One-way Traffic Monitoring with `iatmon`

DUST Workshop, CAIDA, May 2012

Nevil Brownlee

Challenges in telescope data analysis

- UCSD telescope records hourly (full) trace files, 6 to 10.5 GB/h, gz-compressed
  - lots of data to handle in near-real time
  - too much data to store long-term
  - small changes, e.g. rise of a new worm, are now hard to see
- There are no hosts in the telescope's address space
  - we only see incoming packets, no replies
    - very little one-way traffic uses well-known ports
    - TCP packets are only opening SYN packets – no payload
    - UDP packets have (at least some) data, can sometimes recognise 'signatures' in it
- We needed a monitor that could
  - classify the packets from each source address into distinct subsets, so as to make changes more obvious
  - summarise each hour's data
- Long-term goal is to develop tools to mine the hourly summaries for short-term events and long-term changes

`iatmon` (IAT monitor) implementation

- `iatmon` scans traces, building its *sources* table (IPv4 and IPv6 addresses)
- records information about all the packets from each source
  - first and last packet times, inter-arrival time distribution
  - protocols and ports used, etc
- Managing memory
  - `iatmon` does its own storage management, requesting memory in large chunks as needed
  - nine times last year it was unable to get enough memory to build a whole hour's source table
- Handling *flood* (DoS) conditions
  - `iatmon` discards sources that have been inactive for more than 120 s, and that only sent two or one packets
  - it also reports statistics for 'unanalysed' packets from such sources

Unanalysed source statistics

- For the hour starting at 1700 on 16 Apr 11

source address	122,252,855 from 6..15/8
destination address	122,251,708 to *ta*.186/16
destination port	122,250,778 to port 445
TTL	122,264,055 had TTL = 106
tcp parameters	122,250,864 sent TCP SYNs that had window size 0x4000 and no options

- The statistics above indicate that
  - more than 122 M source addresses were in 12/256 of IPv4 space, and were sent to 1/256 of the telescope space
  - all had the same destination port, TTL and tcp parameters
- It seems clear that these were DoS attacks using spoofed source addresses (122 Mpkt/h !)
- There were very few other sources that were also discarded

`iatmon`'s **two classification schemes**

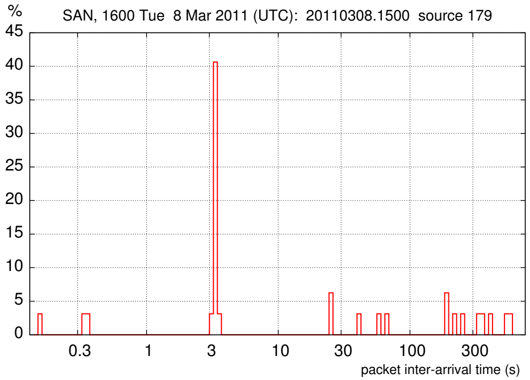
- For each hour's trace, `iatmon`
  - classifies each source in two ways: *type* and *group*
  - writes a summary file for the hour, including
  - $type \times group$ matrices for counts, packets and volumes
- *types*
  - identify common source behaviours using attribute values from the IP header, e.g.:
    - Protocol – TCP/UDP/ICMP only, several protocols
    - Destination address – single destination vs multiple
    - Destination port – single or multiple
    - These are enough to classify common scanning behaviours
    - Backscatter sources send TCP ACK || RST, or ICMP TTL exceeded || destination unreachable packets
    - ICMP sources send only ICMP packets other than above
    - TCP and UDP sources send packets using both protocols
  - we have algorithms that recognise Conficker C and μTorrent packets, we therefore included *types* for those

Source types and (IAT) groups

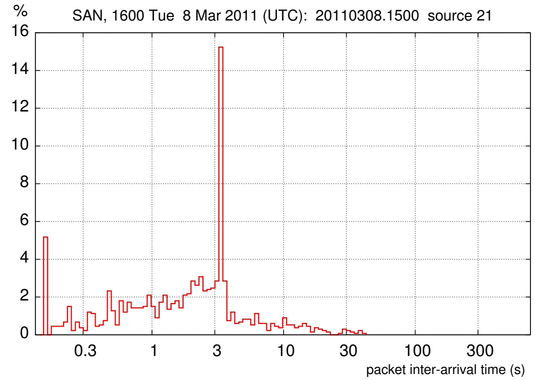
Description	**Type**
TCP	TCP probe
	TCP vertical scan
	TCP horizontal scan
	TCP other
UDP	UDP probe
	UDP vertical scan
	UDP horizontal scan
	UDP other
Other	ICMP only
	Backscatter
	TCP and UDP
	μTorrent
	Conficker C
	Untyped

IAT distribution	**Group**
Long-lived	Stealth +3 s mode
	Stealth + Spikes
	Stealth other
3 s mode	Left-heavy
	Even
	Right-heavy
Other	Short-lived
	High-rate
	DoS
	Ungrouped

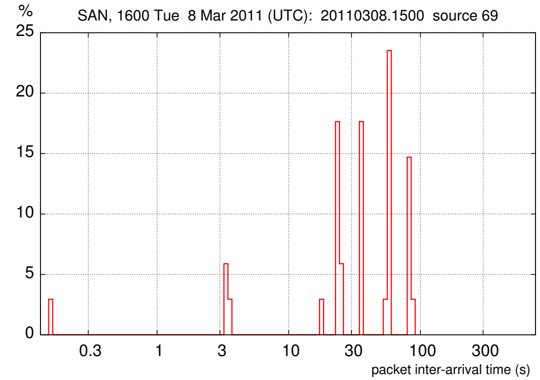
IAT distribution *groups*



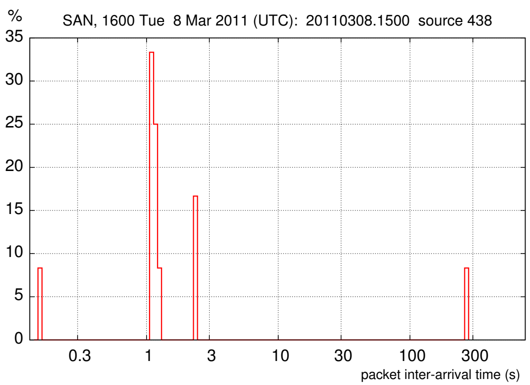
Stealth + 3 s mode



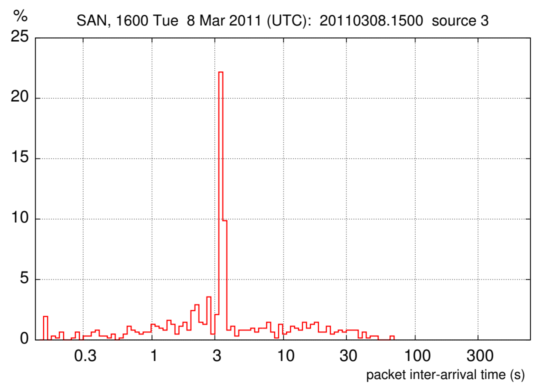
3 s mode left



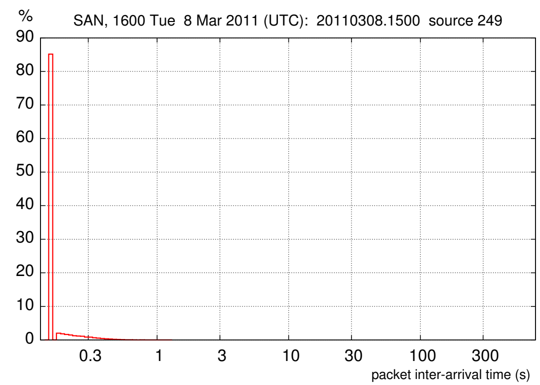
Short-lived ($< \frac{1}{2}\, hour$)



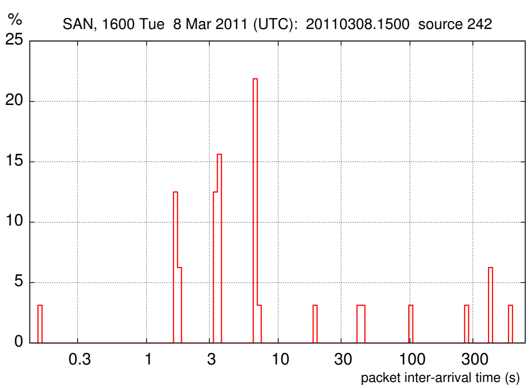
Stealth + spikes



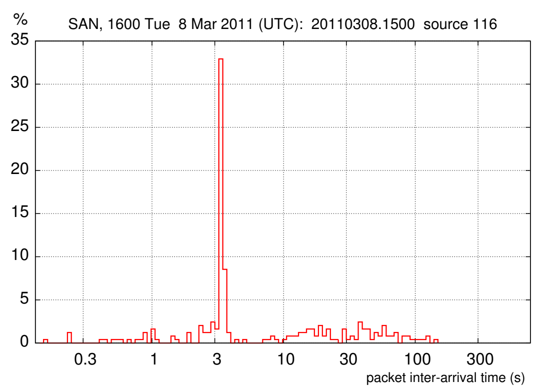
3 s mode even



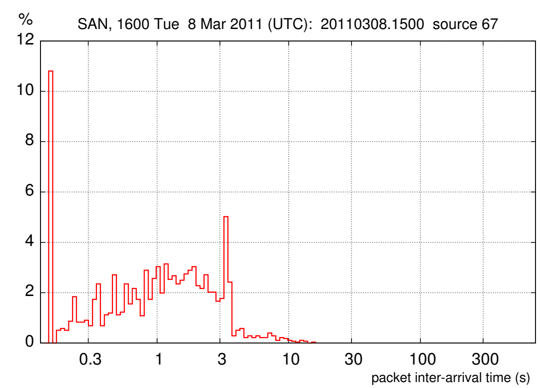
High-rate ($> 5\, p/s$) DoS



Stealth other



3 s mode right



DoS

Example summary matrix (1500, 3 Apr 2011 UTC)

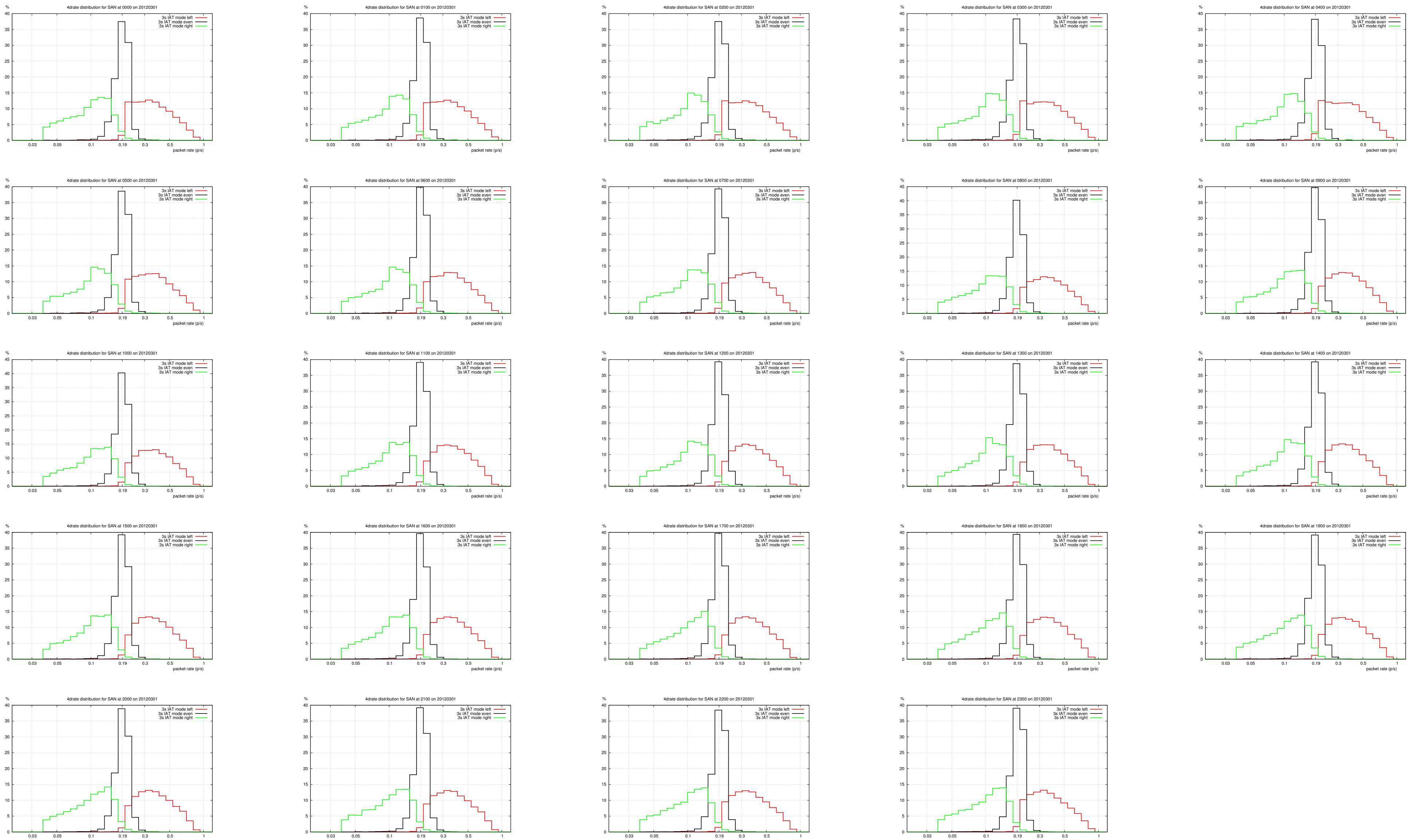
```
#source_counts
 159     715    0    39     4    10    23    30    48    44

 105   31383    1    27     5     5   117  1089   449   496
  14    4487   20    22     1     0     3    43   317   128
 940   61394   29  8270 58387 17040 26685 80182   326  1974
 409   11674  111  1894   221   126   489  4408   618  3657

 613  272654    9  1315     5     3    12   174  4176  5685
   0       0    0     0     0     0     0     0     0     0
 161    2387   69   147     0     0     7    10   444   477
 869  153227    4  6307     7    11    57  2068 12169 17359

2384    1655   14   126    27    34   141     6   304   623
   4      57   39    22     1     1     0     2    11    16
   0    1778    0     0     0     0     0   205    71     0

 437    8598    0  6210     3    24   115  2653   960 16853
  17    2125    2     0     2     1    96 19973   297  4662
```

- *types* are the matrix rows, *groups* the columns
- TCP Horizontal Scans, and UDP probe are commonest
- Bottom two rows are Conficker C and μTorrent

3 s mode *group* simulations

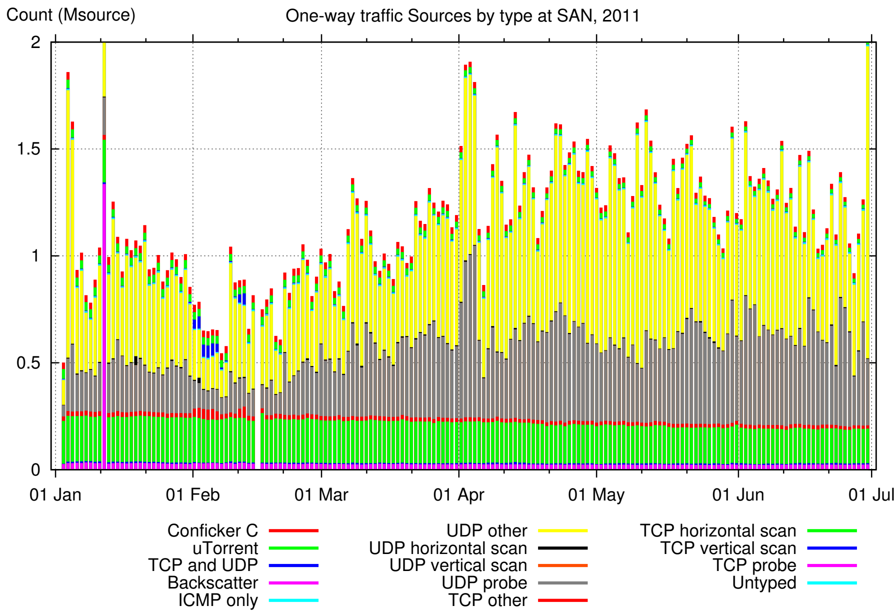
- Question: what kind of process would produce at least some of the *group* IAT distributions?
  - can we test this with a simulation?
- Consider the 3 s mode *groups*
  - two process at least:
    - Try exponential intervals between sending packets
    - 3 s mode comes from retry after 3 s
- Model reproduces distributions in centre column of previous slide
  - left | even | right weight depends on average *packet rate*
  - increasing *rate* values shift distribution weight to left, i.e. decreasing average IAT
- It also works well for the high-rate and DoS *groups*
- Do the *group* IAT distributions vary during a day? ...

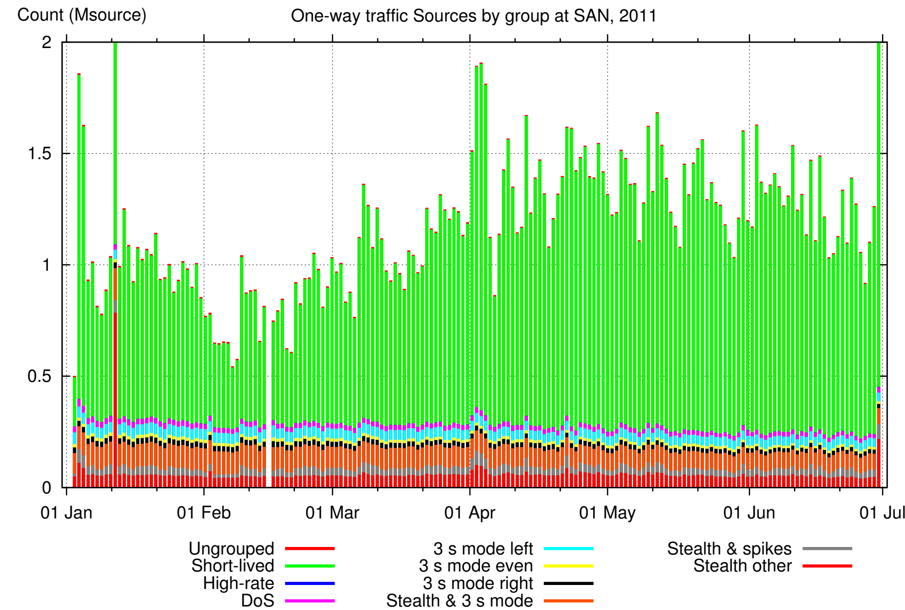
3 s mode *group rate* distributions (for 1 Mar 2011)

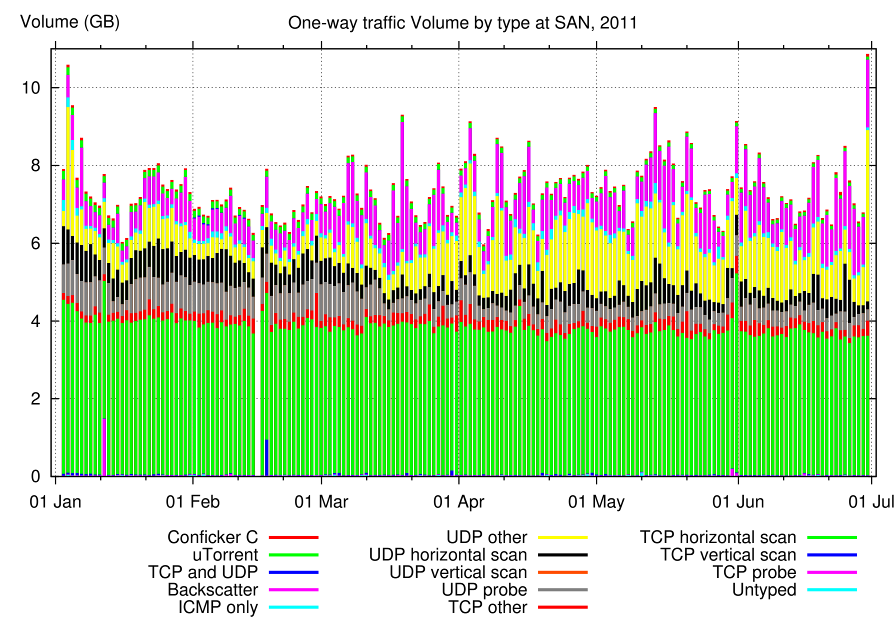


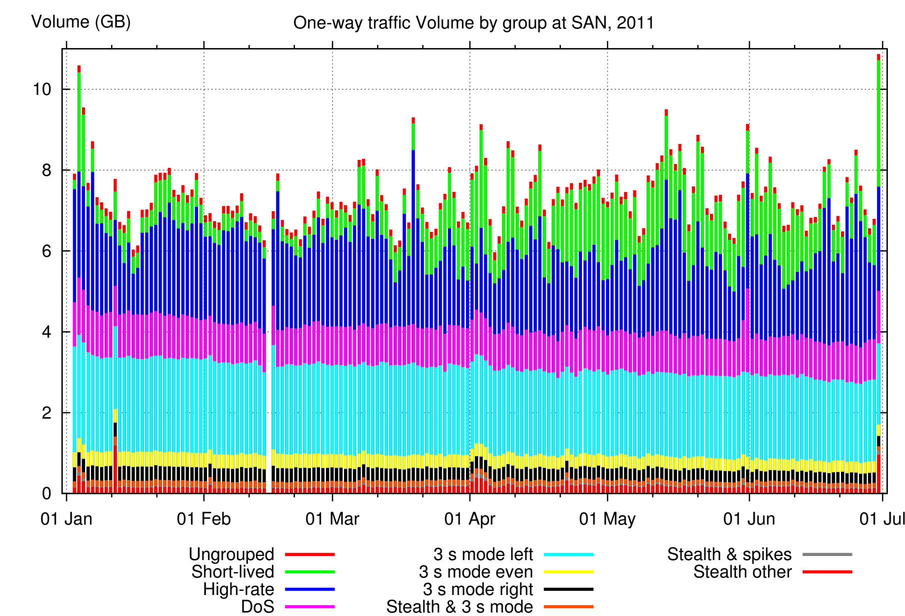
green: mode right, $rate \sim 0.1$ black: mode even, $rate \sim 0.19$ red: mode left, $rate \sim 0.35$ p/s

6-month summary plots, Jan-Jun 2011

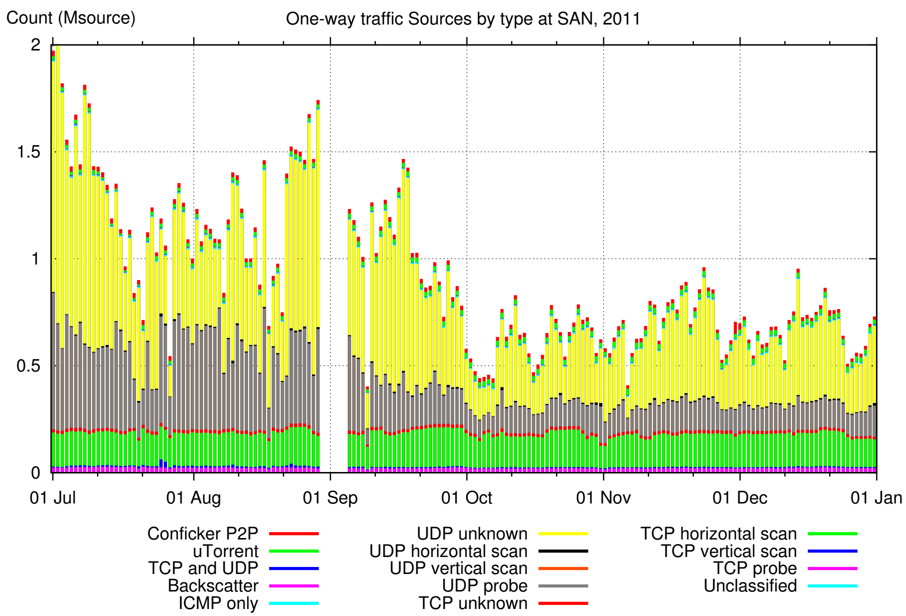


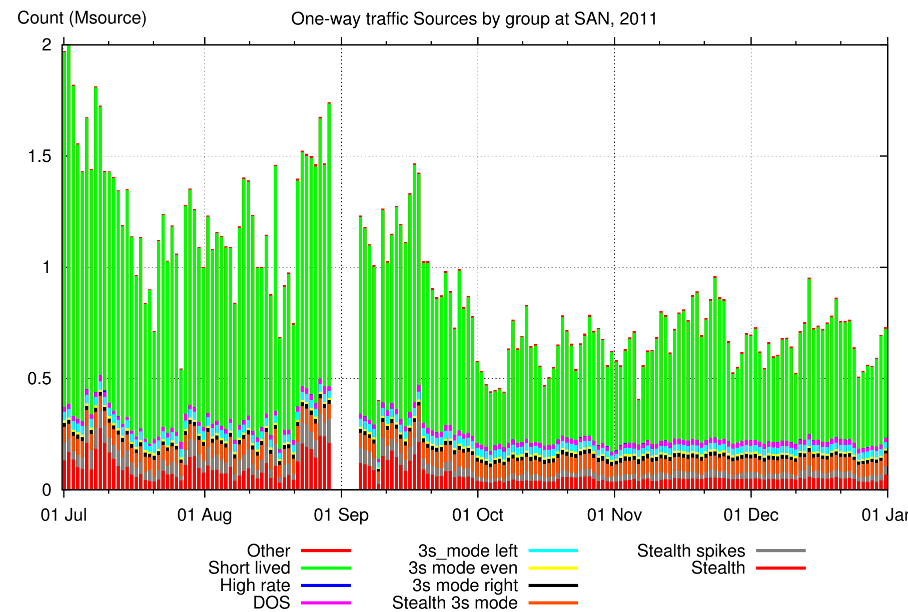


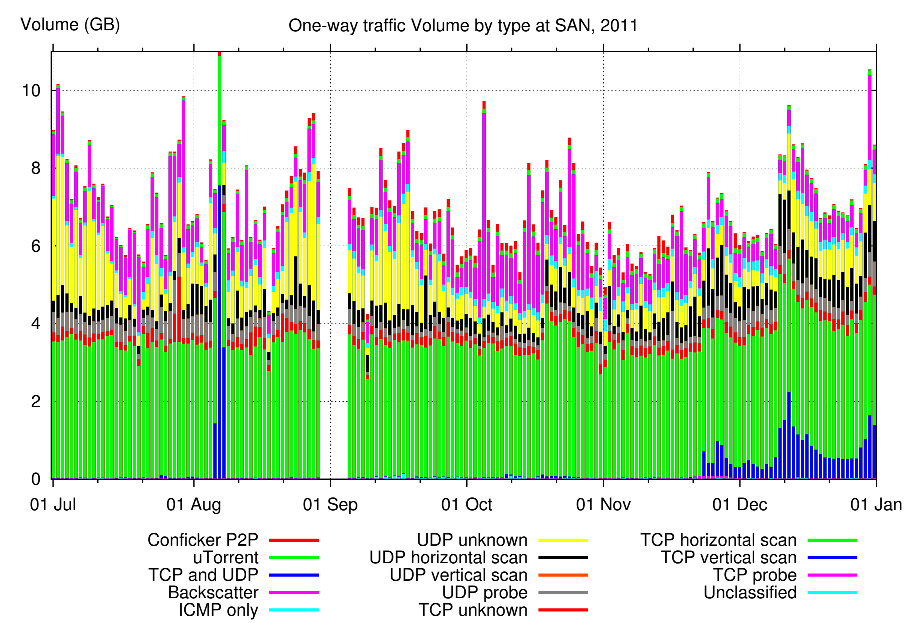


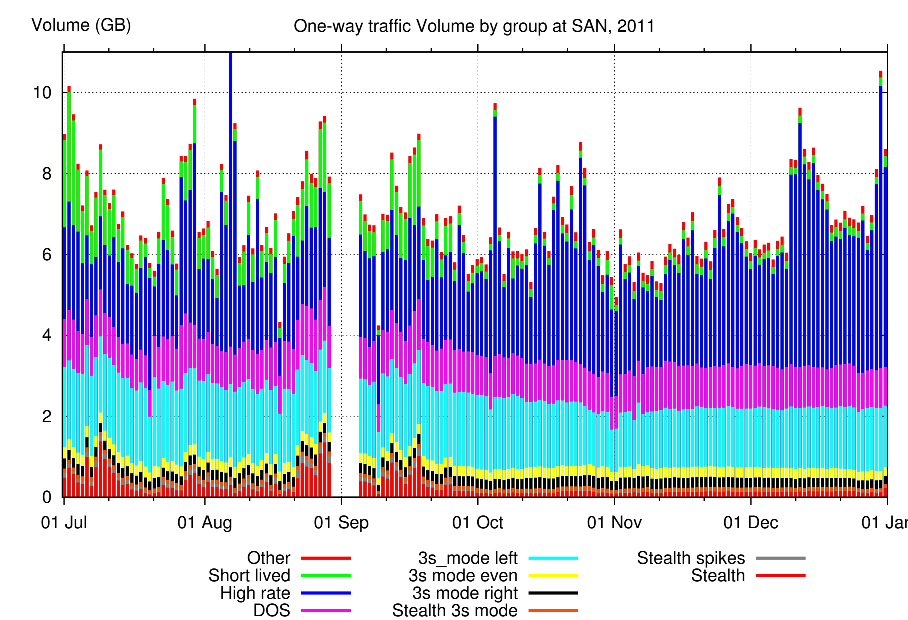


6-month summary plots, Jul-Dec 2011









Could `iatmon` work in *grey* space?

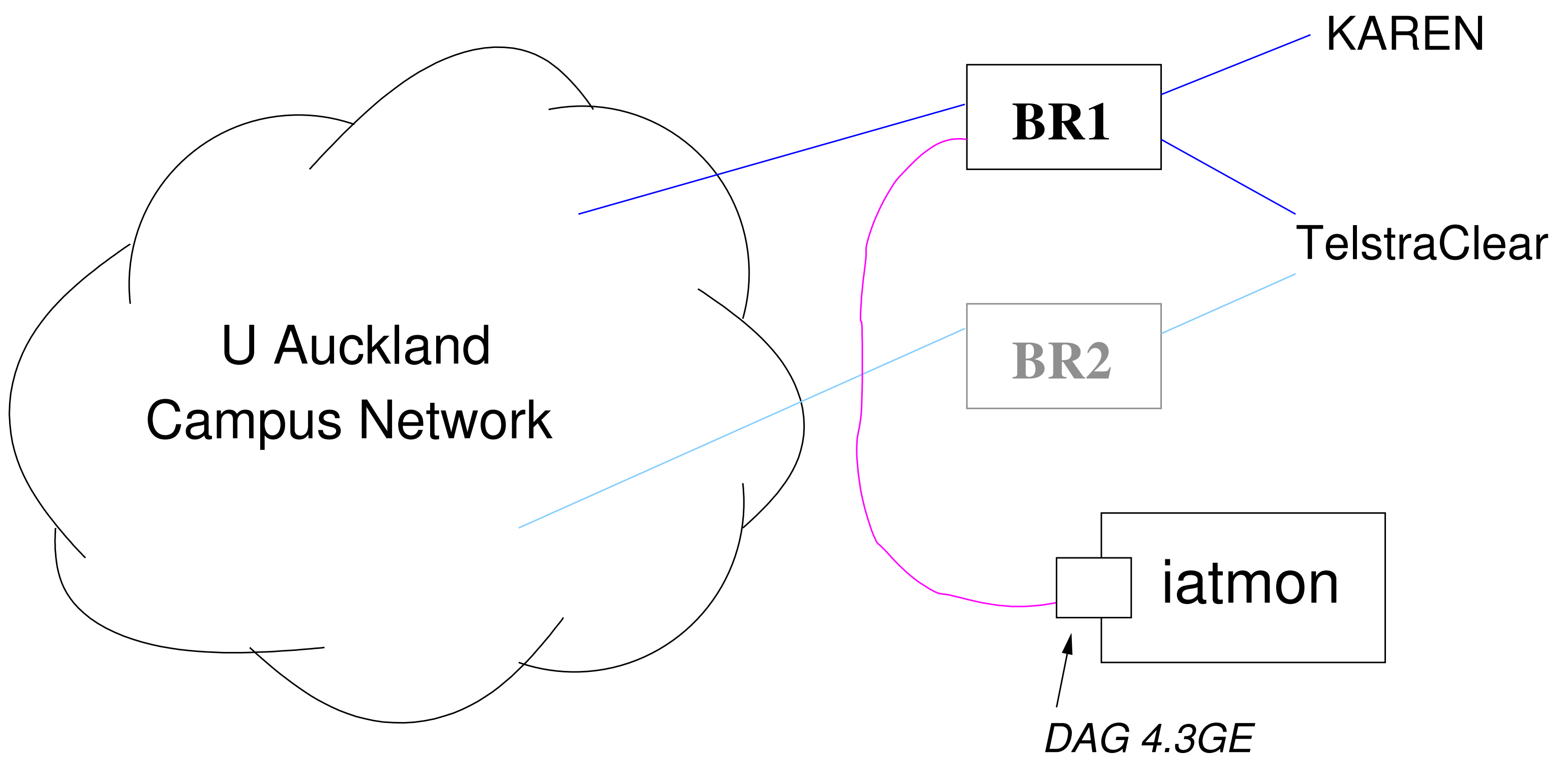
- 'Grey' space means IP address space that contains some active hosts
  - for example, U Auckland network
  - we are 130.216.0.0/16, we have room for about 65,000 hosts
  - however, we only have about 16,000 active hosts
  - we have a little IPv6 traffic, as well as lots of IPv4
- How hard is it to "just filter out the two-way flows?"
  - `iatmon` implementation makes a good platform for this . . .
  - Ruby outer block, easy to write summary files
    - C threads for input packet-watching, output flow-watching
    - uses `libtrace` library (from WAND) to read packets
    - reuses NeTraMet code (from ~2001) for its source info table, and for building IAT distributions
- Using `iatmon` in this way could allow us to collect IPv6 one-way traffic data from many sites!

Development of `iatmon` at Auckland and Trondheim

- Re-used NeTraMet's 'dynamic stream timeout' algorithm
  - assume packet rate is steady for the life of a flow
  - wait for a *minimum time*, determine average IAT
  - multiply that by *timeout multiplier* to get flow's *inactivity time*
  - repeat above from time to time
- Early results at Auckland (October 2011)
  - much less one-way traffic than at UCSD (no surprise there!)
  - a few $type \times group$ subsets with high traffic volumes
  - started to consider what could cause that
- Running stably in production since February 2012
- Ditto for (two-way IPv4) Uninett trace files at Trondheim

More detail on two-way traffic filtering on next few slides

U Auckland network topology

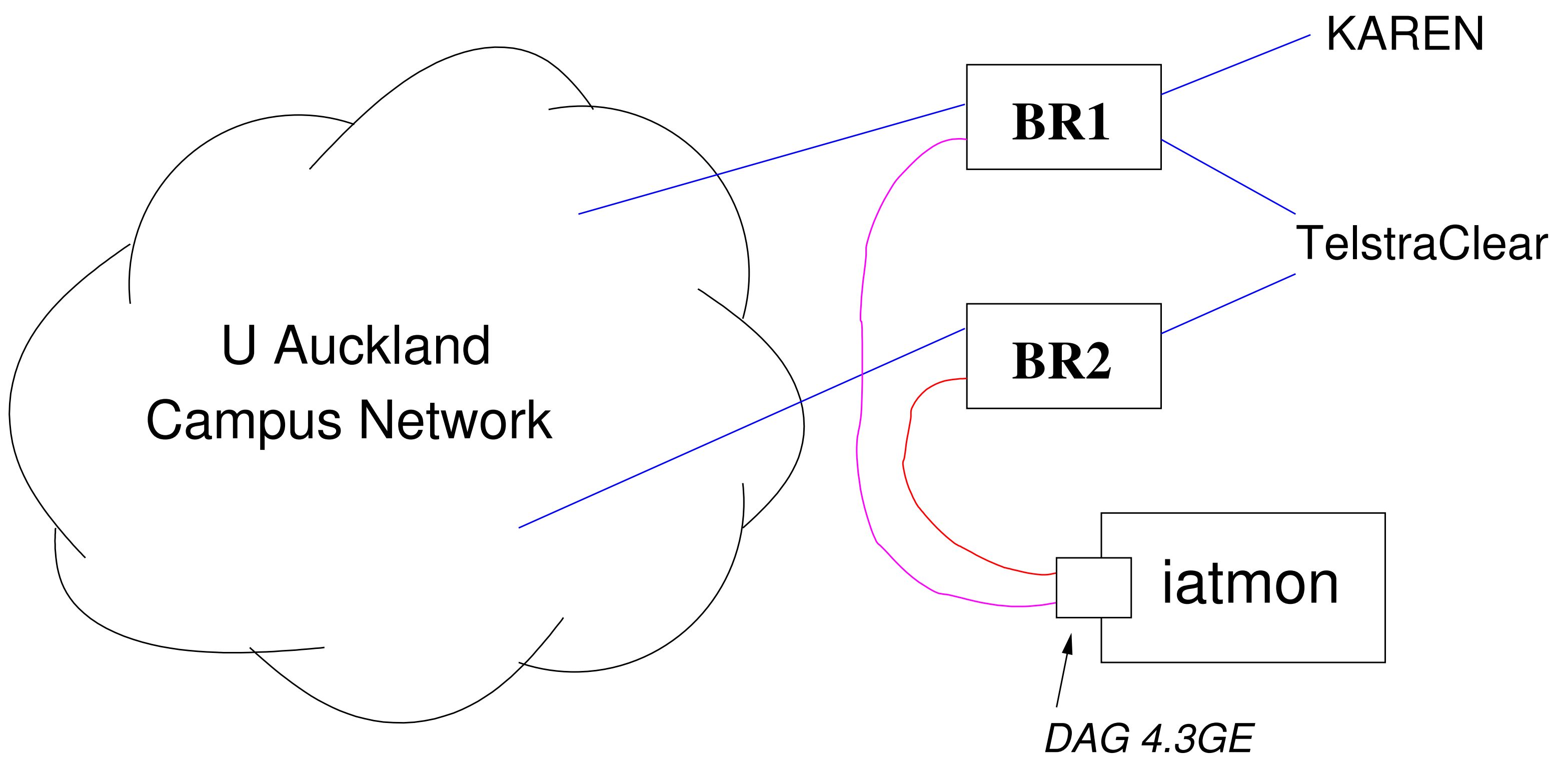


- BR1 and BR2 are configured as primary and secondary
  - BR2 takes over if BR1 fails, mostly it's idle
- Packets through BR1 are copied to a monitoring port, which is connected to a host running `iatmon`
  - the monitor uses a DAG card so as to get accurate time stamps at full line rate

'Half-way' flows

- `iatmon` allows one to select out source subsets
  - collected some sample trace files from BR1, and looked at some flows in the anomalous subsets
- Realised that some of them transfer lots of data –
  - their TCP sequence (SEQ) and acknowledgment (ACK) numbers increase steadily
  - however, we only see packets in one direction
  - I call this kind of flow a 'half-way' flow
- Added code to `iatmon` to watch for half-way flows
  - collect statistics on 1-, 2- and $\frac{1}{2}$-way flows
  - treat half-way flows as though they were two-way
- Discussed results with ITS
  - added a monitoring connection to BR2

Improved monitoring topology

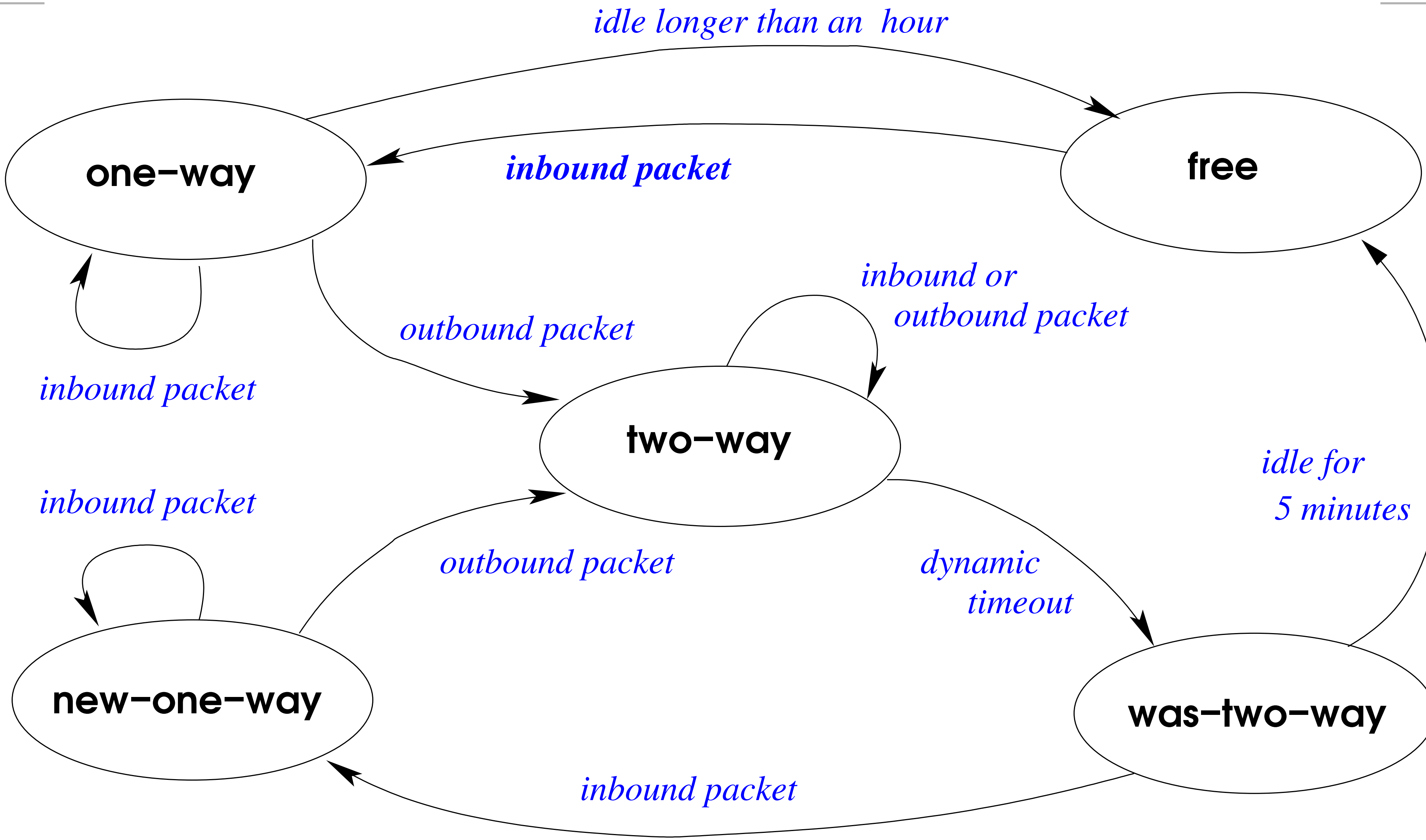


- BR2 is about 0.6 km away from BR1 and `iatmon`
  - connected via a single-mode fibre to second DAG port
- Much better n-way statistics, but
  - still seeing some half-way flows
  - needs some further investigation!

Problems with timing out 2-way flows

- Can run flow-watcher in 1, 2 or 4 threads
  - `iatmon` runs faster with fewer threads
  - need more work to reduce interlocking overheads
- *Noticed that we see more one-way sources when we use fewer threads!*
- Modified `iatmon` to keep all source info, to see which two-way flows are timed out then re-appear as one-way
- Testing on 15 minute trace shows that more packets from an external host can reappear up to 120 s after data transfer finishes, even for sessions that started with a packet out from our network!
- Some such sources (in the test trace) came from a compromised host in the U Auckland network,
  it was part of a spam botnet

iatmon **source-object states**



collecting statistics **counting packets** **not collecting**

Monitoring `iatmon's` performance

- `iatmon` writes a log file with '#' records
  - #Stats: list of xxx=nnn values for system parameters
    written every minute
  - #SrcStates: number of flows in each state
    written every hour
- We looked at plots of these parameters for U Auckland in the following slides
  - packet rates – total packet rate through BR1 + BR2
    - average around 100 kp/s around 0100 UTC (1400 NZDT)
    - maximum as high as 138 kp/s around midday (NZDT),
      occasional high maxima at various times
    - no packet losses reported by DAG card
  - maximum source-objects in use
    - drops at end of each hour, climbs during hour
    - daily peaks around early afternoon, 300 to 400 sources
      occasional high maxima at various other times

Conclusion

- `iatmon` works well, at UCSD and Trondheim on trace files, and at Auckland on a live 1 Gb/s interface
- It's two classification schemes separate the sources into 140 subsets, making it easier to notice changes
- $type \times group$ subsets of the packets can easily be separated out from a large trace file for detailed analysis
- There is *lots* more monitoring and analysis work to do, e.g.:
  - establish more `iatmon` monitors at other sites
  - explore data mining techniques to detect changes in $type \times group$ subset behaviour
  - explore worm behaviour, so as to better explain the $IAT groups$. (More simulations? Metasploit??)
  - what are the UDP sources? Why are there so many of them?
- URL in PAM 2012 paper is incorrect, `iatmon` is available online at http://www.caida.org/tools/measurement/iatmon/