TELE9752 Network Operations and Control

Lecture 5: Remote monitoring

Outline

- Network monitoring
  - RMON1 and 2
- RMON tables
- Protocol statistics (`etherStats, protocolDist`)
- Notifications (`alarm, event`)
- Hosts
  - traffic per host (`Host, nlHost, alHost`)
  - addresses (`addressMap`)
  - top talkers (`hostTopN`)
  - traffic matrices (`nlMatrix, alMatrix`)
- Packet capture (`filter, channel, capture`)
- Probe config (`protocolDir, probeConfig`)

References

Section 9.1.1 of D. Mauro and K. Schmidt: *Essential SNMP*, O'Reilly

W. Stallings: *SNMP, SNMPv2, SNMPv3 and RMON 1 and 2*

over 100 pages on RMON, for better or worse

RFCs:

- 3577 overviews the RMON family (v1 and 2, as well as SMON for link layer switching, ...)
- 2819 defines RMON1
- 4502 defines RMON2

Network monitoring goals

Traffic on networks can be monitored to determine

- **Hosts**:
  - Which are new? Possibly shouldn't be attached
  - "**Top talkers**": Which generate the most traffic?
    - Excess traffic may indicate malware
    - Target the few that create most traffic† for cost cutting
- **Traffic matrices**: Which combinations of source/destination (rows/columns of matrices) create traffic?
  - Useful for provisioning: Measure demand; co-locate talkative pairs on same subnet
- **Traffic statistics**:
  - How are packet lengths distributed?
  - What protocols dominate?
  - What patterns are evolving?

† http://www.nytimes.com/2012/01/06/technology/top-1-of-mobile-users-use-half-of-worlds-wireless-bandwidth.html

Terms...

monitor: consists of

- **probe**: A system that implements the RMON MIB
- **collector**: Often just capture and little analysis
- **analyzer**: Examines existing data

Monitor can be a

- network element
- a component of a network element (e.g. card in router)
  - often records *source* of observations, e.g. which port/interface was a particular host observed on
- device connected to a network
- device connected to a network element, e.g. using port mirroring

Remote Network Monitoring

Remote Network Monitoring: device on network gathers and processes data; results viewed remotely.

RMON essentially defines new MIBs for monitoring.

IETF standards

- **RMON1**: RFC 2819 (STD 59), originally RFC1757
  - Focuses on link layer stats.
  - A reasonable start: all packets go over link
- **RMON2**: RFC 4502, originally RFC 2021
  - Covers network layer ("nl") and above (misleadingly called "application layer" "al" e.g. alHost [9X>)
  - Doesn't replace v1; mainly complementary functions

Flow monitoring mechanisms (NetFlow, IPFIX) came later [K9>

Wikipedia is/was wrong in saying ""In short, RMON is designed for "flow-based" monitoring, while SNMP is often used for "device-based" management" RMON uses SNMP & flows differ [8Y>

2KL

RMON1 groups

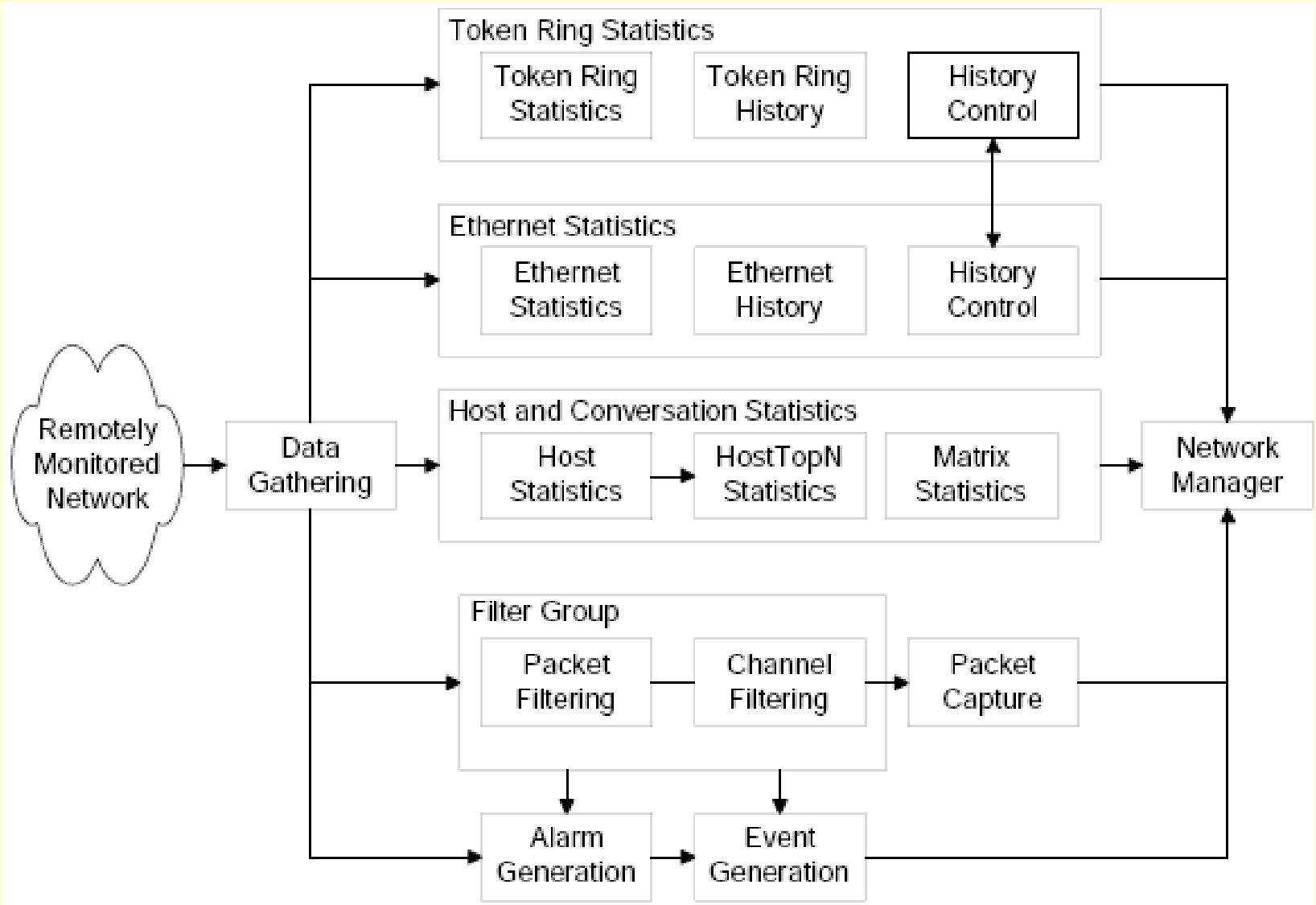


Figure 8.3 RMON1 Groups and Functions

Fig from Subramanian

2GQ

Outline

- RMON tables
- Protocol statistics (`etherStats, protocolDist`)

Sharing RMON tables

- Monitoring can be resource-intensive & expect multiple managers to need monitors (e.g. local & global NOC <TX]) => need to share access
- **Arbitration**:
  - RMON was introduced before SMI v2's `rowStatus` <66] => has its own way to arbitrate access to rows
  - `EntryStatus` column is one of {`valid, createRequest, underCreation, invalid`}
    - c.f. SMIv2 `rowStatus` {`active, notInService, notReady, createAndGo,createAndWait,destroy`}
- Column to identify current `Owner`, in case they don't release the row after reasonable time. e.g. slide [1H]

(Another common row field, unrelated to sharing: Most rows also have a unique `Index`† column, used for cross referencing (except traffic matrices))

2ZL † Unrelated to the INDEX parameter for tables <VD] which indicates how rows are identified

Control and data tables

Most remote monitors use 2 types of tables:

- **control tables**: To configure the monitor:
  - indicate source of measurements
    (e.g. which interface)
  - how much is to be collected
  - how often to sample
  - where to report events
  - etc

  e.g. slide [3P>
- **data tables**: hold measurements

  e.g. slide [1H>

etherHistory control and data tables

3 tables:

- `historyControlTable` specifies over what intervals to record `etherStatsTable`
- `etherStatsTable` contains current stats
- `etherHistoryTable` contains `etherStatsTable` data from past intervals.
  Each row contains objects called `etherHistory...`
  - `...Index` <ZL] to refer to `historyControlTable`
  - `...SampleIndex`: which interval does this table record?
  - `...IntervalStart`: time of start of interval [3P>
  - `...Utilization`: Simple arithmetic on pkt & byte counts & interval

`historyControl` also applies to Token Ring stats

RMON1 EtherStats data tables

```
EtherStatsEntry ::= SEQUENCE {
  etherStatsIndex          Integer32,
  etherStatsDataSource   OBJECT IDENTIFIER,
  etherStatsDropEvents     Counter32,
  etherStatsOctets         Counter32,
  etherStatsPkts           Counter32,
  etherStatsBroadcastPkts  Counter32,
  etherStatsMulticastPkts  Counter32,
  etherStatsCRCAlignErrors Counter32,
  etherStatsUndersizePkts  Counter32,
  etherStatsOversizePkts   Counter32,
  etherStatsFragments      Counter32,
  etherStatsJabbers        Counter32,
  etherStatsCollisions     Counter32,
  etherStatsPkts64Octets   Counter32,
  etherStatsPkts65to127Octets Counter32,
  etherStatsPkts128to255Octets Counter32,
  etherStatsPkts256to511Octets Counter32,
  etherStatsPkts512to1023Octets Counter32,
  etherStatsPkts1024to1518Octets Counter32,
  etherStatsOwner    OwnerString,
  etherStatsStatus   EntryStatus
}
```

```
EtherHistoryEntry ::= SEQUENCE {
    etherHistoryIndex
    etherHistorySampleIndex
    etherHistoryIntervalStart
    etherHistoryDropEvents
    etherHistoryOctets
    etherHistoryPkts
    etherHistoryBroadcastPkts
    etherHistoryMulticastPkts
    etherHistoryCRCAlignErrors
    etherHistoryUndersizePkts
    etherHistoryOversizePkts
    etherHistoryFragments
    etherHistoryJabbers
    etherHistoryCollisions
    etherHistoryUtilization
}
```

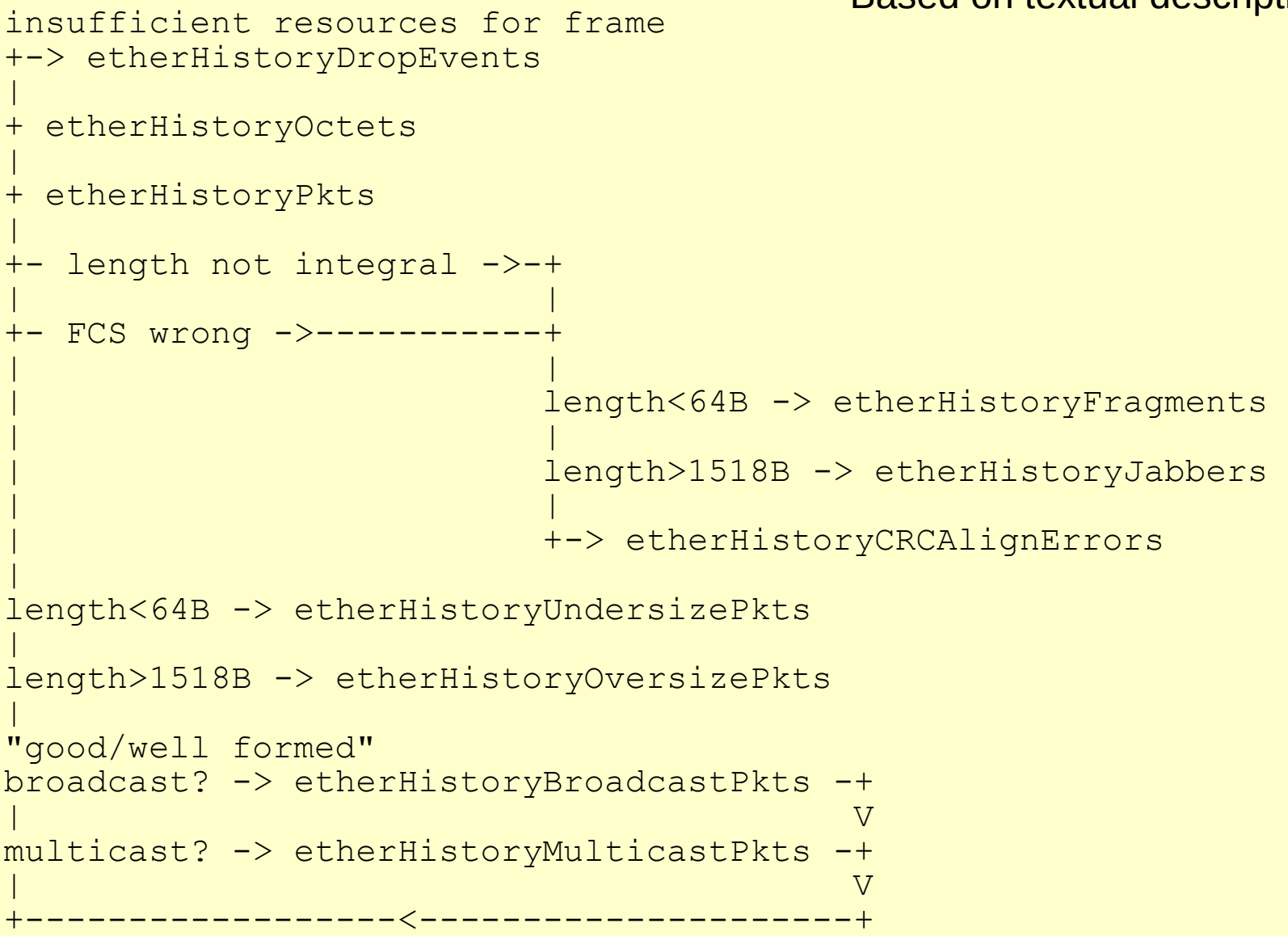
Critical thinking:

- Why doesn't history record length distribution?
- Why use a separate table for history rather than just recording history as separate rows in EtherStats table?
- Why noOwner/Status for HistoryEntry?

21H} Undersize = too short; aka “runts”. fragments = Undersize + invalid CRC != IP fragments jabber = oversize & invalid CRC

Case diagram for `EtherHistoryEntry`

Based on textual descriptions from RFC1271



2!!!

EtherStats discussion

Other points:

- `etherStatsPkts` records distribution of frame lengths
- History of `etherStats` can be recorded using `HistoryControl` [3P> + `etherHistoryEntry`
- There are similar `tokenRing` extensions
- RMON2 has `protocolDist`, recording pkt/byte count for arbitrary protocols.

```
HistoryControlEntry ::= SEQUENCE {
  historyControlIndex                Integer32,
  historyControlDataSource           OBJECT IDENTIFIER,
  historyControlBucketsRequested     Integer32,
  historyControlBucketsGranted       Integer32,
  historyControlInterval             Integer32,
  historyControlOwner                OwnerString,
  historyControlStatus               EntryStatus
 }
```

`DataSource`: which Ether interface (listed in `interfaces` group of IP MIB)

`Interval` (in seconds) over which stats are recorded

- e.g. short-term (e.g. 30sec) deviation from long-term (30min) average often highlights abnormalities

`Buckets`: Storage containers. Each interval is stored in a bucket, but buckets get recycled once filled with data from past intervals.

- `BucketsGranted` <= `Requested` due to memory limitations.
- e.g. 6 granted 30sec interval => 1st bucket reused after 180sec

History example

Packets arrive every second, length in bytes:

3 1 4 1 5 9 2 6 5 3 5 8 9 7 9 3 2 3 8

Measure bytes

3 buckets, 4 s

B1:

3 4 8 9

B2:

\- - - - **5 14 16 22** *22 22 22 22 22 22 22 22 22* **2 5 13**

B3:

\- - - - - - - - **5 8 13 21** 21 21 21 21 21 21 21

Boldface = current interval, - = invalid

Average of 9, 22, 21, 28 = 20 bytes/4sec = 5B/s

?!!!

RMON2 `usrHistory`

Generalises `etherHistory`, allowing recording of history of any `INTEGER`-derived object (`Integer32, Counter, Gauge, TimeTicks`)

- `usrHistoryObjectVariable` => which object

More complicated control (2 control tables + data table):

- `usrHistoryControlTable` specifies interval & # of buckets
- `usrHistoryObjectTable` controls
  - *which* objects (OID)
  - *how* to sample, e.g. values 1 4 8
    - `absoluteValue`: e.g. values 1 4 8, e.g. for total pkts
    - `deltaValue`: e.g. values - 3 4, e.g. for pkts/sec

RMON vs protocol MIBs

Both RMON and link layer protocol MIBs collect link layer statistics. How do they differ?

RMON observes **the link**, including traffic to/from **any element** that uses the link.

Protocol MIBs observe **the element's use of the link**, excluding traffic exchanged by other elements.

e.g. Ethernet statistics (collisions, CRC errors etc) are measured by both:

- Dot3Stats MIB: Traditional protocol MIB: observes behaviour *in managed network element* (e.g. collisions when it tx/rx)
- **EtherStats in RMON:** e.g. collisions involving *any* tx/rx
  - `etherStatsDataSource` records port/interface

Outline

- Notifications (`alarm, event`)

`alarm` group

Alarms monitor object values and trigger events if they exceed thresholds

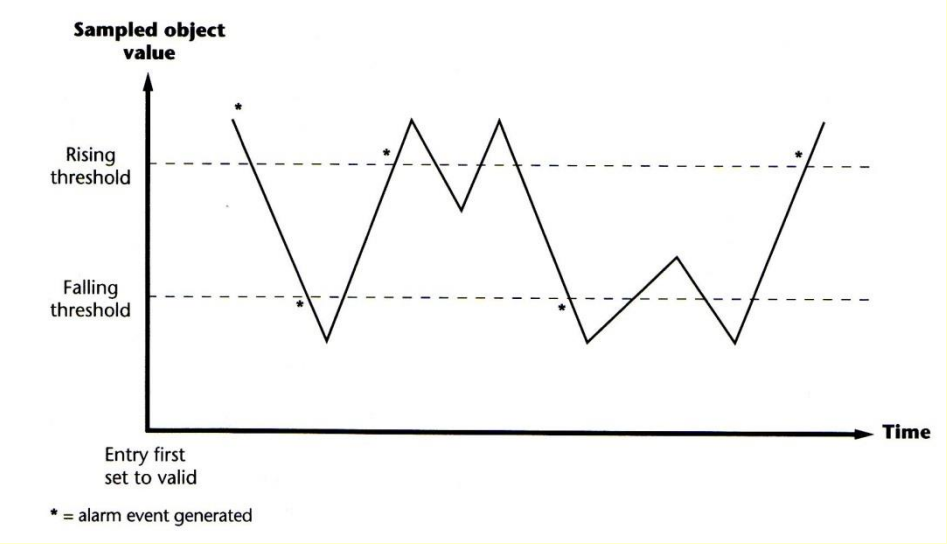
```
AlarmEntry ::= SEQUENCE {
  alarmIndex              Integer32,
  alarmInterval           Integer32, timeTicks between samples
  alarmVariable           OBJECT IDENTIFIER, which RMON1
                                                 object to monitor
  alarmSampleType         INTEGER, absoluteValue or deltaValue
  alarmValue              Integer32, last sampled value
  alarmStartupAlarm       INTEGER, alarm on rising, falling or both?
  alarmRisingThreshold    Integer32,
  alarmFallingThreshold   Integer32,
  alarmRisingEventIndex   Integer32,  indexes eventTable  to
                                                 define response
  alarmFallingEventIndex  Integer32,
  alarmOwner              OwnerString,
  alarmStatus             EntryStatus
}
```

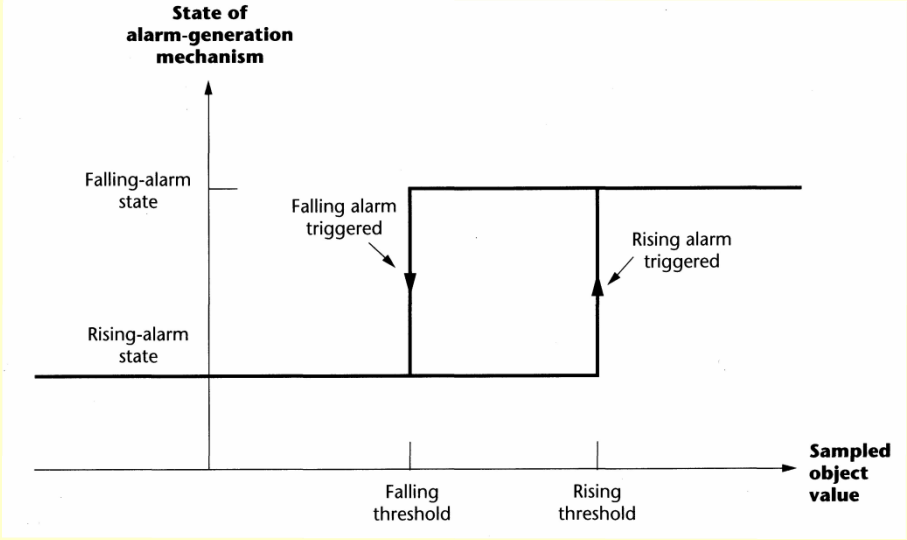
Engineering principle: hysteresis

Avoid frequent switching when condition is near target by

- defining 2 separate thresholds
- next switch according to one threshold only occurs after passing other threshold.
- Smaller variations do not cause switch.

e.g.





- heater turning on/off when temperature near target
- NM: one alarm when cross target, rather than many

2LJ Figures from W. Stallings: *SNMP, SNMPv2, SNMPv3 and RMON 1 and 2*, pp. 254-5

Alarming discussion

- Alarms can monitor any RMON variable
  - etherStats
  - Host/matrix data
  - Output of packet capture filters
  - Does RMON2 extend alarms to cover OIDs outside RMON? (Get a bonus mark if you answer this!)
- Alarms can trigger events

`event`† group

- **events are triggered** by conditions occuring elsewhere in the MIB, e.g.
  - Alarm <64]
  - packet passing a filter [9M>
- **events trigger actions** of different `eventType`s:
  - log in a `logTable`
  - `snmp-trap`
    - `eventCommunity` defines NMS that can receive it, but RMON1 doesn't seem to define dest addr. RMON2 `probeConfig` defines `trapDestTable`
  - turn `channel` [U2> (used for counting or capturing packets) on or off
    - not indicated in `event` as such, but in `channelTurn[On|Off]EventIndex`

† The events here are in the context of RMON, and differ from the events in the context of fault management

Outline

- Hosts
  - traffic per host (`Host, nlHost, alHost`)
  - addresses (`addressMap`)
  - top talkers (`hostTopN`)
  - traffic matrices (`nlMatrix, alMatrix`)

Host discovery & monitoring

Hosts are discovered from addresses in “good” packets seen being sent across the network

- next slide [TE] defines “good”, “bad” & implications

Host group keeps stats for each such discovered host:

- **`hostControlTable`** specifies `DataSource` (interface) & size of table
- 2 **data tables** provide different access:
  - **`hostTable`** is indexed by `Index&Address`
    e.g. what has host H been up to?
  - **`hostTimeTable`** indexed by `Index&CreationOrder`
    e.g. what's new?

We'll return to device discovery in Config Mgt lecture [N8]

The good, the bad and the RMON

NOC is often concerned with faults and errors.
Useful to have counts of errored packets, but how well can you classify errored packets?
e.g. what if address contained a bit error?

RMON defines:

- Good packets: error-free & have a valid frame length
- Bad packets:
  - have proper framing => recognized as packets (e.g. Ethernet: valid preamble and SFD), but
  - have errors within the packet or invalid length (e.g. Ethernet: bad CRC, <64B or >1518B)

Host group

Counts of bad pkts assume that address was not bad in packet

```
HostEntry ::= SEQUENCE {
  hostAddress          OCTET STRING,
  hostCreationOrder    Integer32,
  hostIndex            Integer32,
  hostInPkts           Counter32,    good pkts to this host
  hostOutPkts          Counter32,    good/bad pkts from this host
  hostInOctets         Counter32,
  hostOutOctets        Counter32,    in good/bad pkts
  hostOutErrors        Counter32,    bad pkts
  hostOutBroadcastPkts Counter32,  good pkts only
  hostOutMulticastPkts Counter32   good pkts only, no b'cast
 }
```

In contrast, RMON2 `nlHost` and `alHost` only count *good* packets

RMON2 `addressMap`

RMON1 defines hosts in terms of link layer (“Physical”) addresses

RMON2:

- provides `nlHost` (defined by IP address) and `alHost` (defined by protocol within IP)
- maintains **`addressMap`** to indicate binding of Physical and IP addresses
  - populated from monitoring transmissions
    - monitor need not have ARPed address of interest, unlike address translation MIB <N5]
  - for each `NetworkAddress` seen, `addressMapEntry` indicates last seen `PhysicalAddress` and time of `LastChange` (recent may suggest IP address duplication).

hostTopN

hostTopN refines Host data: identifies TopN hosts that caused most traffic in an interval.

- Saves NMS from downloading all Hosts data to determine this
- Requires Host group be implemented

Normal control & data tables:

- hostTopNControlTable: main object is hostTopNRateBase => which host stat (e.g. in/out pkts/octets) should be used for ranking
- hostTopNTable records data about top talkers, identifying them by link layer address

`matrix` group

Traffic matrices record statistics about conversations between pairs of addr's.

(“conversation” here = any communication @ any time; no distinction of different sessions.)

```
MatrixSDEntry ::= SEQUENCE {
  matrixSDSourceAddress OCTE
  matrixSDDestAddress   OCTE
  matrixSDIndex         Inte
  matrixSDPkts          Coun
  matrixSDOctets        Coun
  matrixSDErrors        Coun
}
```

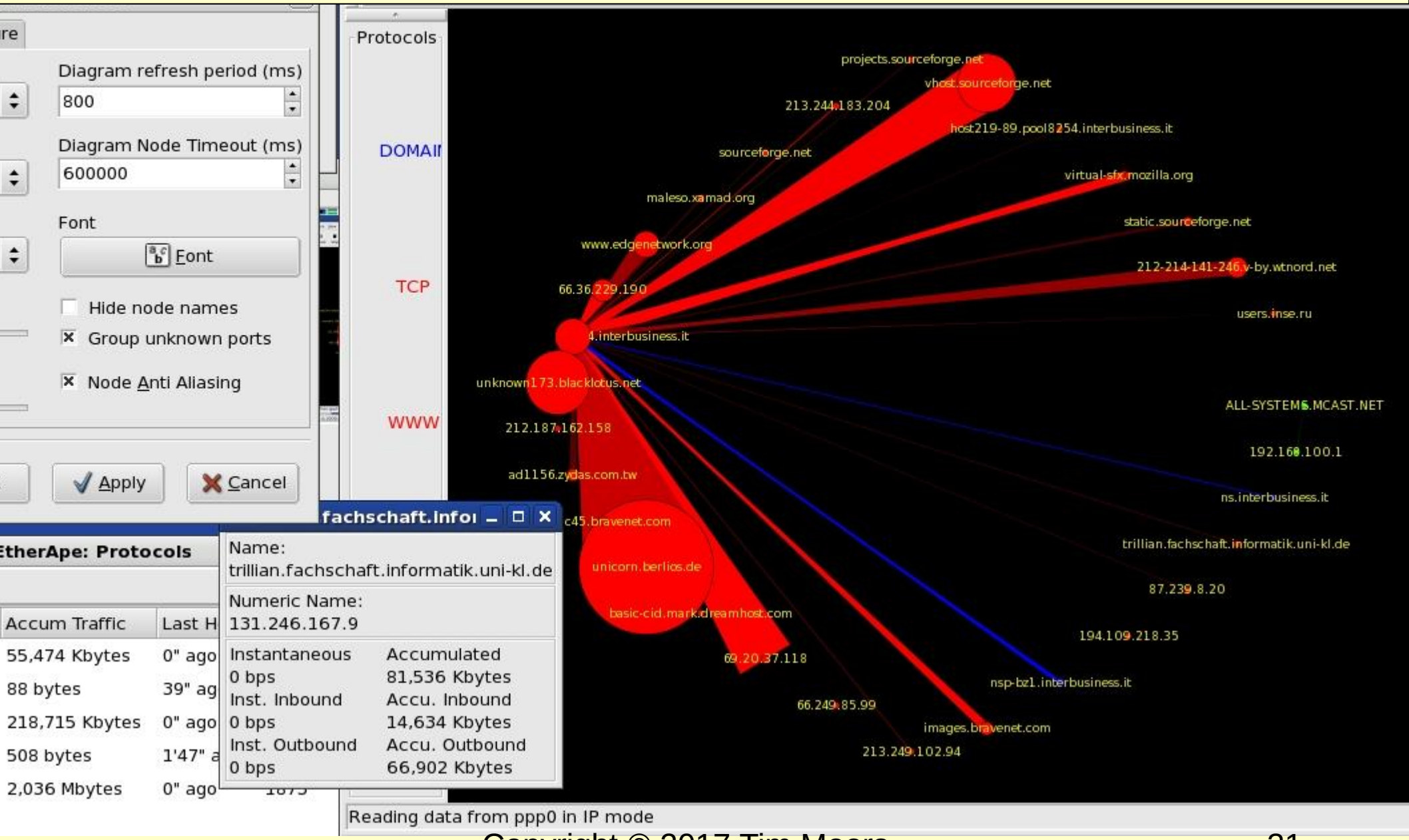
Stats recorded: # of `Pkts` (good & bad), `Octets`, `Errors`

Data stored in two tables:

- `matrixSDTable`: indexed by source then dest'n
- `matrixDSTable`: indexed by dest'n then source

=> more efficient GetNext access when NM is concerned with a particular source (`SD`) or destination (`DS`)

Visualisation of traffic matrices



2T2 EtherApe screenshot from http://etherape.sourceforge.net/images/v0.9.3.png

Outline

- Packet capture (`filter, channel, capture`)

RMON packet capture & filtering

Packet **capture** (aka “sniffing”) gives insight into
- erroneous behaviour
- what protocols are being used & by whom

`tcpdump` and Wireshark [G6] provide *local* packet capture and filtering

Often need to **filter** captured data to
- limit memory needed (“capture filters”)
- highlight packets of interest (“display filters”)

RMON passes all observed packets through `filter`s. Output of filter forms a **`channel`**, for which pkts will be counted & may trigger an `event` or be `capture`d

`FilterEntry` definition

```
FilterEntry ::= SEQUENCE {
   filterIndex                    Integer32,
   filterChannelIndex             Integer32,
   filterPktDataOffset            Integer32,
   filterPktData                  OCTET STRING,
   filterPktDataMask              OCTET STRING,
   filterPktDataNotMask           OCTET STRING,
   filterPktStatus                Integer32,
   filterPktStatusMask            Integer32,
   filterPktStatusNotMask         Integer32,
   filterOwner                    OwnerString,
   filterStatus                   EntryStatus
}
```

Filters

`...ChannelIndex`: Which channel does this filter feed?

2 types of filters:

- Data filters: based on *content* of packet
  - `...Offset` indicates where to start matching. Where to stop implied by length of target (`...Data`)
- Status filters: based on protocol processing of packet. `filterPktStatus` is a bitmap:
  - bit 0: pkt > 1518B?
  - bit 1: pkt < 64B?
  - bit 2: pkt has CRC or alignment error

Filter masks

`...Mask` specifies (with 1s) which bits are "relevant" to the filter

`...NotMask` specifies whether the bit is relevant† because it needs to match (0) or mismatch (1) pktData

actual	pkt	0	0	0	0	0	0	1	1	1	1	1	1	0
target	pktData	0	0	0	1	1	1	0	0	0	1	1	1	1
	Mask	0	1	1	0	1	1	0	1	1	0	1	1	nonsensical
	NotMask	0	0	1	0	0	1	0	0	1	0	0	1	
	Result	Y	Y	N	Y	N	Y	Y	N	Y	Y	Y	N	

In C code: Result = ~((pkt&Mask)^(pktData&Mask)^NotMask)

i.e. only consider pkt & pktData if Mask bit is set.

If Mask not set, NotMask should also be not set & result = true/Y.

If Mask set, then xoring (^) pkt^pktData (i.e. whether bits differ) with NotMask

= 0 if differ&NotMask==1 or 1 if same&notMask==0

† Similar to ternary values in TCAMs if you're familiar with them, e.g. from TELE9751

An applied example

Email servers may blacklist your IP address block (149.171/16) if they receive spam. You want to detect email (SMTP port 25) leaving your address block except from your authorised server 149.171.1.2.

Filter: Mask = 1 for ip.src & tcp.dst (0 for tcp.src & other bits)
NotMask=0 for ip.src MS 16b & 1 for LS 16b, 0 for tcp.dst
PktData: ip.src=149.171.1.2, tcp.dst=25

Packets being processed: (green matches, red doesn't)

ip.src=123.45, tcp.dst=25	incoming email
ip.src=149.171.1.2, tcp.dst=25	authorised outgoing email
ip.src=149.171.3.4, tcp.dst=25	unauthorised outgoing email
ip.src=149.171.3.4, tcp.dst=80	web browsing not email

Similar functionality could be achieved by firewall logging [AP>.

`capture` group

Has the usual control & data tables:

- `bufferControlTable`:
  - Size of buffer in bytes (Requested/Granted)
  - `CaptureSliceSize`: How many packet bytes to store
    - e.g. often only care about headers (Eth+IP+TCP)
  - `DownloadSliceSize` & `Offset`: Where do SNMP GET* requests start retrieving packet & how many bytes does each request retrieve
- `captureBufferTable`:
  - `PacketData`
  - `PacketLength`
  - `PacketTime`

Outline

- Probe config (`protocolDir, probeConfig`)

RMON2 `protocolDir` ectory

`protocolDir` ectory provides a table of protocols supported by the monitoring element

- Protocols **identified** by list of demultiplexing identifiers that lead to the protocol, e.g. Ethertype, IP Protocol, transport layer port #
- **Capability**: e.g. can the monitor:
  - correctly count higher layer PDUs when this protocol segments & reassembles?
  - decode protocol header? (to locate higher layer protocol header?)

RMON2 `probeConfig`

Enables remote configuration of a monitor

- `probeDownload`*: Controls boot image used by monitor, e.g. which file, from which server, download to durable (flash EPROM) or volatile (RAM) memory?, status of boot image
- `serial`*: Configure serial (console <4U]) interface on monitor.
- `net`*: IP configuration for interfaces (`IpAddress, SubnetMask, Gateway`)
- `TrapDest`: `Community`, (destination) `Address`, `Protocol` (IP or IPX)

Summary

- (Remote) Monitoring allows observation of traffic “on-the-wire” with element processing to reduce volume sent to NM.
- Most monitoring groups have control tables (=> what to monitor) and data tables (store results)
- RMON can
  - collect stats about protocols (e.g. pkt/byte/error count) and record history (e.g. buckets of intervals)
  - filter and capture packets seen on the medium
  - summarise traffic, by host/protocol, top talkers, traffic matrices
  - sample MIB objects and trigger event according to value

Things to think about

- Critical thinking:
  - What statistical techniques might be useful in monitoring network traffic? e.g. sample traffic rather than account for every packet?
- Engineering methods:
  - Hysteresis is widely used in engineering to reduce switching rates (as in RMON alarm processing)
- Links to other areas:
  - Compare RMON to local monitoring using Wireshark
  - RMON is essentially just another MIB of objects
  - Flows group packets (RMON is equally affected by all individual packets) & we'll study flow analysis later [K9>
- Independent learning:
  - "Switch Monitoring" is described by RFC 2613