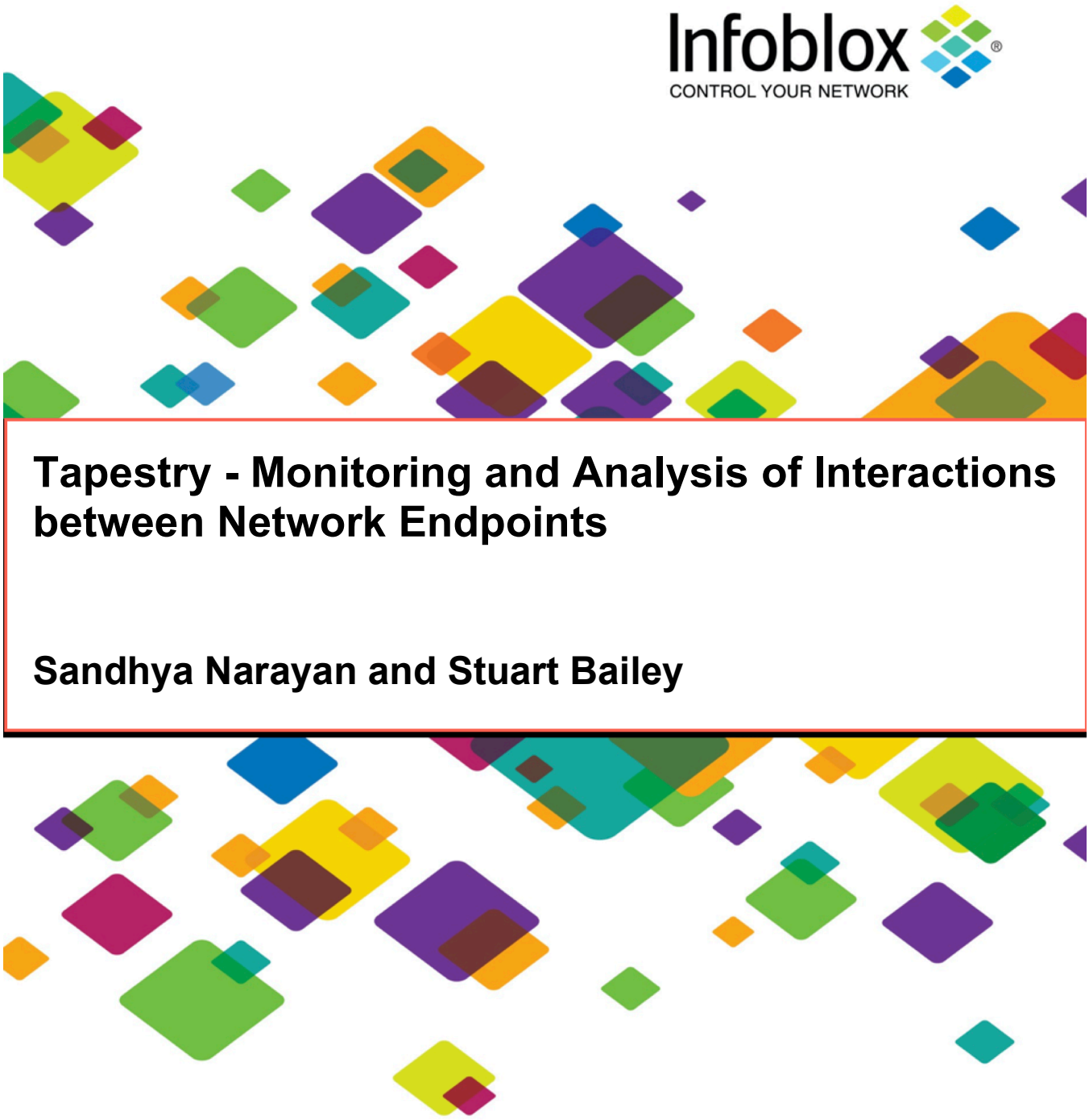


# Tapestry - Monitoring and Analysis of Interactions between Network Endpoints

**Sandhya Narayan and Stuart Bailey**

## ABSTRACT

Tapestry is an innovative network application that gathers data about endpoint-interactions from network-wide control systems such as DNS, analyzes them, computes an indicator called Network Complexity Index (NCI) and provides a new level of visibility into the network.

Whether an endpoint making use of an enterprise network is a BYOD device or a company-owned system, when that endpoint uses internal or external resources and applications, it first typically makes a DNS request for domain name resolution for the service or resource of interest. By tracking the use of DNS by endpoints it is possible to identify interactions between endpoints, no matter what application causes the interaction. Tapestry uses this endpoint-interaction data to show how an enterprise network is being used and quantify how complex it is, regardless of where the endpoints, applications, and services of that enterprise reside. Unlike products and solutions in the industry that focus on endpoint visibility, Tapestry gives its users a unique macro view by providing visibility into endpoint-interactions, showing which endpoints are interacting with each other endpoints, for example, addressing various use cases such as in network security and network management.


## INTRODUCTION

With the growth of BYOD, mobile devices and specialized external services used by enterprises, the corporate network and its partner extensions have become a crucial part of enterprises. Understanding and advocating the requirements and issues of this network is one of the primary tasks of a CIO. It is becoming clear that complexity rather than bandwidth is the barrier to network growth. However, traditional approaches to network planning focus on bandwidth, network infrastructure devices, and the interconnecting wires. In contrast, Tapestry offers a unique solution to the network operator. It focuses on network complexity, identifying relationships of business processes to an increasingly large, dynamic, and shared global IT infrastructure, and arrives at, for example, a single number called Network Complexity Index (NCI) such as described further below. By monitoring NCI over time, network operators can understand changes in the complexity of their network and plan resources accordingly.

Tapestry analyzes real-time endpoint interaction data and shows higher-level relationships between endpoints. Tapestry's unique graphical UI brings out new insights by combining individual endpoint information together with classification of endpoints into groups or communities based on the nature of interactivity between endpoints. Tapestry application is a completely new tool for a network operator that provides a unique approach and solution to identify complexity of interacting business processes and the underlying network that supports these processes. For example, observing endpoint-interaction behavior of related endpoints rather than just an endpoint's behavior alone provides Tapestry's users with a quick broad view of related activities in the network.

In an example implementation, Tapestry is a distributed online real-time, big data graph clustering and index computation application. A significant outgrowth of the Tapestry application is the Tapestry platform, an integrated network telemetry and analytics software platform that can be executed on commercial off-the-shelf (COTS) hardware, is easy to deploy, requiring very little change to existing infrastructure. In an example implementation, the Tapestry platform includes an integrated collection of components for data collection, aggregation, analytics, and visualization. The same software program can perform real-time analytics as well as data collection from the network as all the functions of data collection, aggregation and analytics can be embedded. This platform is a novel architecture, simple enough and providing full control to the application developer to seamlessly work on what is collected, how it is analyzed and displayed. The platform is flexible and can be used to support many applications around network telemetry, security, analytics and control based on this platform. This architecture is in contrast to the common practice of using separate systems such as Hadoop or Spark for analytics and

feed it data collected by a totally separate mechanism, requiring developers, admins and systems very different in nature for the two tasks.

## HOW TAPESTRY APPLICATION WORKS

For example, interactions between endpoints in a network can be examined and grouped into network-activities based on how closely some endpoints are interacting among themselves as compared to how sparsely the same group of endpoints interacts with others. A network's complexity can be quantified from the number of groups or activities and the size of those activities. In some cases, bigger tightly knit activities indicate a more complex network.

In an example implementation, positive DNS responses can serve as a simple but good indicator for interactions between endpoints. **Figure 1** shows a graph of the endpoint interactions where the endpoints are shown by dots and edges indicate interaction between two endpoints.

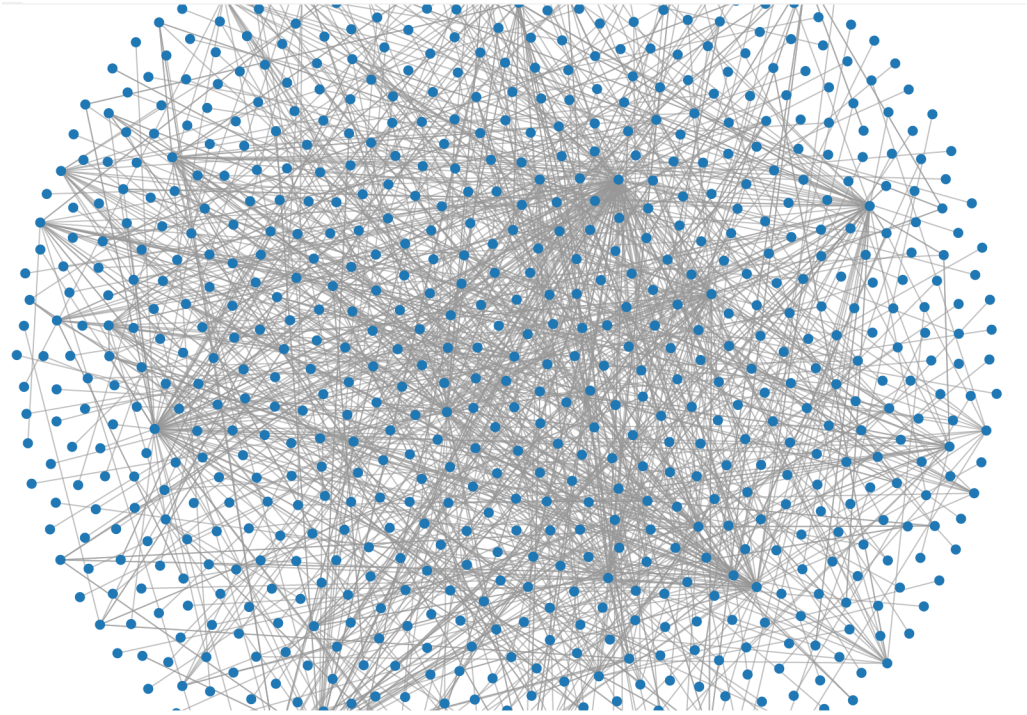

**Figure 1**

The "hairball" of the interactions between endpoints (Figure 1) can be analyzed using community detection algorithms to identify groups of endpoints that are interacting more closely among themselves and relatively sparsely with endpoints in other groups, the result being much like finding knots in a hair ball. **Figure 2** shows the same set of endpoint interactions, with the endpoints colored according to the activity they were classified into. Figure 3 is redrawn to show the different groups or "activities" in different colors. **Figure 4** shows the interactions between endpoints within one activity.

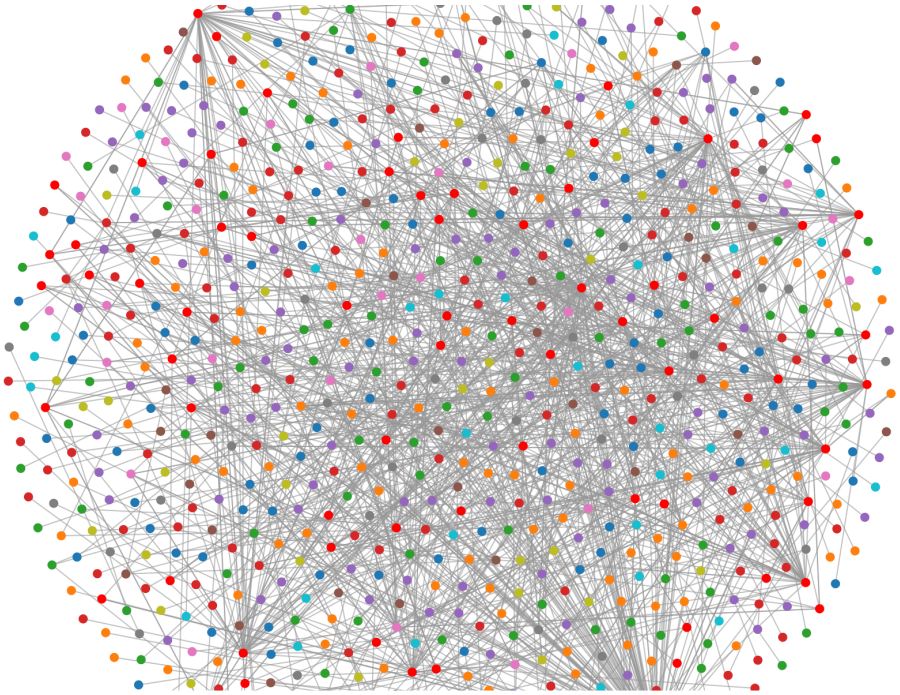

**Figure 2**

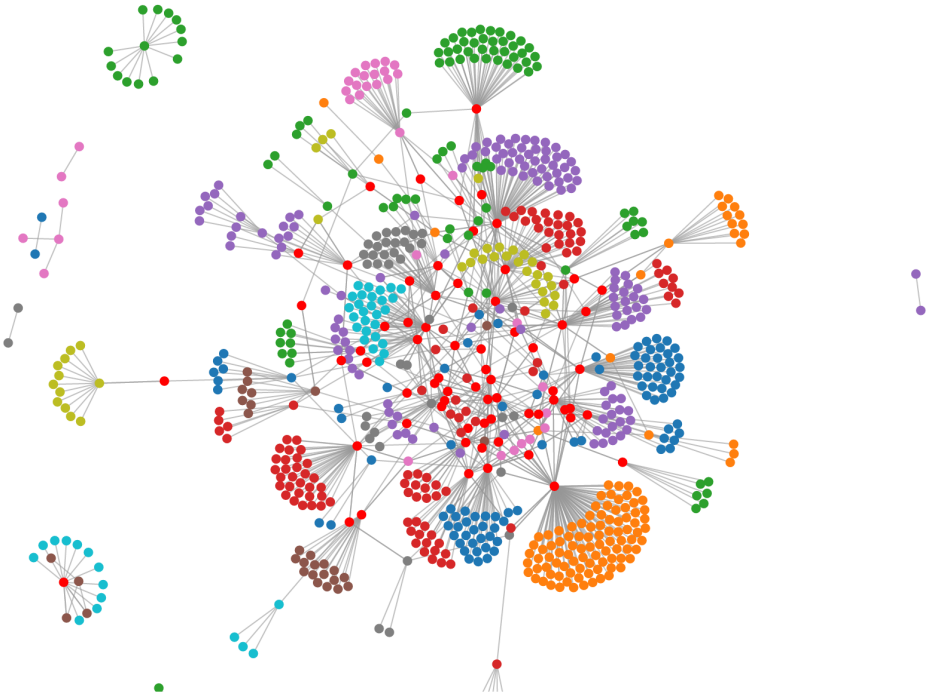

**Figure 3**

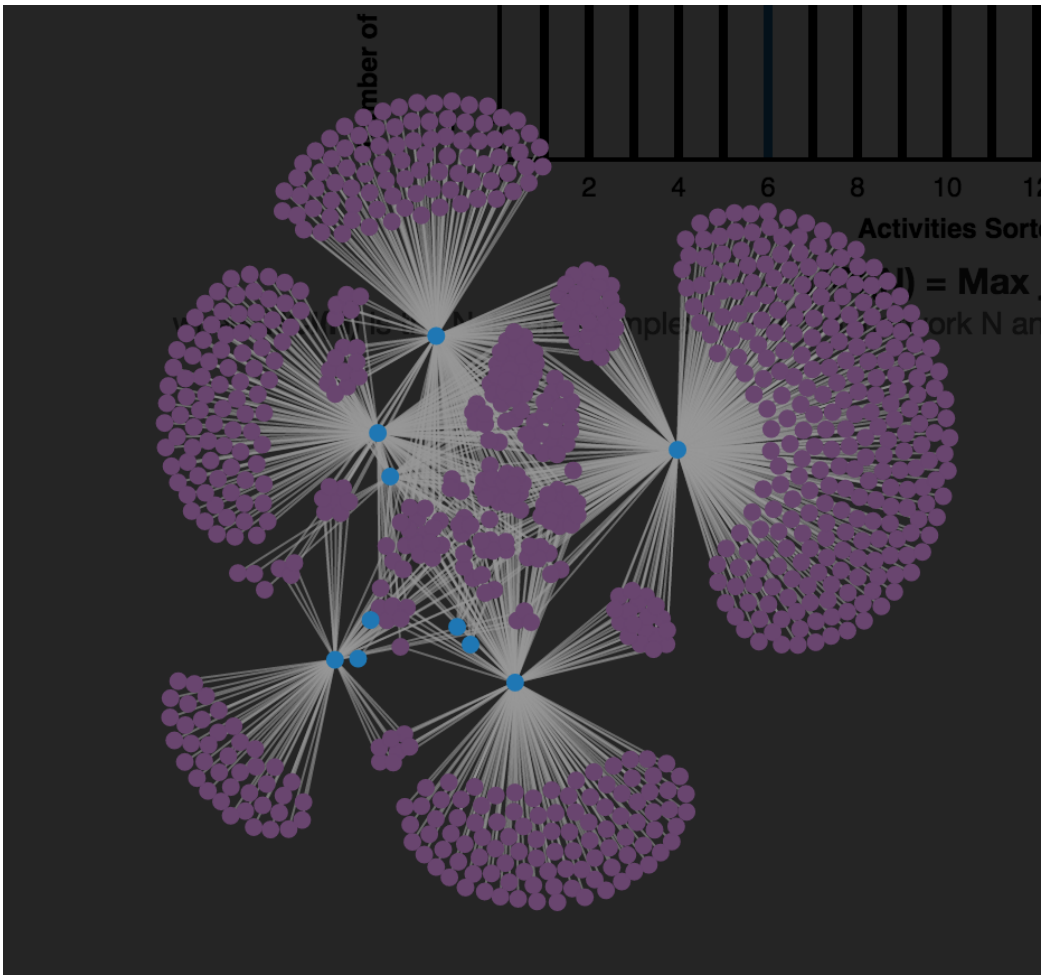


**Figure 4**

When the activities identified by the analysis are lined up based on their size as seen in the histogram of **Figure 5**, then NCI can be calculated, which can be a single number indicating how many “knots” of significant size are present in the graph.

In an example NCI calculation, mathematically NCI can be defined as follows:

*NCI(N) = Max j, X[j] >= j*

where NCI(N) is the Network Complexity Index of network N and X[j] is the number of endpoints engaged in an activity.

More details about NCI can be found in [1].

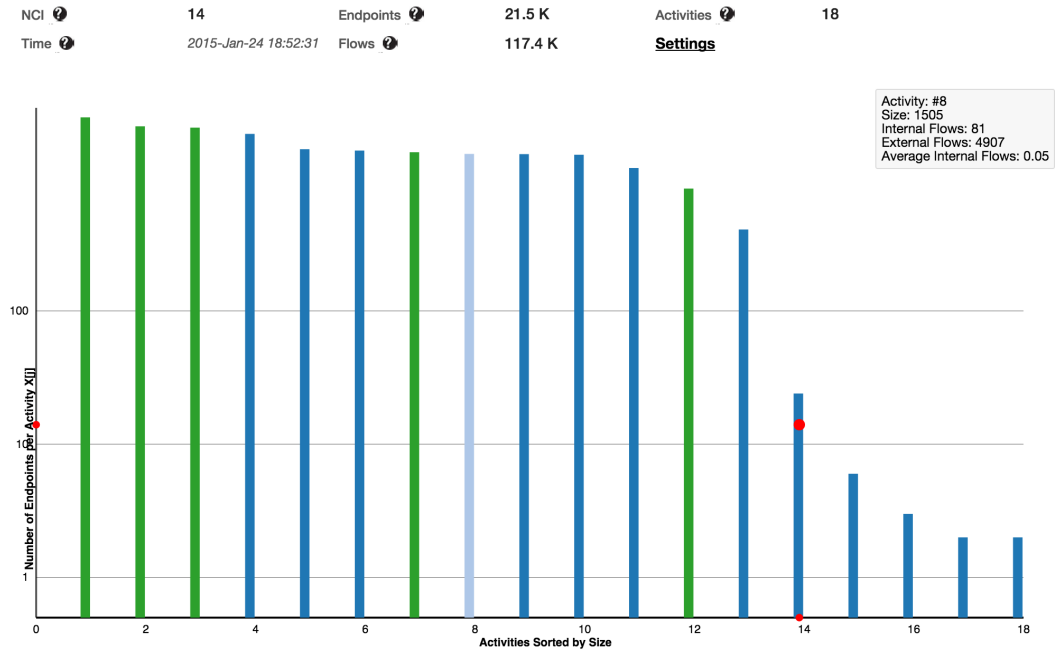


**Figure 5**

**Figure 6** shows a more detailed view into endpoints in the network, showing the nature of their interactions, whether they are to internal or external endpoints, which activity they got classified into and how they are connected to endpoints in other activities. This is just an example to demonstrate that Tapestry brings out information deeply embedded in the network and makes it available to everyone, without needing any other specialized tool or setup.

| Endpoint | Internal | External | Total ▼ | Activity | Outside Connections |
|---|---|---|---|---|---|
| 10.102.3.50 | 248 | 20978 | 21226 | Activity #1 | 18814 |
| 10.102.28.108 | 17 | 5704 | 5721 | Activity #1 | 5479 |
| 10.102.29.187 | 9 | 4712 | 4721 | Activity #1 | 4399 |
| 10.102.29.184 | 19 | 3961 | 3980 | Activity #1 | 3778 |
| 172.23.18.99 | 42 | 3715 | 3757 | Activity #1 | 2963 |
| 172.23.18.168 | 28 | 3655 | 3683 | Activity #1 | 2990 |
| 10.102.29.116 | 10 | 3627 | 3637 | Activity #1 | 3457 |
| 10.102.19.182 | 12 | 3577 | 3589 | Activity #1 | 3420 |
| 172.23.18.171 | 18 | 3424 | 3442 | Activity #1 | 2727 |
| 10.102.16.240 | 20 | 3410 | 3430 | Activity #1 | 3200 |

**Figure 6**

Tapestry tracks NCI over time and displays a chart with historical values as well as the current value of NCI (**Figure 7**). This chart provides a quick view into how the network has behaved over time. A sudden big change to NCI is expected to indicate big changes in the network. For example, deployment of a new application such as Hadoop in an enterprise can affect the NCI of that network.

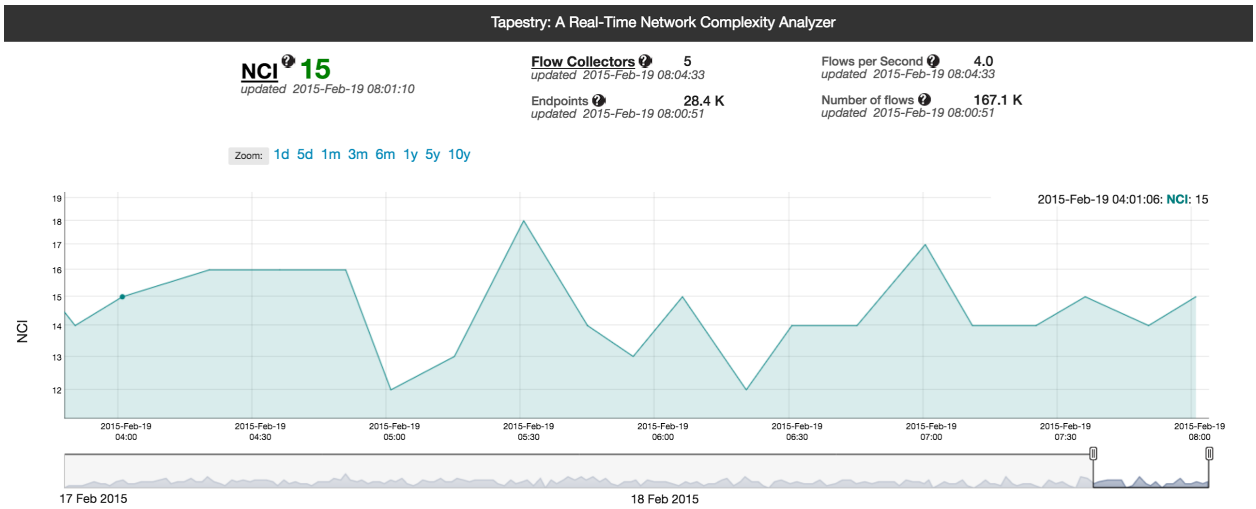


**Figure 7**

# EXAMPLE TAPESTRY PLATFORM ARCHITECTURE AND IMPLEMENTATION

The Tapestry application described in the previous section is built to execute on the Tapestry platform, which is designed to support different methods for data collection, storage, analytics, and visualization. As discussed above, the platform is designed to be flexible and to support the development of other applications that can similarly be executed on the same platform, and such applications can also involve collecting different data, performing different analytics, and/or providing different visualization. In this section, we describe the platform as it applies to the Tapestry application.

The various components making up the Tapestry distributed platform are listed below:

- Tapestry Collectors
- Tapestry Data Aggregators and Data Store
- Tapestry NCI Calculator
- Tapestry Web Server and User Interface
- Tapestry Loom Application and Loom Control Plane

## Tapestry Collectors

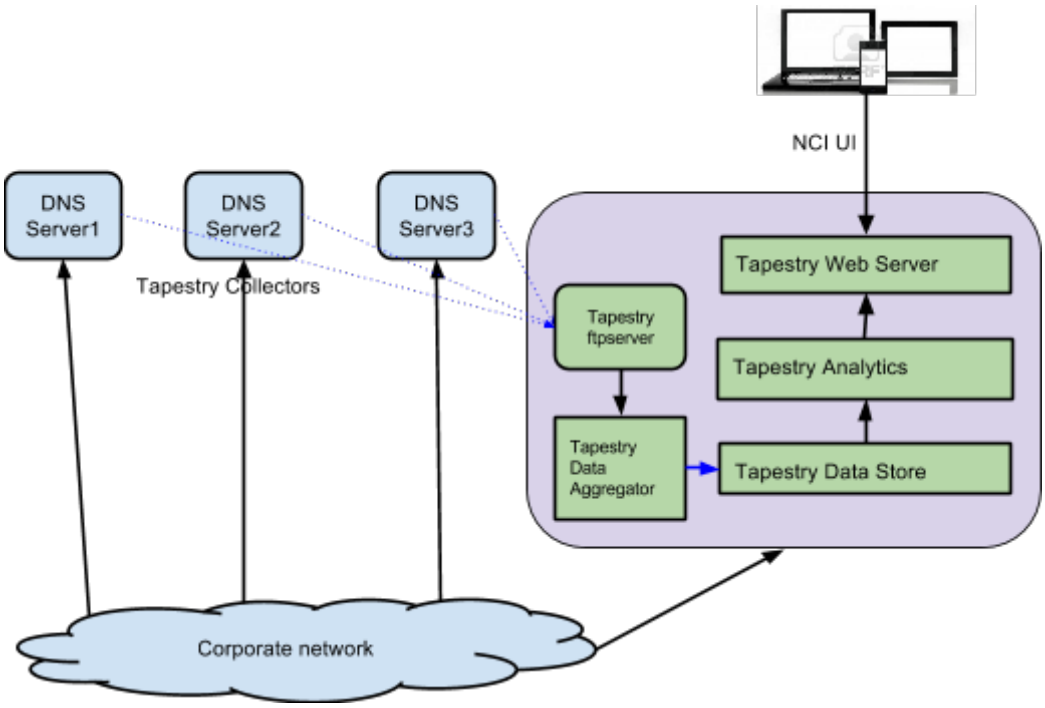


**Figure 8**

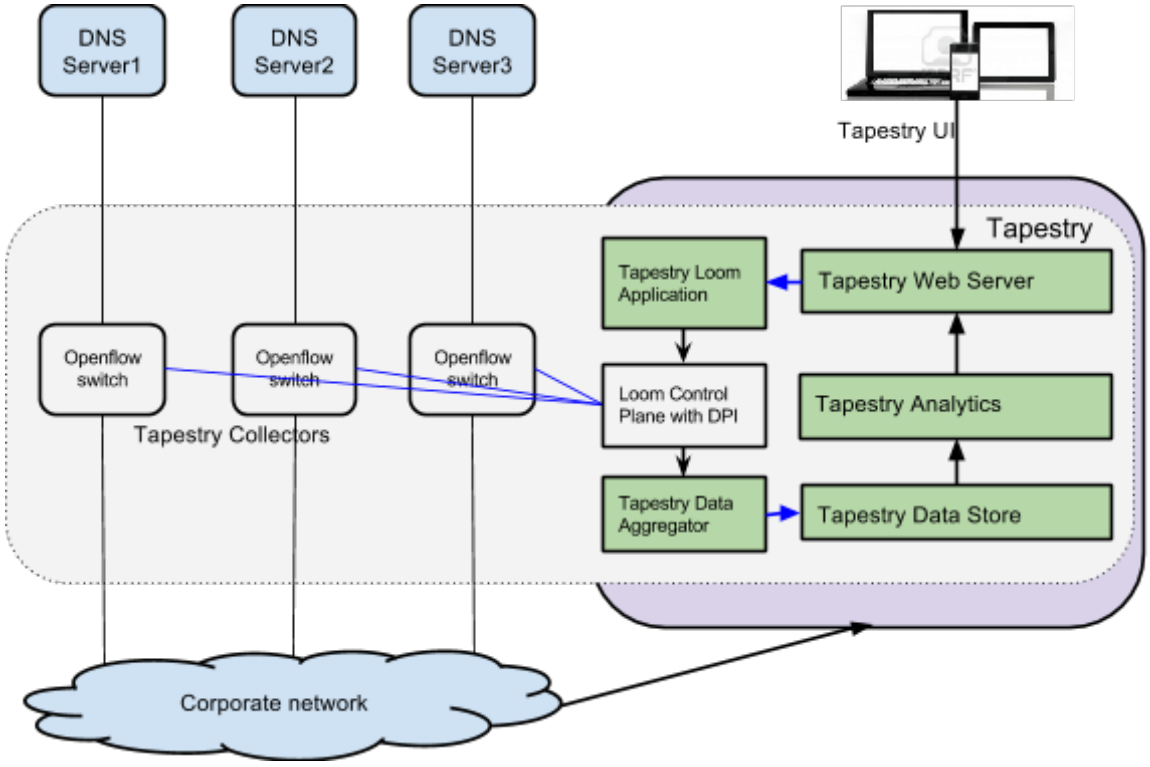


**Figure 9**

In this example architecture and implementation, Tapestry supports two types of data collectors which gather DNS responses: a) receiving DNS logs via FTP; and b) tapping DNS responses on the wire.

In the first method shown in **Figure 8**, Infoblox DNS appliances are configured to send their logs to an FTP server. The log data is gathered and parsed to capture DNS requesting client IP address as well as the resolved DNS response IP address.

In the second method shown in **Figure 9**, OpenFlow switches are placed in front of DNS servers and are setup to match and tap DNS responses. These responses are sent to the Loom OpenFlow controller. With the ability to program appropriate rules into the OpenFlow switches Tapestry can be configured to have full control of when and what type of network data is collected, making Tapestry a novel SDN telemetry platform and application that works in traditional networks.

### Tapestry Data Aggregators and Data Store

In this example architecture and implementation, Tapestry supports two types of Data Aggregators, one for each type of collector. These parse and prepare the DNS response data to be stored for further analysis. When data is collected by using DNS logs received via FTP, log lines are parsed to identify DNS requesting client IP address, and the requested IP address. When data is collected by receiving tapped traffic consisting of actual DNS responses, deep packet inspection of the DNS response provides the DNS requester and requested IP addresses.

Tapestry Data Store stores the collected and aggregated data for further analysis. The DNS response pairs of requesting and requested IP address pairs {V1, V2} are stored as directed edges in an in-memory graph store where V1 and V2 are vertices in the graph with an edge between them. Over time as new edges are formed, the graph database captures a view of all the dynamic interconnections between endpoints. This serves as an input to the NCI Calculator. In some cases, other related metadata can also be stored on vertices as well as the edges in the graph (e.g., timestamp(s), number of monitored connections between a given pair of endpoints, etc.).

### Tapestry NCI Calculator

In today's world where interconnection of any kind is of increasingly growing importance, community detection and analysis [2] has become an important field employed in various real-world networks. In an example implementation, Tapestry implements two well-known community detection algorithms to automatically identify and group different activities using the network. In addition, it is designed to support the implementation of any other community detection algorithm.

The first algorithm to be implemented was the Label Propagation Algorithm (LPA) [3]. Quoting from the paper "The label propagation algorithm uses the network structure alone as its guide and requires neither optimization of a predefined objective function nor prior information about the communities. Every node is initialized with a unique label and at every step each node adopts the label that most of its neighbors currently have. In this iterative process densely connected groups of nodes form a consensus on a unique label to form communities."

For example, the implementation of LPA worked well when the number of vertices and edges in the graph were small (under 500, 5000 respectively), but with increasing size of the graph, many

communities collapsed into an increasingly large community. We believe that there are variations of LPA that could solve this problem.

Louvain method [4] was the second community detection algorithm implemented and it greatly reduced the formation of one big community. Quoting from [4] “The Louvain method is a simple, efficient and easy-to-implement method for identifying communities in large networks. The method has been used with success for networks of many different types (see references below) and for sizes up to 100 million nodes and billions of links. The analysis of a typical network of 2 million nodes takes 2 minutes on a standard PC. The method unveils hierarchies of communities and allows to zoom within communities to discover sub-communities, sub-sub-communities, etc. It is today one of the most widely used method for detecting communities in large networks.

The method is a greedy optimization method that attempts to optimize the "modularity" of a partition of the network [5]. The optimization is performed in two steps. First, the method looks for "small" communities by optimizing modularity locally. Second, it aggregates nodes belonging to the same community and builds a new network whose nodes are the communities. These steps are repeated iteratively until a maximum of modularity is attained and a hierarchy of communities is produced. Although the exact computational complexity of the method is not known, the method seems to run in time O(n log n) with most of the computational effort spent on the optimization at the first level. Exact modularity optimization is known to be NP-hard.”

### Tapestry Web Server and UI

For example, the Tapestry Web server can facilitate the users to view Network Complexity Index (NCI) in real time or for any chosen period. In addition, it provides detailed view into the endpoint-graph, providing activity based and endpoint-based information. In an example implementation, the Tapestry Web Server can be implemented using Yaws, an HTTP high performance 1.1 web server written in Erlang, particularly well-suited for dynamic-content web applications [6]. YAWS supports the WebSocket Protocol, which enables two-way communication between clients and web servers. Tapestry Web Server and client use WebSockets for interaction with each other.

The Tapestry NCI UI can be implemented as a web client that uses HTML, CSS, and JavaScript in a single page. It can be implemented to use WebSockets to communicate with the server. Tapestry UI API has been an ongoing development and provides a rich interface as users dig into details of network interaction.

### Tapestry Loom Application and Loom Control Plane

In an example implementation, the Tapestry Loom application belongs to a family of applications that utilize the Loom control plane software to program OpenFlow switches. In particular, it uses Loom to send setup rules in OpenFlow switches to tap DNS traffic and also to collect the tapped traffic.

In an example implementation, the Loom control plane is a platform to provide a scalable and robust OpenFlow based distributed, yet logically centralized controller fabric to control OpenFlow based switch

fabric that can grow to thousands of switches. Loom facilitates functionality for the requirements of Tapestry's data collection using OpenFlow switches.

## REFERENCES


[1] http://www.flowforwarding.org/nci-article.
[2] http://en.wikipedia.org/wiki/Community_structure
[3] Usha Nandini Raghavan, Reka Albert, Soundar Kumara, "Near linear time algorithm to detect community structures in large-scale networks," *Physical Review E 76*, 036106 (2007)
[4] http://perso.uclouvain.be/vincent.blondel/research/louvain.html
[5] http://en.wikipedia.org/wiki/Modularity_%28networks%29