Network Monitoring Using Traffic Dispersion Graphs (TDGs)

Marios Iliofotou

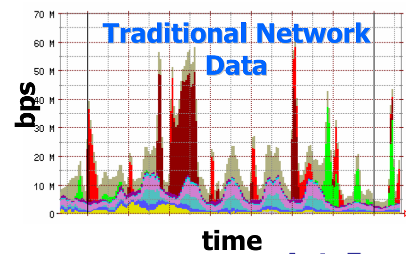
Joint work with:

Prashanth Pappu (Cisco), Michalis Faloutsos (UCR), M. Mitzenmacher (Harvard), Sumeet Singh(Cisco) and George Varghese (UCSD)

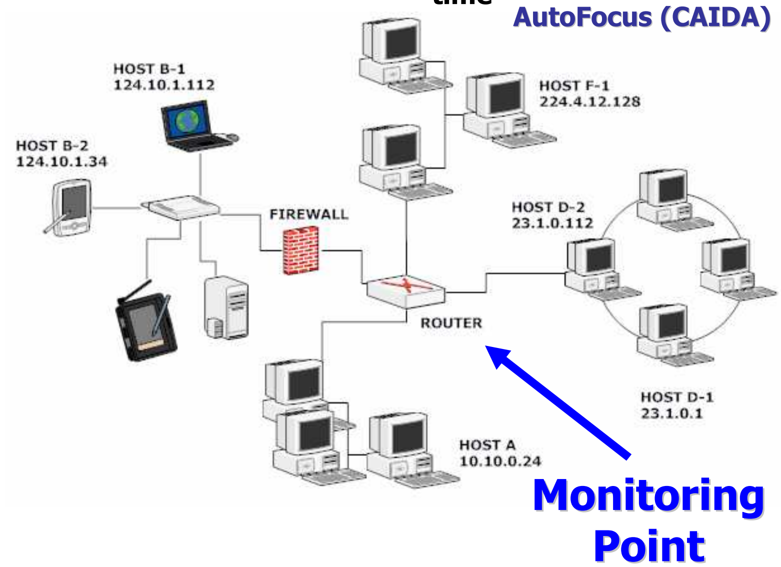
UC Riverside, Computer Science and Engineering Department

Introduction

- Task of Network Monitoring/Management: know your network.
  - Detect attacks, spot abnormalities
  - Get informed about changes in traffic trends
  - Adjust bandwidth allocation (rate limit or block flows etc.)
- Network traffic as seen at a router
  - 'Finer' granularity: **Packets**
    - Bytes/sec, pkts/sec, etc.
  - **Flows**, aggregating a set of packets
    - Flow records summaries (Cisco NetFlow)
      - Flows/second
      - Heavy Hitters (**Top 10 Flows**)
  - Individual **Hosts** that send packets
    - Top hosts in number of pkts, flows etc.
  - **Payload inspection** (Packet or Flow Level)
  - **New Dimension**: What we also see?
    - Set of interacting hosts (Graph) (who is talking to whom?)
      - Gives new source of information.

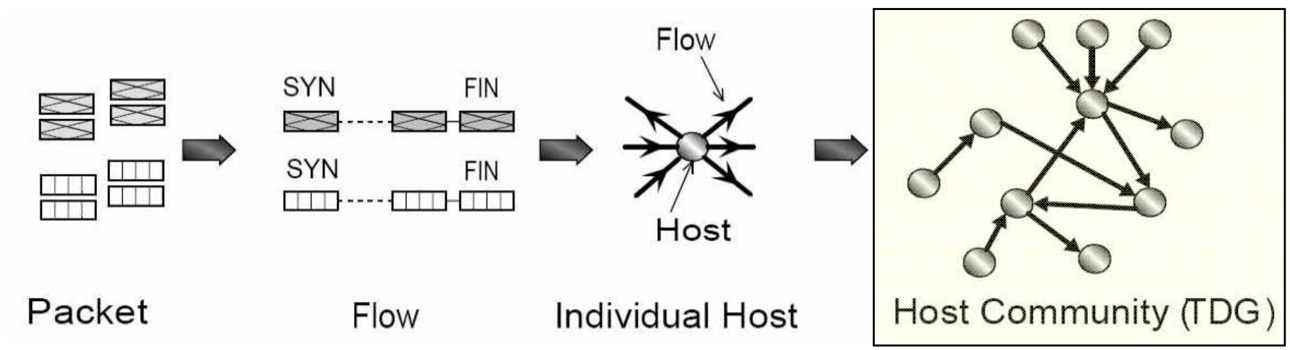


AutoFocus (CAIDA)



Introduction

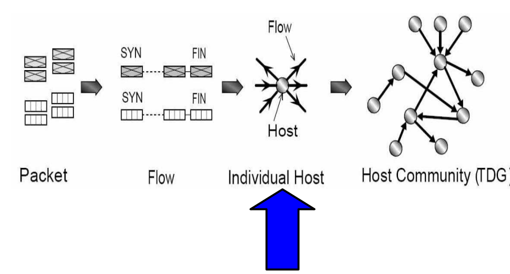
- From a Monitoring Level Perspective
  - Flows: aggregate a set of related packets
  - Hosts: aggregate a set of related flows (belong to the host)
  - TDGs: aggregate a set of related hosts



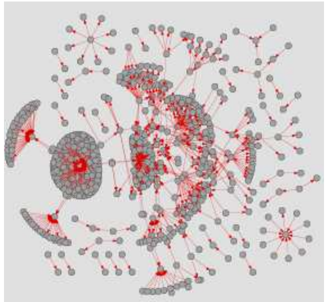
- Contribution: In this work, we propose TDGs as a way to
  - **Monitor**/**Analyze** and **Visualize** Network Traffic

Related Work

- Related work using host interactions
  - Ellis et al. in ACM WORM 2004. Try to detect the tree-like structure of a self propagating code (worm detection).
    - Complicated link predicates (worm spread signature)
    - Spread of communication, depth, fan-out
    - (*) Only on worm detection, enterprise networks
  - Xie et al. in ICNP 2006. Internet Forensic Analysis.
    - Backward random walk
    - Post-mortem analysis → identify patient zero (origin of the attack)
  - Aiello et al. in PAM 2005.Communities of Interest in Data Networks.
    - Grouping of hosts based on their interaction patterns
      - Popularity and Frequency
  - Karagianis et al. in ACM SIGCOMM 2006. **BLINC**.
    - Operates at the Host aggregation Level
      - Profile the users, and subsequently classify their flows
      - E.g., a host with many longed lived connections that carry large amount of data and uses different ports for each flow is labeled as p2p



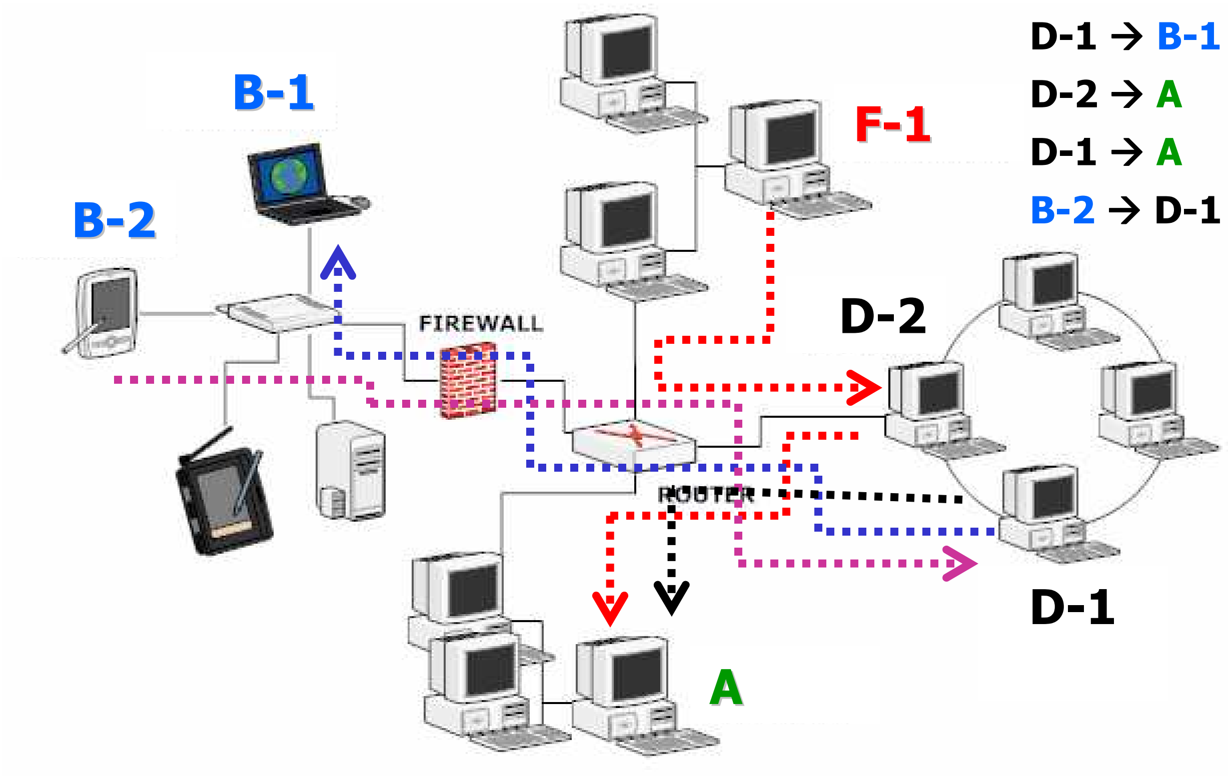
Outline



- *Introduction*
- *Related Work*
- **Defining TDGs**
- Exploration using TDG Visualizations
- Quantifying TDGs using graph metrics
  - Translate visual intuition into quantitative measures
- Future Work and Conclusions

Traffic Dispersion Graphs (TDGs)

- Example of a TDG Formation Process



The Links:

F-1 → D-2

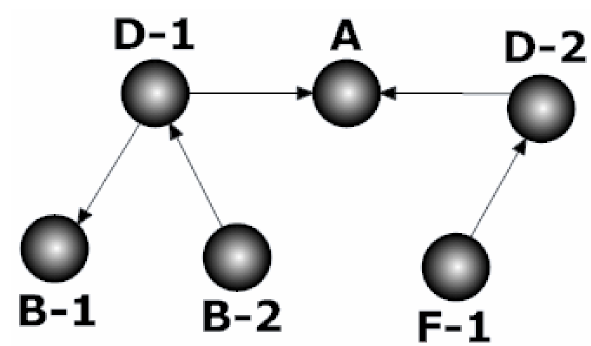
D-1 → B-1

D-2 → A

D-1 → A

B-2 → D-1

Generated TDG



Questions that arise:

- **When do we add a link?**
- **How long do we monitor?**
- **How do we characterize a TDG?**

Generating a TDG

- What are the steps for generating a TDG?
  1. Select a monitoring point (e.g., central router, backbone link)
  2. Select an "**edge filter**". Very important operation!
     - Edge Filter = "What constitutes an edge in the graph?"
     - E.g., TCP SYN Dst. Port 80
  3. For a packet that satisfies the edge filter, derive the **link**
     - srcIP → dstIP
  4. Collect the set of produced links within a **time interval**
     - E.g., 300 seconds (5 minutes)
  5. Gather all the links and generate a Graph.
     - This is the TDG for the particular "**edge filter**" and **observation interval** selected
- Observation: TDGs are formed by the **online addition of links**
  - Dynamic Graphs
- Why do we use edge filters?
  - Try to isolate specific communities of interacting hosts (filter out "noise")
    - E.g., a part of a peer-to-peer overlay (**filter-out** everything else)
  - Ask questions (query) the network
    - E.g., how does the graph of all the nodes that send packets having the payload signature "BitTorrent" looks like?

Edge Filtering Operation

- We can have many TDGs depending on the “edge filter”
  - Examples of Edge Filters:
    - a) number of pkts/bytes exchanged
    - b) any combination of L3 and L4 header features
      - TCP with SYN flag set and dst port 25
    - c) sequence of packets (e.g., TCP 3-way handshake)
    - d) Payload properties DPI
      - e.g., use as edges all the packets that match a particular **content signature**
- In this work we focus on studying **port-based TDGs**
  - **UDP** ports we generate an edge based on the first matching packet
    - e.g., on UDP packet with destination port 53 to get the “DNS TDG”
  - **TCP** we add a directed edge on a TCP SYN packet for the corresponding destination port number (thus, we know the initiator)
    - e.g., port 80 for the HTTP TDG, port 25 for SMTP TDG etc.

Experiments

- We will show that even these simple edge filters work
  - They can isolate various communities of nodes
    - Specific interactions corresponding to known application
      - Those applications that operate on the monitored port (e.g., port 53 → DNS)
- We conducted experiments using various real traffic traces
    - Typical duration = 1 hour
  - **OC48** from CAIDA (22 million flows, 3.5 million IPs)
  - **Abilene** Backbone for NLANR (23.5 million flows, 6 million IPs)
  - **WIDE** Backbone (5 million flows, 1 million IPs)
  - Access links traces (University of Auckland) + UCR traces were studied but not shown here (future work)

Outline

- *Introduction*
- *Related Work*
- *Defining TDGs*
- **Exploration using TDG Visualizations**
- Quantifying TDGs using graph metrics
  - Degree Distribution, Component Sizes, etc.
- Future Work and Conclusions

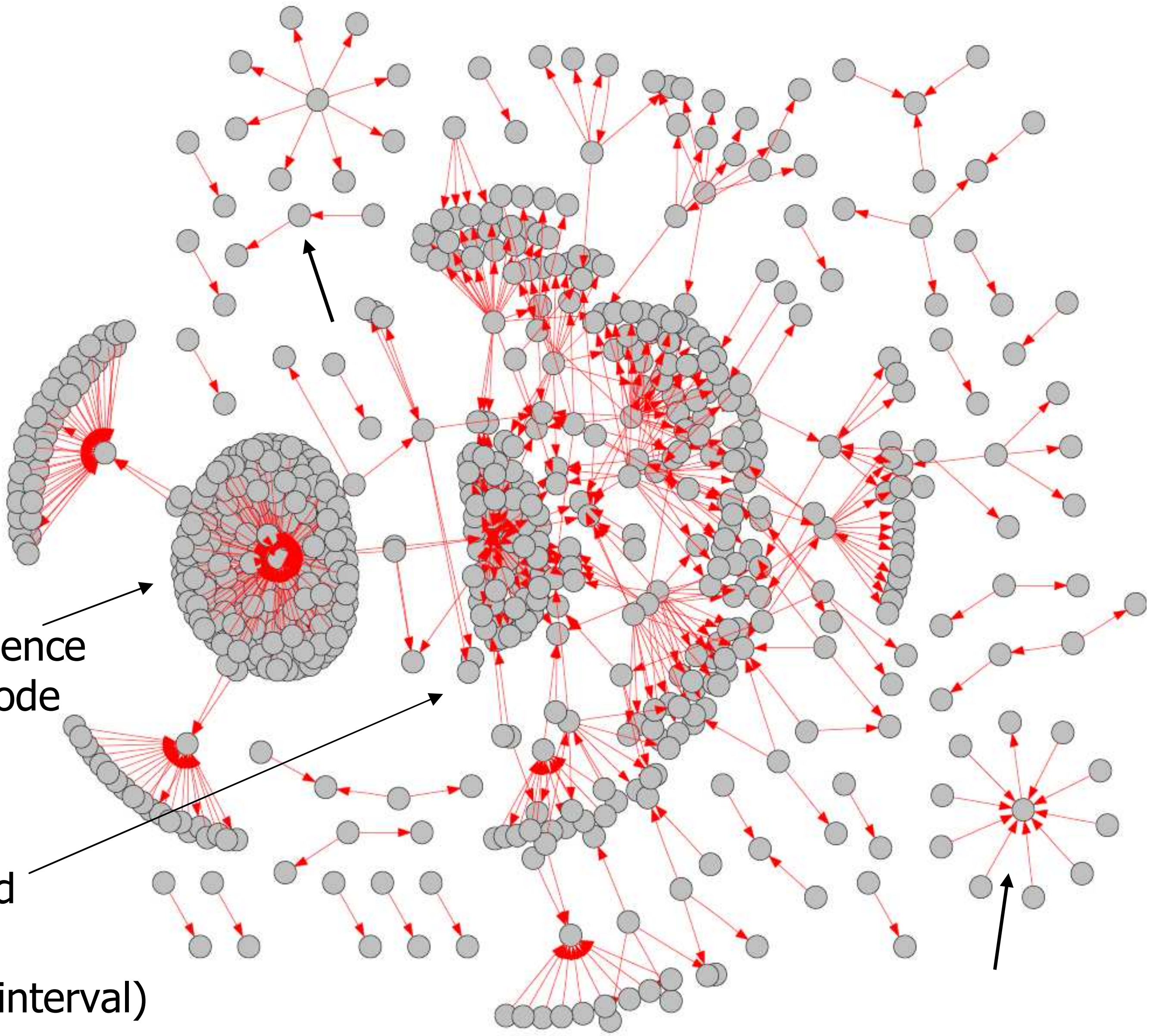
TDG Visualization (DNS)

DNS TDG
- UDP Dst. Port 53
- 5 seconds

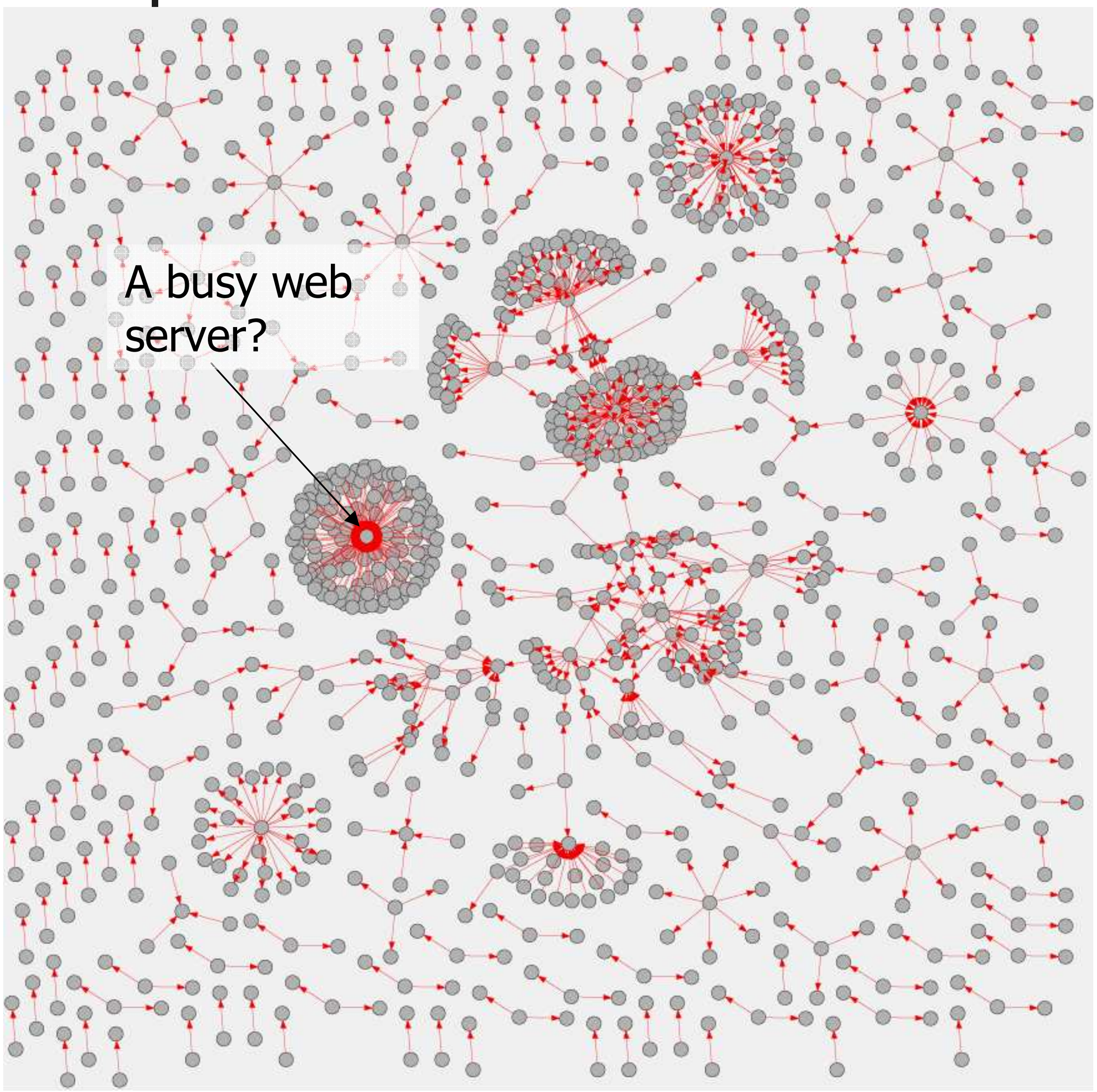
In- and Out-degree nodes

Very common in DNS, presence of few very high degree node

One large Connected Component!
(even in such small interval)



TDG Visualization (HTTP)



HTTP TDG

- **TCP SYN Dst. Port 80**
- **30 seconds**

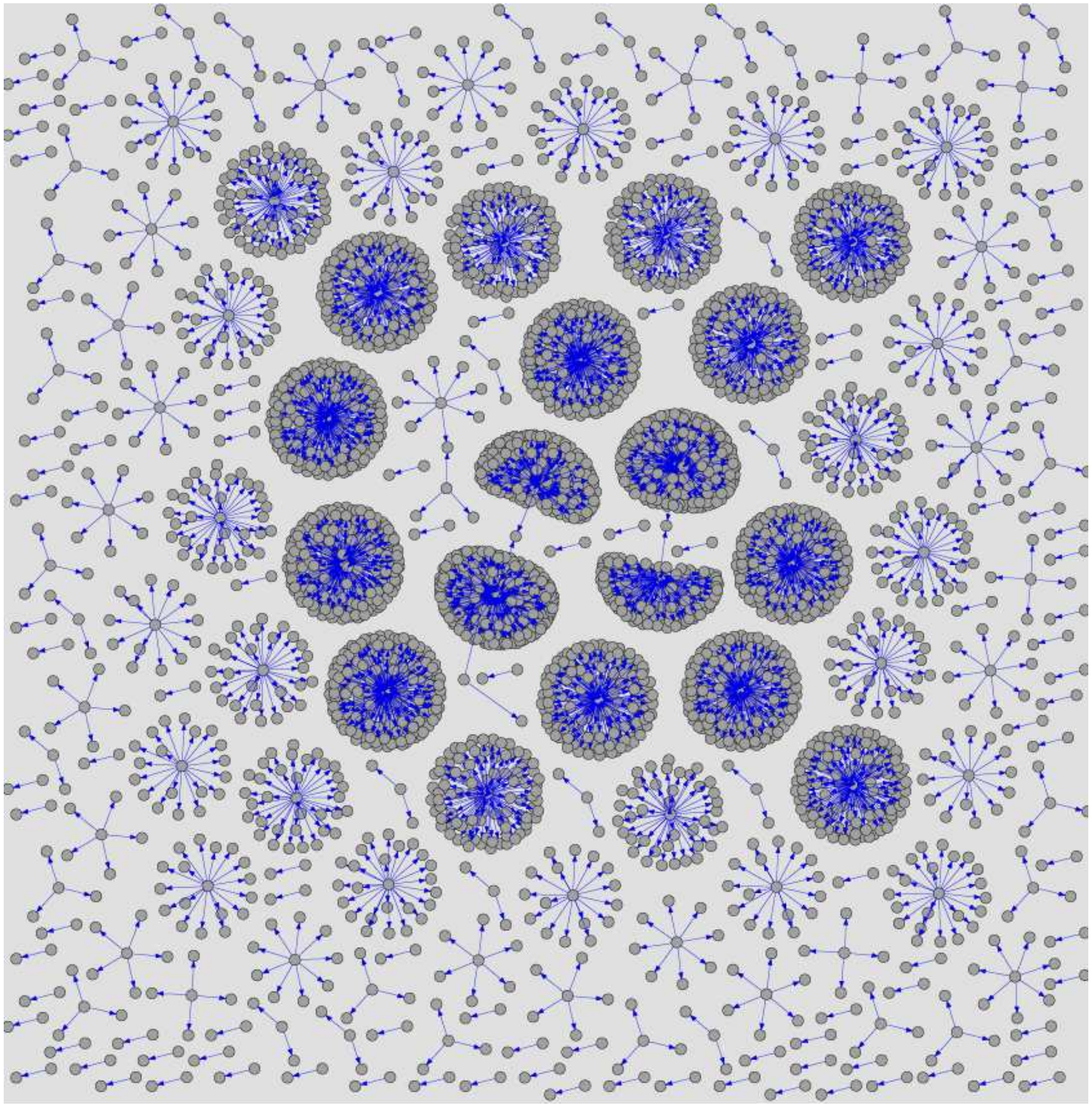
Observations

- There is not a large connected component as in DNS
- Clear roles
  - very few nodes with in-and-out degrees)
    - Web proxies?
- Many disconnected components

TDG Visualization (Slammer Worm)

Slammer Worm
- UDP Dst. port 1434
- 10 seconds

- About:
  - Jan 25, 2003.
    MS-SQL-Server 2000 exploit
    - Trace: April 24th
- Observations
  (Scanning Activity)
  - Many high out-degree nodes
  - Many disconnected components
  - The majority of nodes have **only in-degree**
    - Nodes being scanned



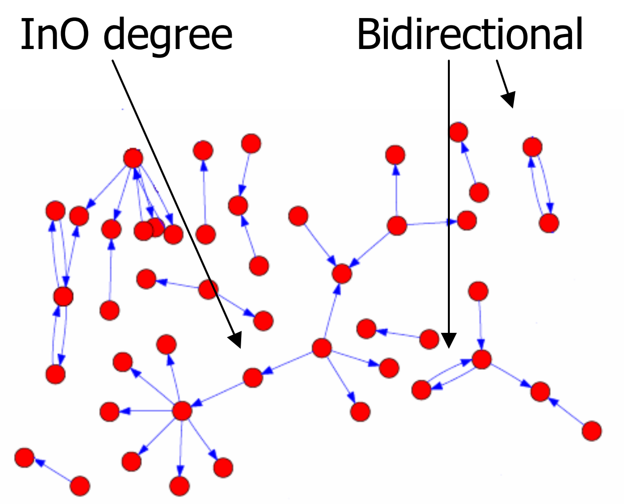
TDG Visualizations (Peer-to-Peer)

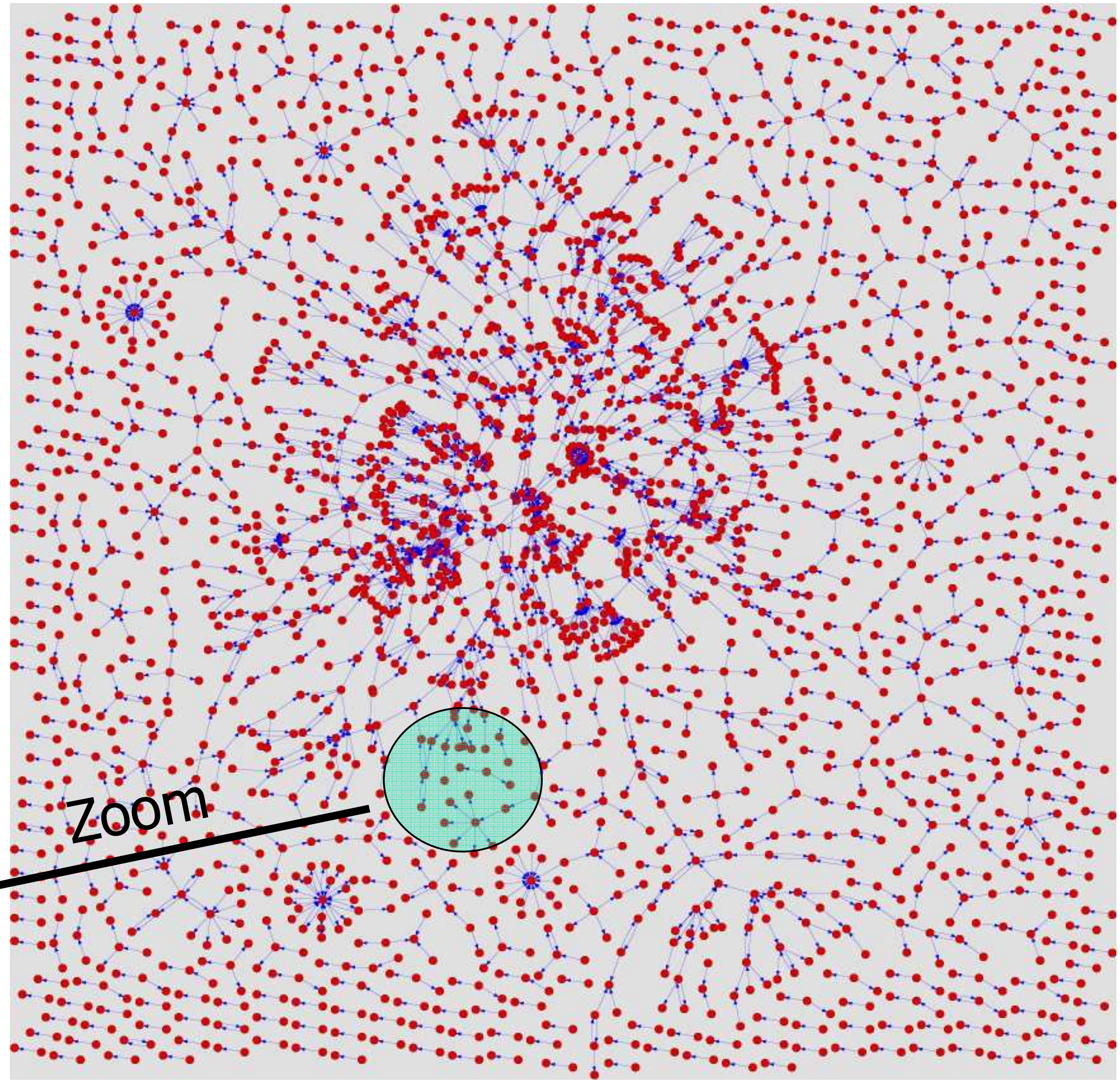
WinMX P2P App

- UDP Dst. Port 6257
- 15 sec

Observations

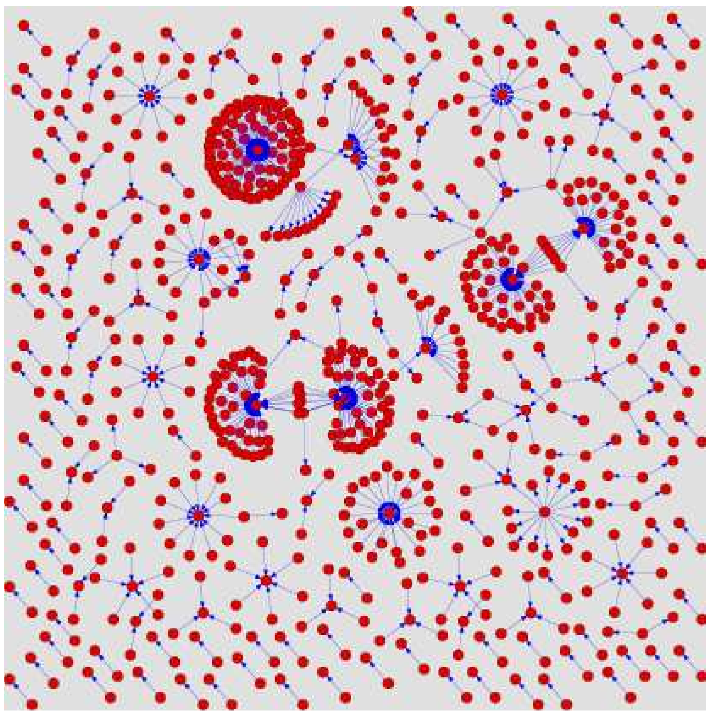
- Many nodes with in-and-out degree **(InO)**
- One large connected component
- Long chains



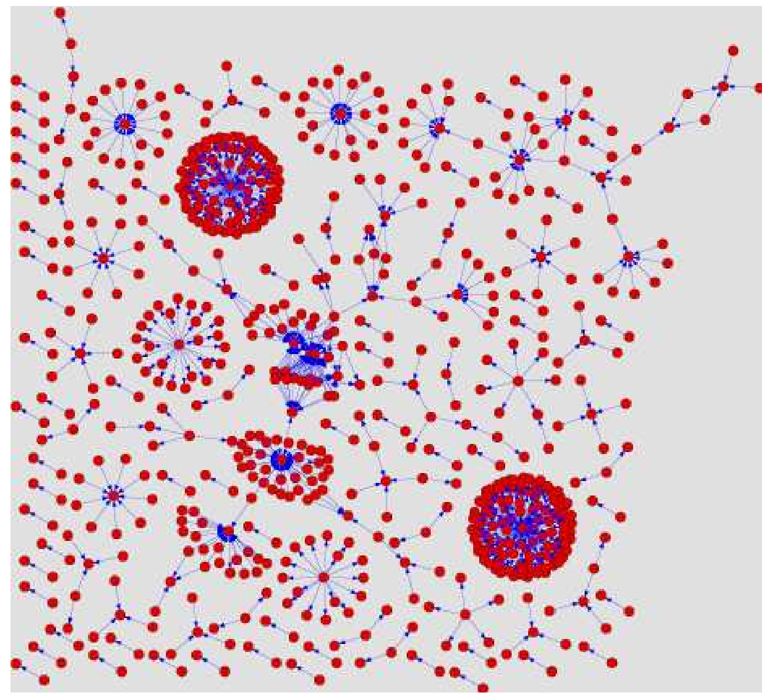


How can we Visualize compare TDGs?

Web: https



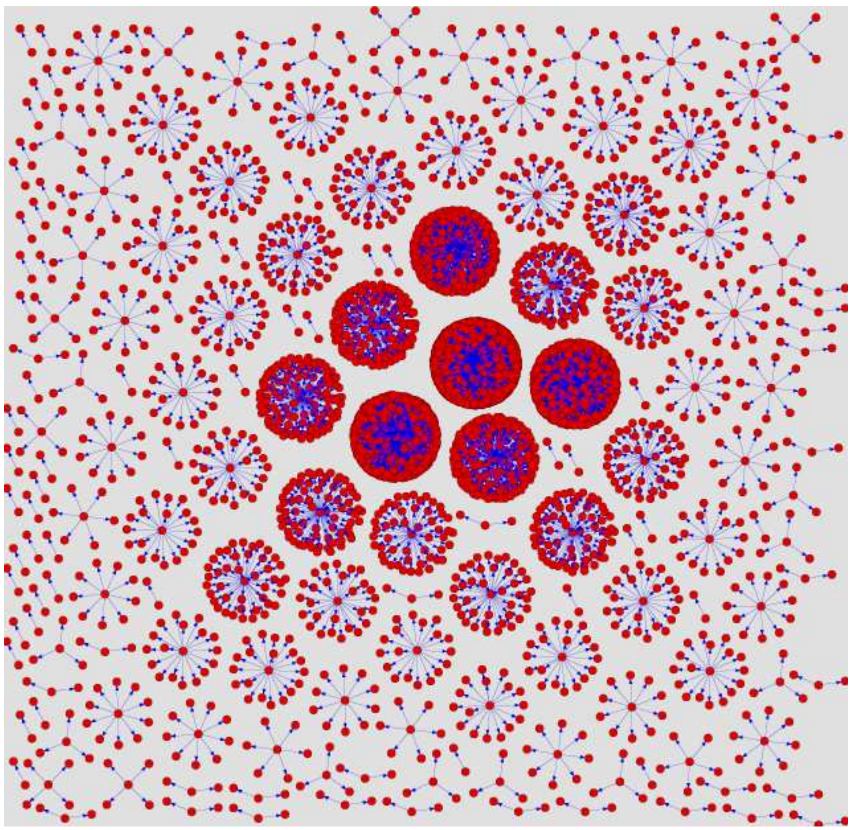
Web: port 8080



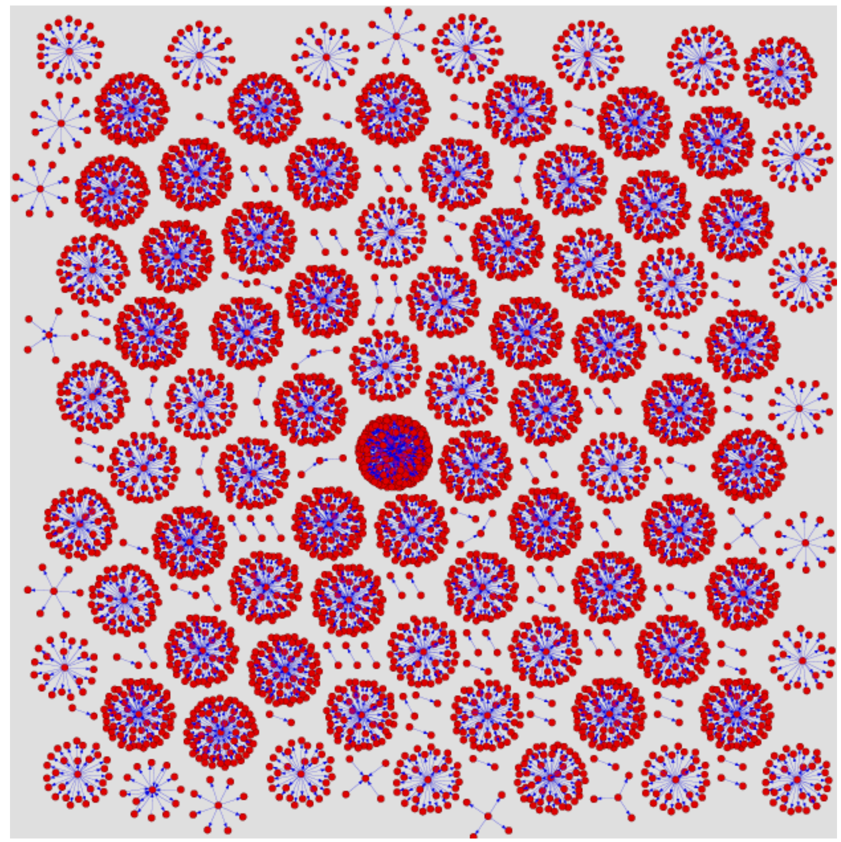
How can we Visualize compare TDGs?

Random IP range scanning activity?

Slammer: port 1434

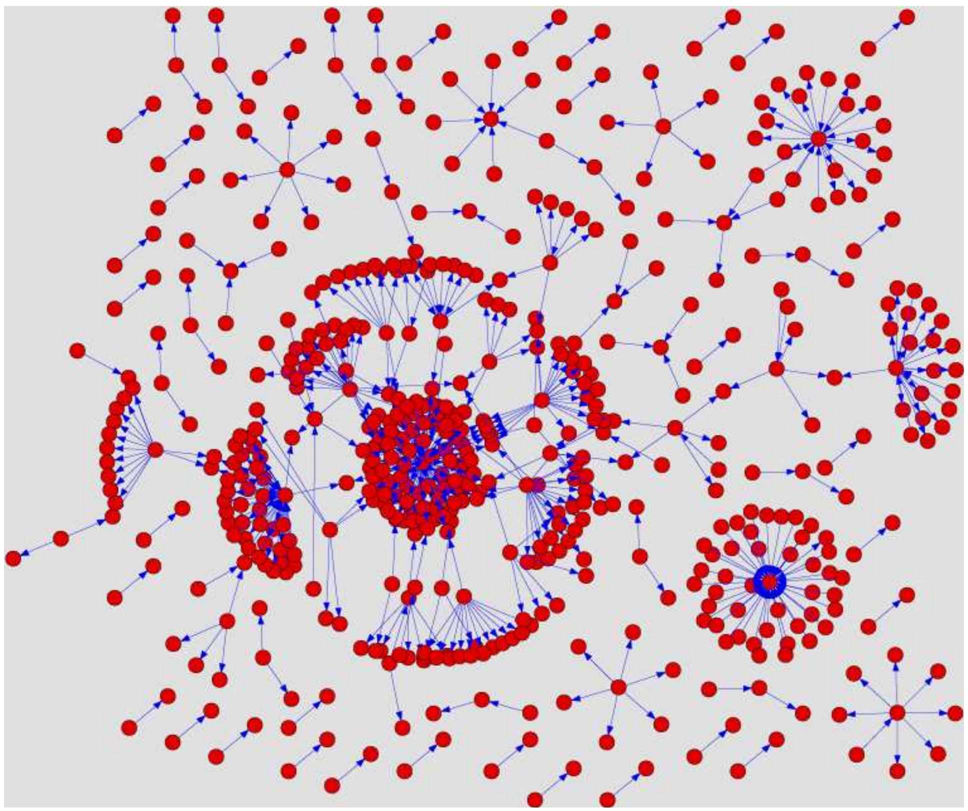


NetBIOS: port 137

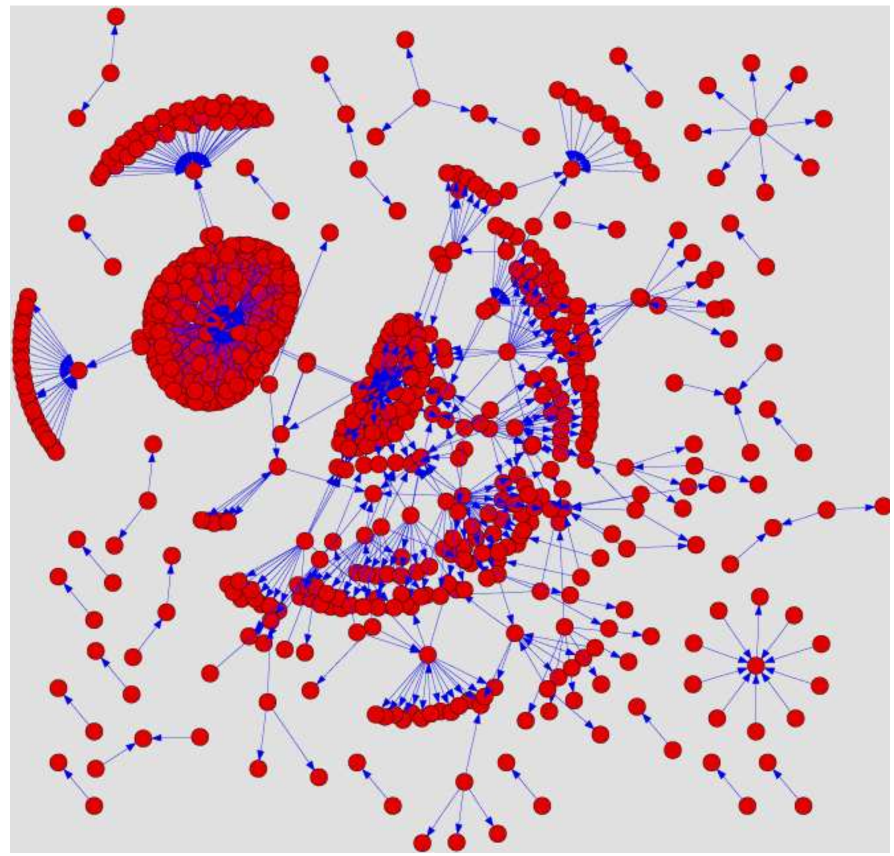


How can we Visualize compare TDGs?

SMTP (email)

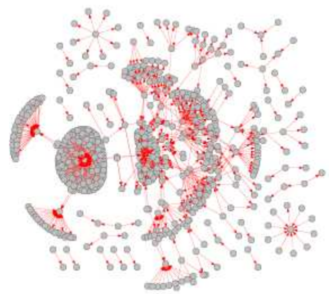


DNS



- Today none of the current monitoring tools provide this dimension of traffic monitoring

Outline



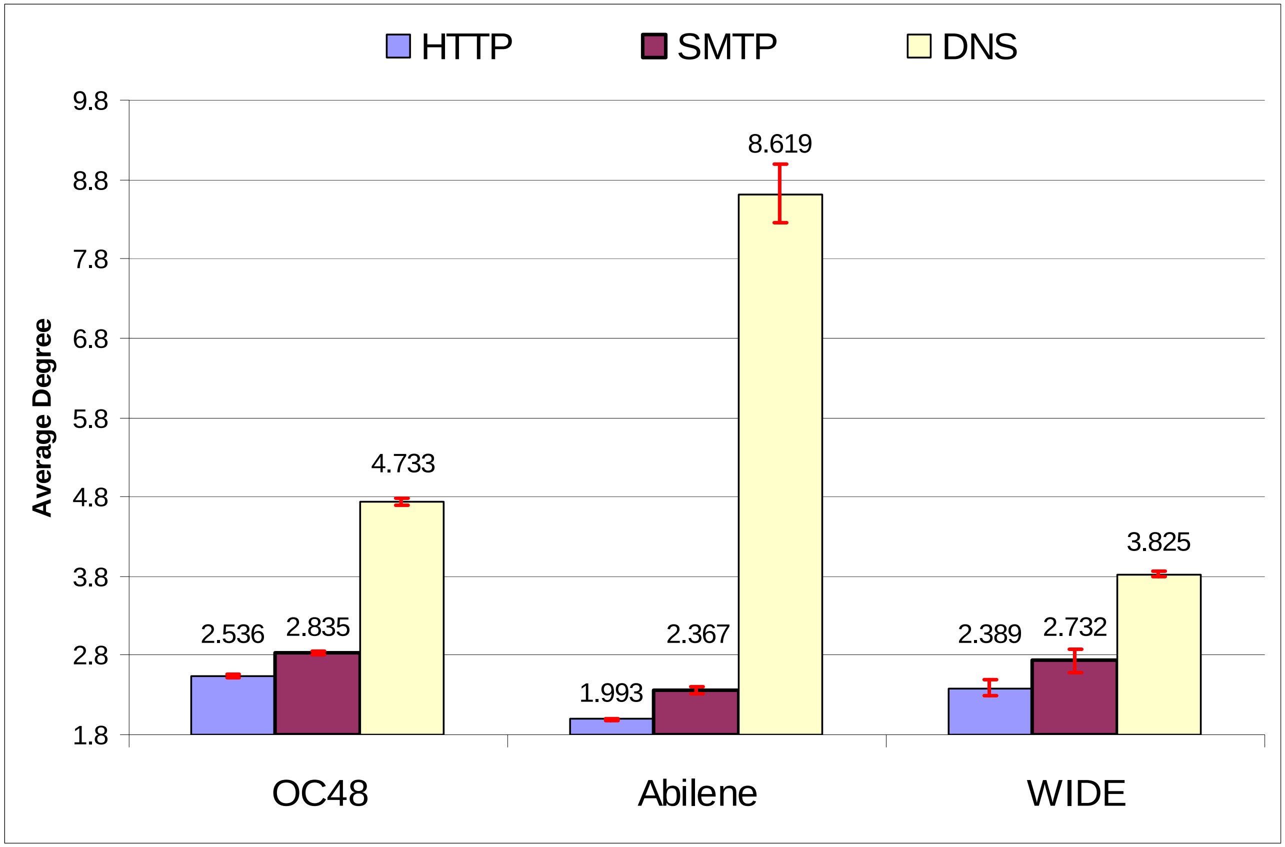
- *Introduction*
- *Related Work*
- *Defining TDGs*
- *Exploration using TDG Visualizations*
- **Quantifying TDGs using Graph Metrics**
- Future Work and Conclusions

Graph Metrics on TDGs

- What we have seen so far: "Visualization is useful by itself"
  - However, it requires a human operator
- Next Step?
  - It is important to translate **visual intuition** **into** **quantitative measures**
- To achieve this, we use a series of graph metrics
  - Goal: Quantitatively characterize TDG properties.
    - Average Degree, degree distribution, component size distribution etc.
- For evaluating and testing our metrics we used real network traffic traces
  - Backbone (OC48 @ CAIDA, WIDE Backbone, Abilene Network)
  - All traces are 1 hour long and monitor millions of hosts.
- Methodology: Each TDG is generated within a **300 sec interval**
  - Presented values are averaged over the 12 disjoint 300 sec intervals of the 1h trace
  - Note: We can always choose to ignore directivity for metrics such as the
    - popularity (distinct IPs with which a node is connected)
    - component distribution etc.

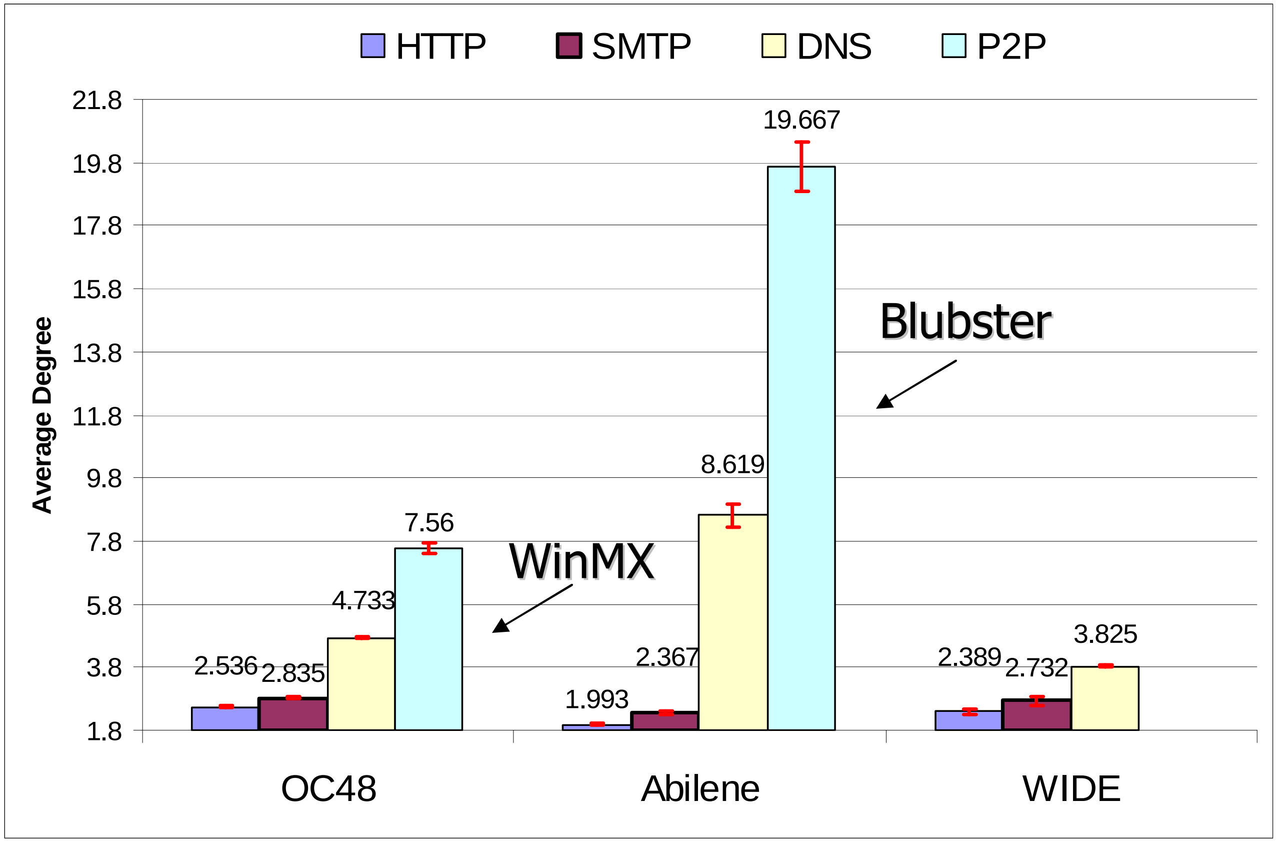
Graph Metrics on TDGs

- **Average Degree:** On average how many neighbors each node has.
  - High average degree in TDG usually indicates collaboration
    - e.g., p2p apps, online gaming overlays
- **Stability** of TDG metrics over time !
  - Small Std Div.



Graph Metrics on TDGs

- **Average Degree:** On average how many neighbors each node has.
  - High average degree in TDG usually indicates collaboration
    - e.g., p2p apps, online gaming overlays
- **Stability** of TDG metrics over time !
  - Small Std Div.



Graph Metrics on TDGs

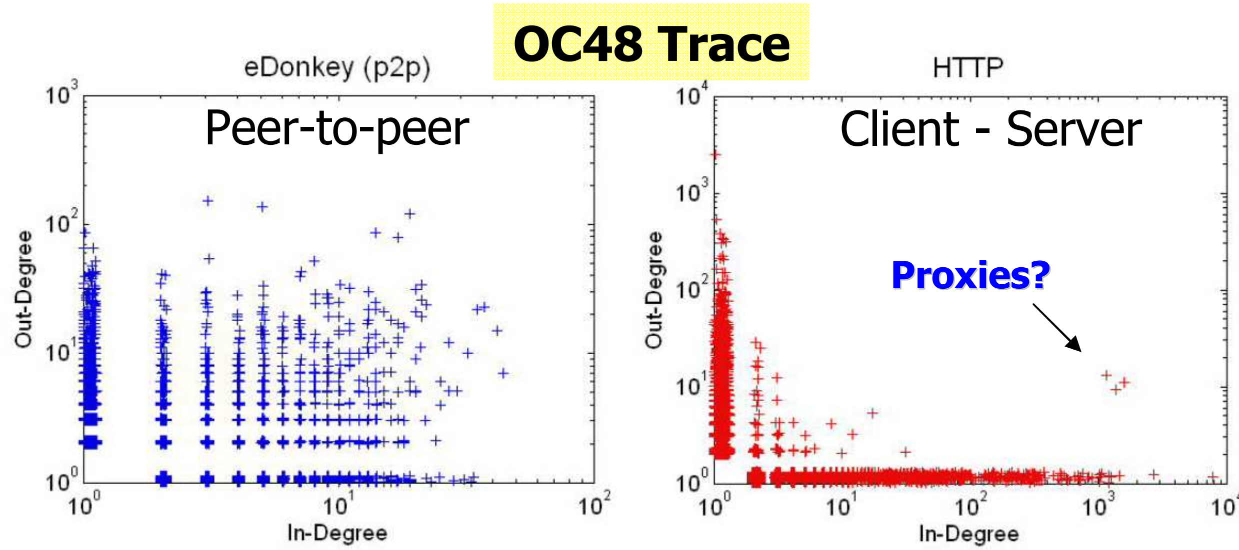
- **Directionality:** The percentage of nodes with only in-edges (sinks), only out-edges (sources) OR both in-and-out edges (**InO**).

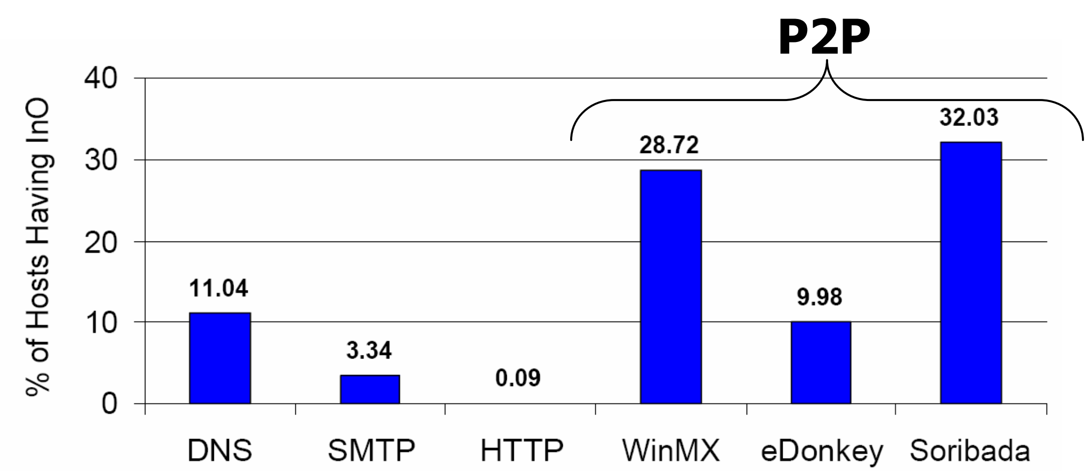
  Very useful metric:

  - Collaborating communities have
    - **High InO** → Act both as clients and server
  - Heavy scanning activity
    - **High % of only-in-edges (?)**
      - IPs being scanned
  - Client server TDGs have
    - very low InO, and
    - balanced percentages
      - only in-degree (~%20) **OR**
      - only out-degree nodes (~80%)
        - Usually we have more clients than servers.

Example (InO)

- eDonkey Vs HTTP
  - TCP SYN pkts are used in both
- The presence of nodes with both in- and out- degrees (InO).
  - Can be used to discriminate between p2p and client-server application.
- If the % of nodes with InO increases for some port, it can be used as an indication that a p2p app is tunneling traffic under that port.



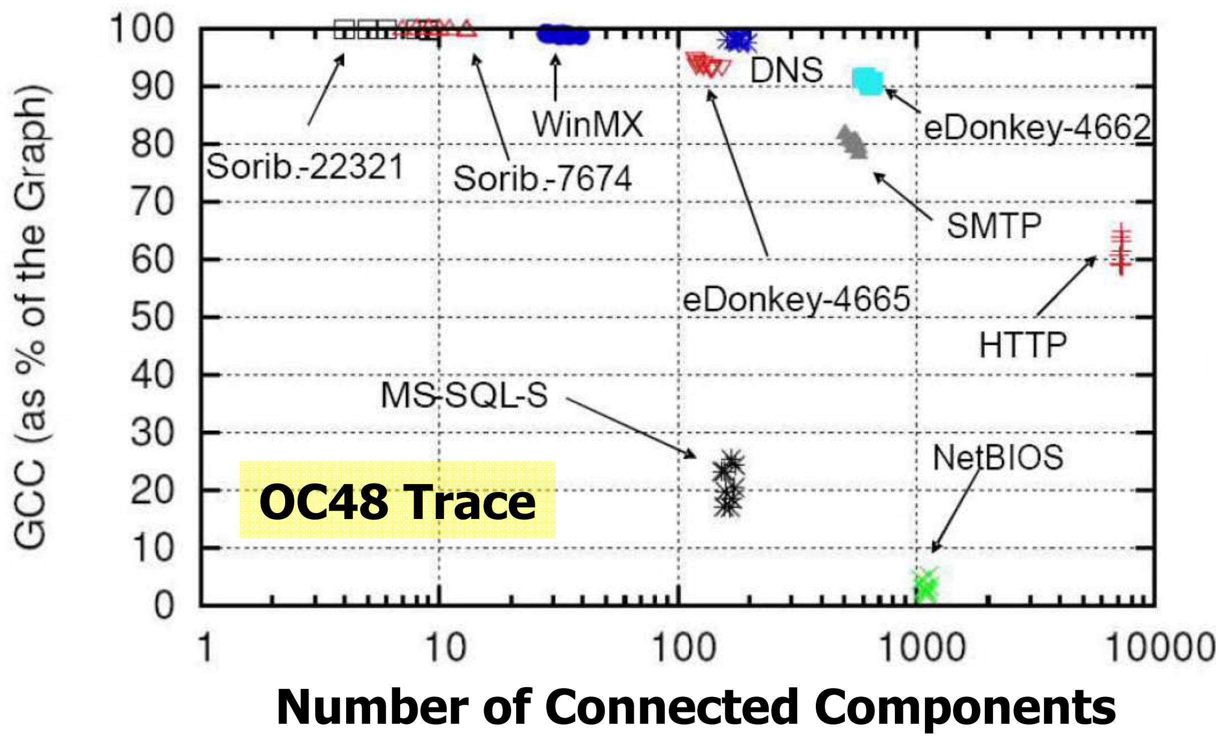


Component Size Distribution

- In general TDGs can be disconnected graphs
  - There is **no** (undirected) path between every pair on nodes in the graph
- The component size distribution captures the % of node that belong to a particular component size
- **Giant Connected Component (GCC):** Is the size of the largest connected subgraph in a TDG, measured as the % of nodes belonging to that component.
  - Collaborative communities are found to have one large GCC
- The size of GCC & total # of disconnect components
  - Has stability over time
  - Captures intrinsic characteristics of the underlining application

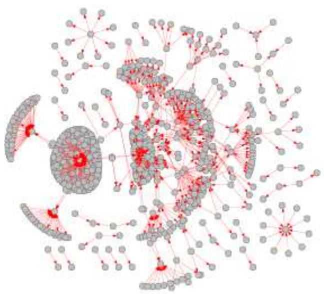
Monitoring Example (Gcc)

- Monitor the top 10 ports number in number of flows.
- Scatter Plot:
  - GCC Vs number of connected components.
  - **Stability over Time!**
- Peer-to-peer
  - large GCC > 90%
- Ms-sql-s, NetBIOS
  - Suspicious activity
  - Many disconnected
  - Small GCC (we would have a large GCC if there was one large scanner)



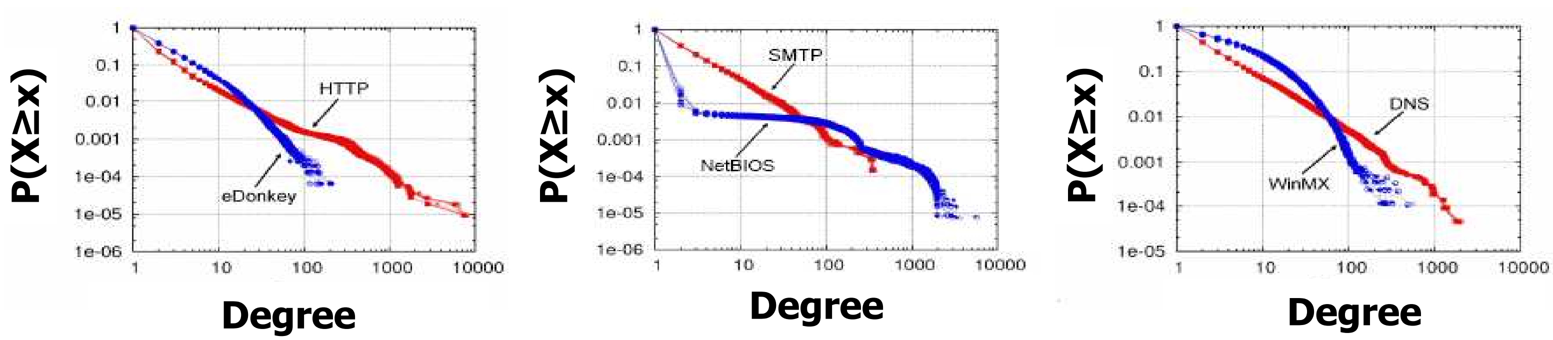
- Soribada
  - UDP port 22321
  - UDP port 7674
- WinMX
  - UDP port 6257
- eDonkey
  - TCP port 4662
  - UDP port 4665
- NetBIOS
  - UDP port 137
- MS-SQL-S
  - TCP por 1433

Outline



- *Introduction*
- *Related Work*
- *Defining TDGs*
- *Exploration using TDG Visualizations*
- **Quantifying TDGs using Graph Metrics**
  - *Scalar Metrics*
  - **Non-scalar Metrics**
- Future Work and Conclusions

Degree Distribution



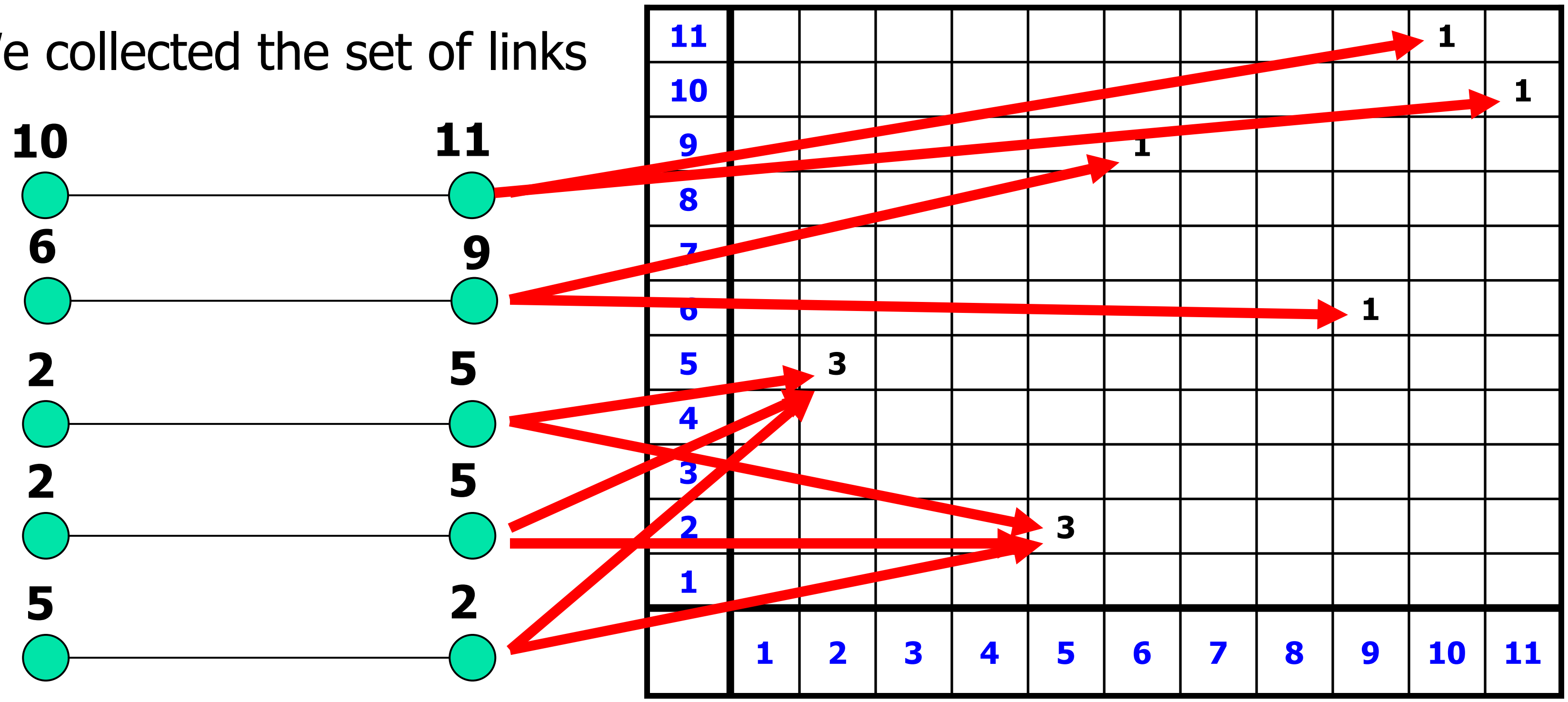
- The degree distributions show heavy tailed behavior
  - Some distribution can be closely modeled with **power-laws** **(HTTP, DNS)**.
- P2P communities tend to have many medium degree nodes (degree 4 to 30).
  - HTTP and DNS have few nodes with very high degrees.
  - High variability (stdv/avg): HTTP=16, DNS=6. WinMX=1.6, eDonkey=1.8.
- **NetBIOS:**
  - Scanning activity !! 98% of nodes have degree of one,

    few nodes with very high degree → scanners

Joint Degree Distribution (JDD)

Note: Not all the links of the graph are shown!

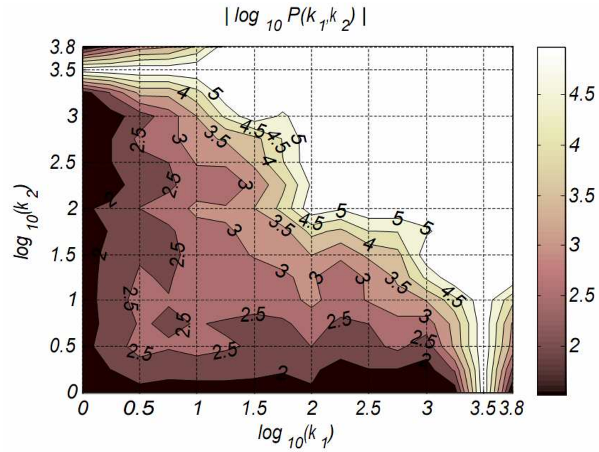
- We collected the set of links



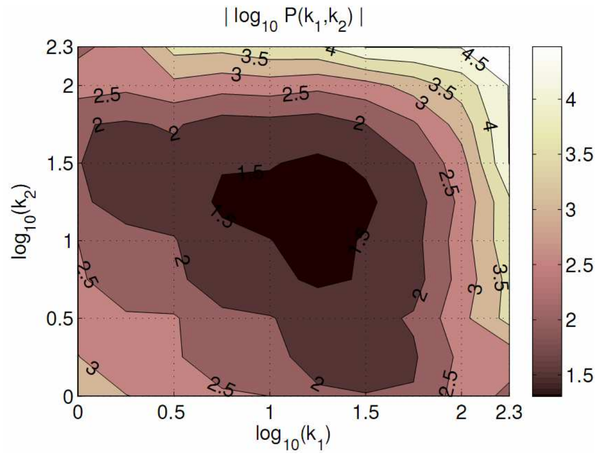
- The matrix is Symmetric
- **$P(k_1,k_2)$**, probability that a randomly selected edge connects nodes of degrees k_1 and k_2
  - Normalized by the total Number of links

Joint Degree Distribution (JDD)

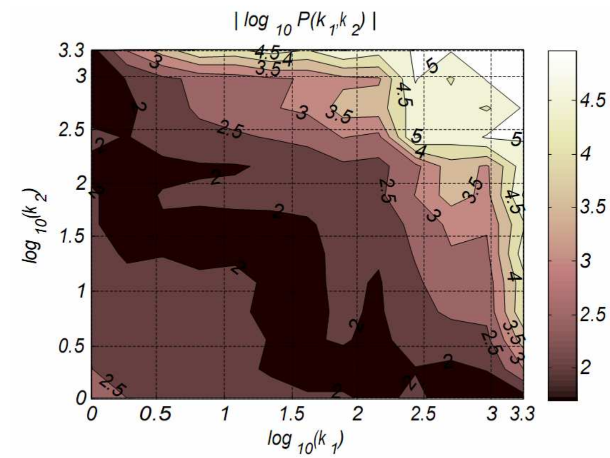
HTTP (client-server)



WinMX (peer-to-peer)

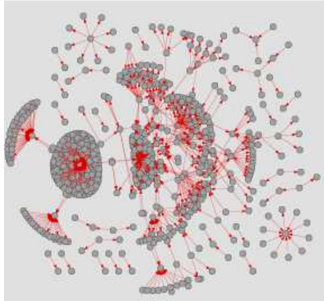


DNS (c-s and p2p)



- Contour plots
  - x-axis: Degree of the node on the one end of the link (logarithmic scale due to high variability)
  - y-axis: Degree of the other node (logarithmic scale due to high variability)
- Observations:
  - HTTP: low degree client to low to high degree servers
    - One end of the link has low degree and the other has low-to-high
  - WinMX: medium degree nodes are connected
  - DNS: sings of both client server and peer-to-peer behavior
- Top degree nodes are not directly connected
  - White regions at the top right corner

Outline



- *Introduction*
- *Related Work*
- *Defining TDGs*
- *Exploration using TDG Visualizations*
- *Quantifying TDGs using graph metrics*
- **Conclusions**

Conclusions

- New way of looking at traffic, that offers:
  - Nice visualization that can enhance intuition
    - We only used general graph visualizations (GraphViz)
      - More application specific tools could be developed
  - Graphs that have information
    - Can be used to describe the interaction of the captured node
      - P2P, client-server, scanning activity?
  - Stability over time
    - It can be used to trigger alarms
    - Potentially, we can derive thresholds to classify TDGs

Future Directions

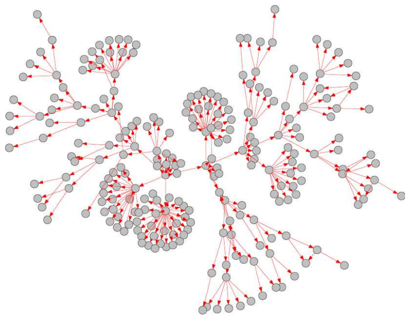
- Develop of a s/w Monitoring Tool, which uses TDGs
- From TDGs can we reveal underline application?
  - Which are the best metrics?
  - Which are the thresholds for this metrics?
- How are TDG features change over time.
  - E.g., within 24 period.
  - A week?
  - Months
    - 107 days (WIDE Backbone trace)
  - Years (historical traces, WIDE Backbone 7 years of trace collection)
    - Can we capture features of the evolution of applications
- Effect of the observation point
  - Backbone **Vs** Assess Link **Vs** Enterprise central router

TDG - Publication

- **“Network Monitoring Using Traffic Dispersion Graphs”**
  - Marios Iliofotou, P. Pappu, M. Faloutsos, M. Mitzenmacher, S. Singh, G. Varghese
  - Internet Measurement Conference (IMC 2007)

Thank You!

Questions/Discussion



Additional Monitoring Example

- Honeypot trace in a LAN (@ UCSD)
  - Blaster Worm spread emulation
- Observations:
  - Tree-like structure (Ellis et al.)
  - High Depth
    - Max = 8
    - Avg = 4.4
  - InO = 21% !

Spread of Blaster Worm