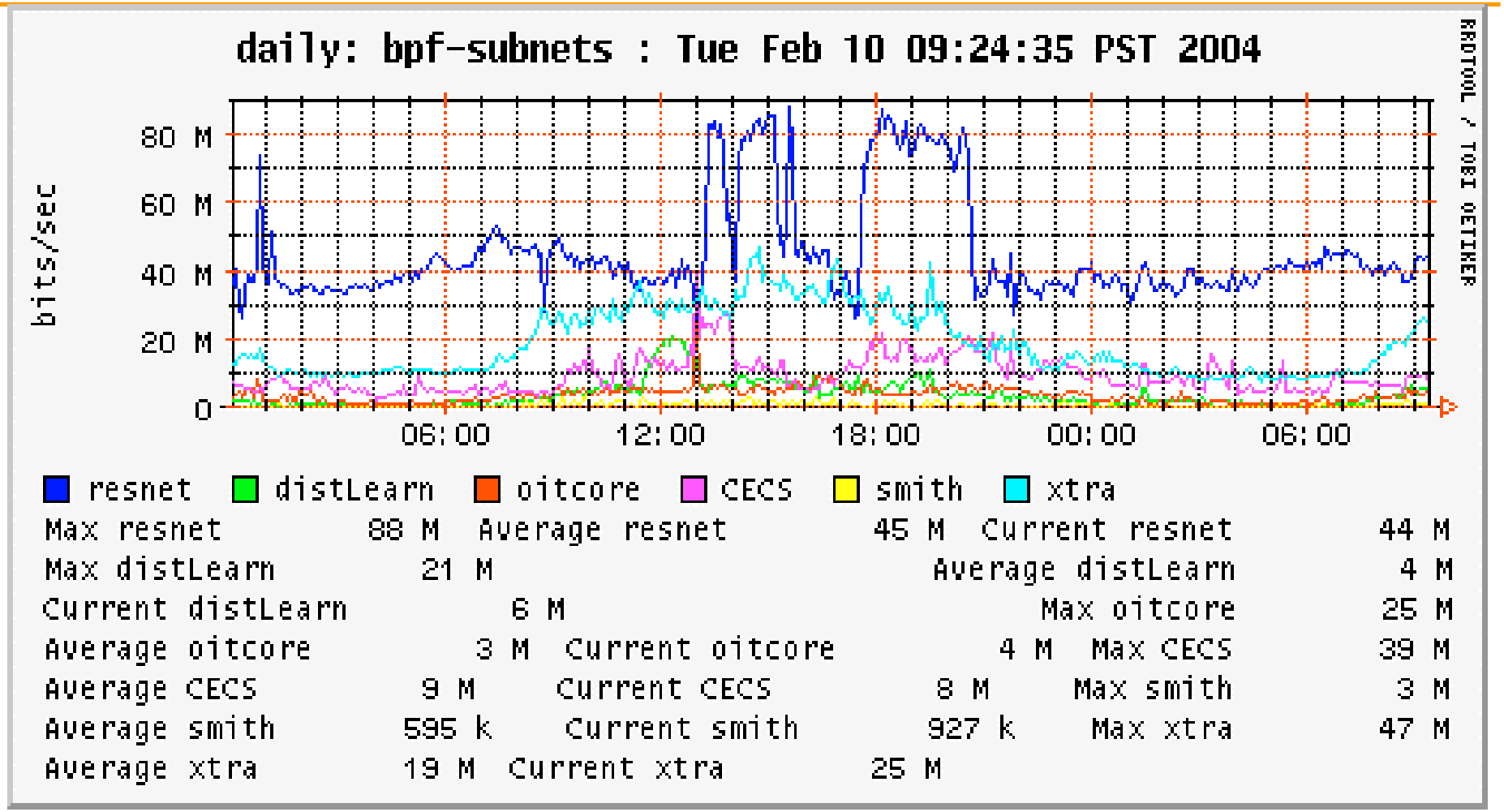
# hmm... size is 100.500 bytes



# flow picture: UDP and ICMP mixed

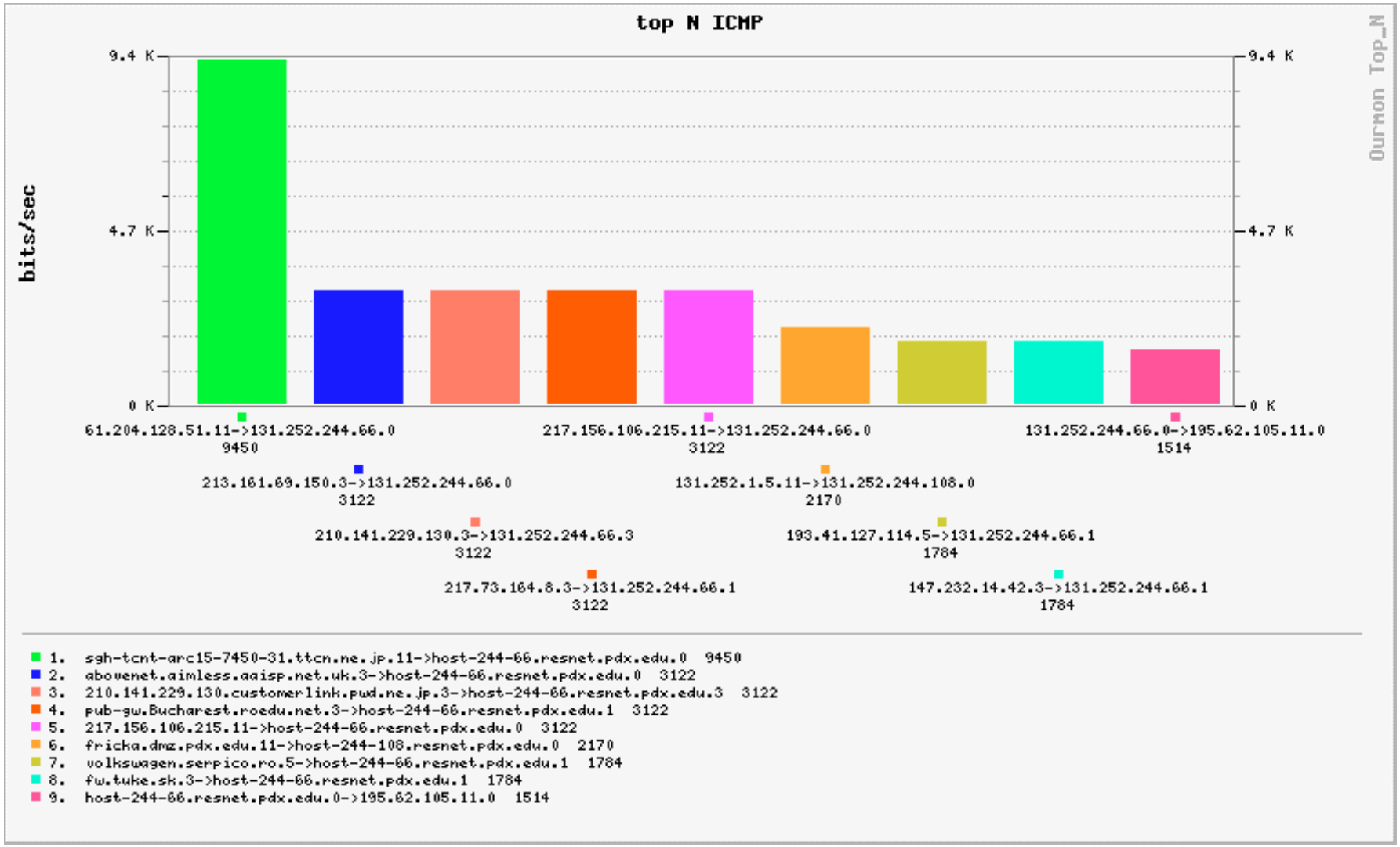


# bpf subnet graph:




OK, it came from the dorms (this is rare ..., it takes a strong signal)

# top ICMP shows the culprit's IP




7 out of 9 slots taken by ICMP errors back to 1 host

# and the answer is

- tcpdump on probe shows us:
- TCP syn attacker
  - syn scan for port 445 (DCOM)
- UDP attacker variant of slammer
  - blizzard of packets sent to port 1434
  - ICMP error logs showed host IP clearly
  - as did cymru report ...

# summary for this section

- TCP syns/fins/resets useful
  - many syns, few fins, some resets sure thing
- ICMP errors useful
  - especially for udp-based attack
  - tcp-based attacks also will generate them
  - including redirects, ttl exceeded, admin prohibited
- UDP weight notion?  send-recv * error
- P2P apps have high numbers of
  - Syns/Fins/ICMP
  - need to better understand their architecture

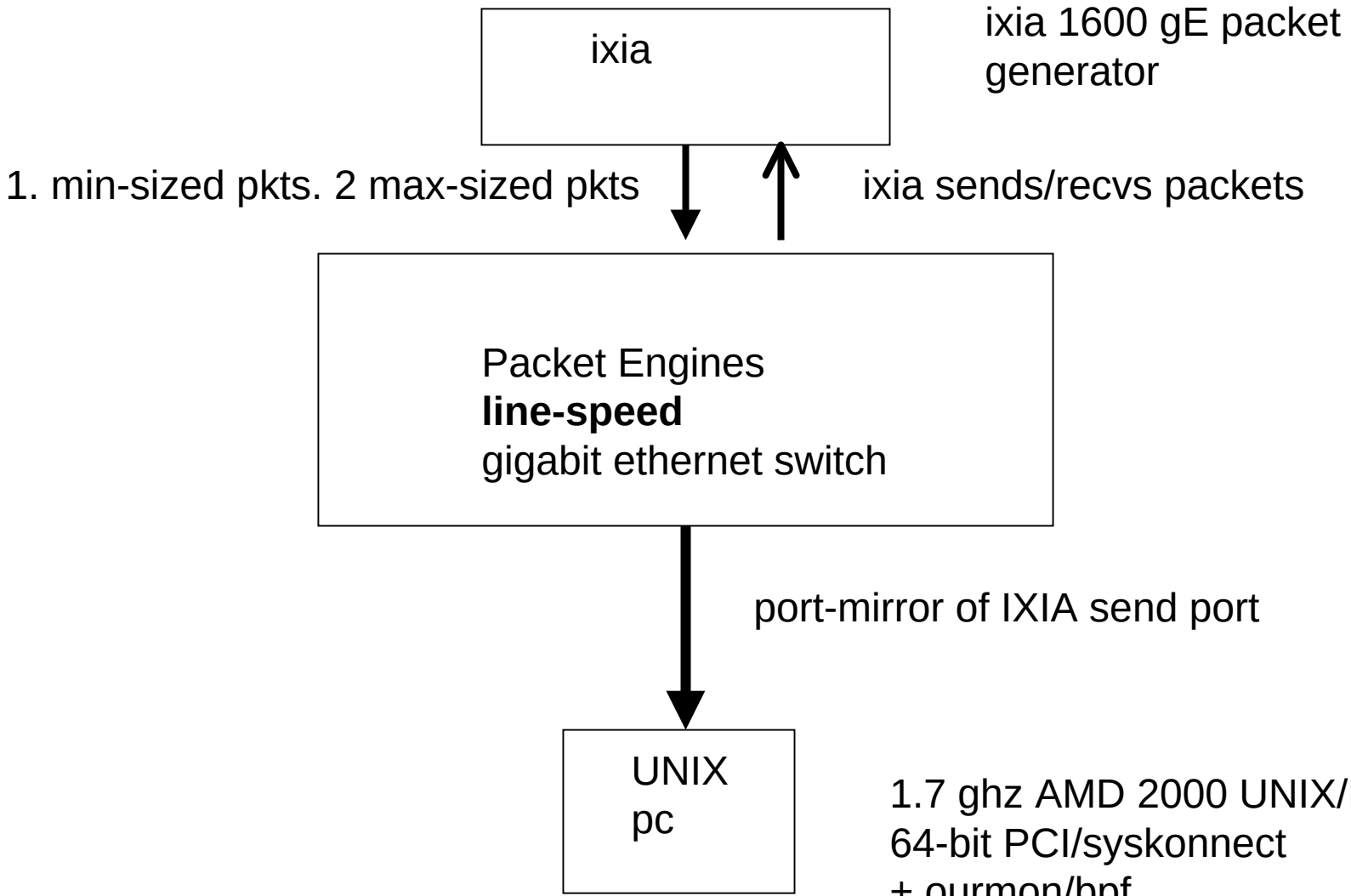
# Gigabit Ethernet speed testing

- test questions: what happens when we hit ourmon and its various kinds of filters
- 1. with max MTU packets
  - can we do a reasonable amount of work?
  - can we capture all the packets?
- 2. with min-sized packets (64 bytes)
  - same questions
- 3. is any filter kind better/worse than any other
  - topn in particular (answer is it is worse)
  - and by the way roll the IP addresses (insert-only)

# Gigabit Ethernet - Baseline

- acc. to TR-645-2, Princeton University, Karlin, Peterson, “Maximum Packet Rates for Full-Duplex Ethernet”:
- 3 numbers of interest for gE
  - min-packet theoretical rate: 1488 Kpps (64 bytes)
  - max-packet theoretical rate: 81.3 Kpps (1518 bytes)
  - min-packet end-end time: 672 ns
- note: the min-pkt inter-frame gap for gigabit is 96 ns (not a lot of time between packets ...)
- an IXIA 1600 packet generator can basically send min/max at those rates

# test setup for ourmon/bpf measurement



# test notes

- keep in mind that ourmon snap length is 68 bytes (includes L4 headers, not data)
  - The kernel BPF is not capturing all of a max-sized packet
- An IDS like snort must do this
  - it must run an arbitrary set of signatures over an individual packet
  - reconstruct flows
  - undo fragmentation

# maximum packets results

- with work including 32 bpf expressions, top N, and hardwired filters:
- we can always capture all the packets
- but we need a N megabyte BPF buffer in the kernel
  - add bpfs, add kernel buffer size
- this was about the level of real work we were doing at the time in the real world

# minimum packet results

- using only the basic count/drop filter
  - NO OTHER WORK!
- using any-size of kernel buffer (didn't matter)
- **we start dropping packets at around 80 mbit speed** (10% of the line rate with overhead)
- **this is only with the drop/count filter!**
- **if you want to do real work, 30-50 mbits more like it**
- **can't deal with healthy system that has 100mbit NIC card ...**

# why is min performance so poor?

- Two points-of-view that are complimentary.
- 1. there is not enough time to do any real work (you have 500 ns or so)
- 2. the bottom-half of the os is at HW priority, interrupts prevent the top-half from running (enough) to avoid drops.
- note that growing the kernel buffer doesn't help
- research question: **what is to be done?**
- btw: this is why switch/router vendors usually publish performance stats on min-sized pkts.

# also top N has a problem

- random inserts means bucket lookup always fails
  - followed by a malloc
- random IP src and/or random IP dst
  - how to deal with this?
  - one obvious answer: make sure hash algorithm is optimized as much as possible
- improved lookup certainly does help
  - insert is logically:  lookup to find correct bucket + insert (node allocation/setup/chaining)
  - our hash bucket size was way too small ...

# this explains my long standing question of

- why does ourmon sometimes drop packets?
  - in the drop/count graph
- but you couldn’t find any reason for it
- reason: looking at BIG things like flows,
- not small things like TCP syn attacks
  - which do not add up to anything in the way of a mByte flow
  - and may be distributed
- small packets are evil...

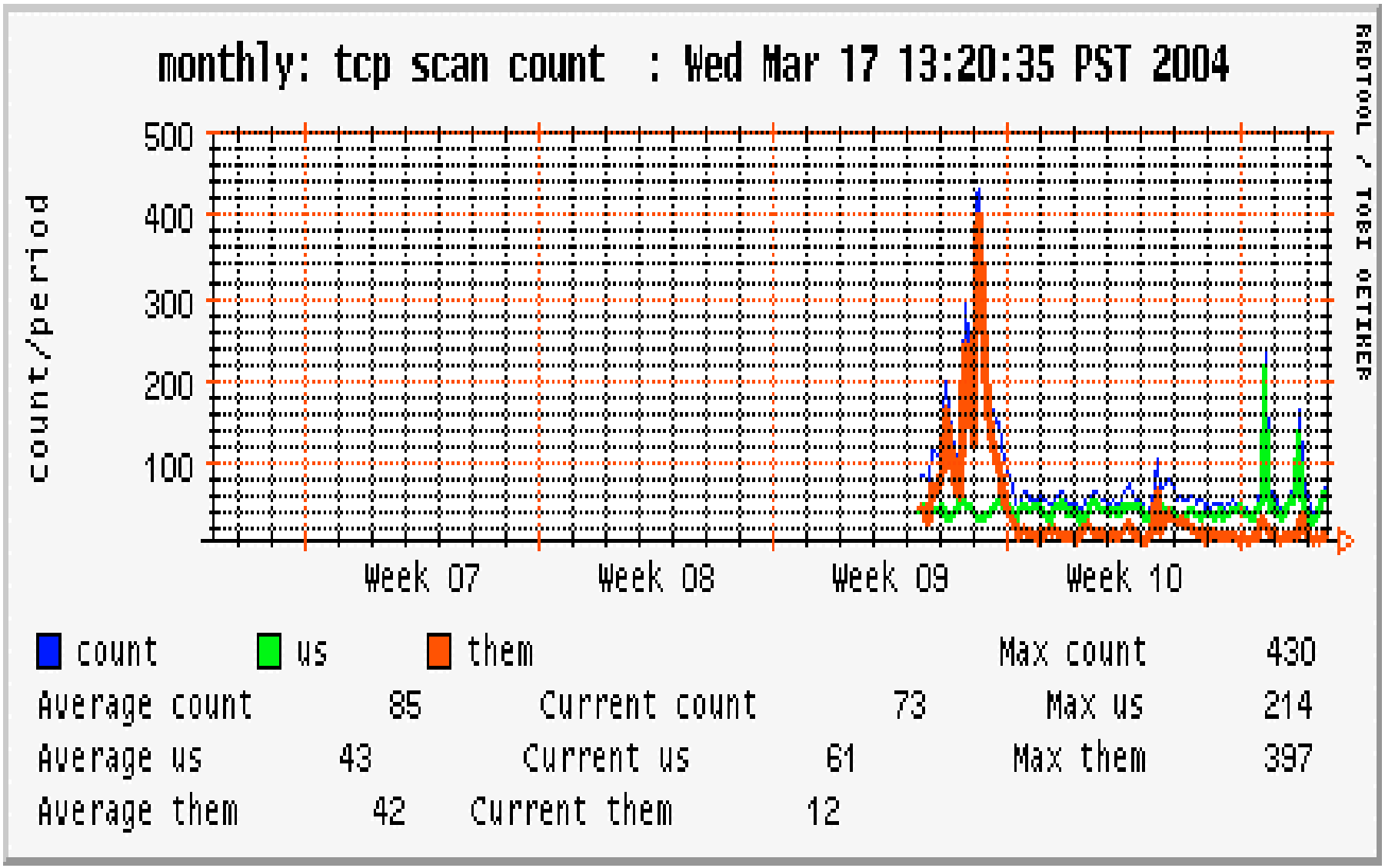
# ourmon gigabit test conclusions

- min packets are a problem
  - no filters and still overflows
  - topn needed optimization (now bpf needs optimization)
  - does topn problem apply to route caching in routers?
- a different parallel architecture is needed.for min pkts
- consequences for IDS snort system are terrible
  - easy to construct a DOS attack that can sneak packets by snort ?
  - 80 mbytes of 64-byte packets will likely clog it.
  - to say nothing of a concerted zombie attack
  - less could always do the trick depending on the exact circumstances and the amount of work done in the  monitor

# control/anomaly conclusions

- control theory approach to net anomaly detection is very useful
  - TCP syn/fin/reset, icmp errors to a lesser extent
  - UDP icmp errors
- carefully chosen metadata is useful too
  - topn inserts shows distributed attacks
- re control theory ... we need a good baseline
  - shouldn't PSU syns == PSU fins?
- baselines take time
  - how do you get a baseline if you are always under attack?

# meta-graph: “worm” count



# future work:

- re min pkts - create a parallel ourmon architecture possibly using Intel IXP 24XX
- BPF optimization
- auto-capture packets with packet-capture probe
  - front-end driven and/or back-end driven
- signal analysis
- make a release ...
  - BSD port too
- http://ourmon.cat.pdx.edu/ourmon
  - next release is ourmon 2.3