Autonomic Swarms for Regenerative and Collaborative Networking

Rui-ping Lua and Wee Keong Ng
School of Computer Engineering
Nanyang Technological University, Singapore
Email: rplua1@e.ntu.edu.sg, awkng@ntu.edu.sg

***Abstract*—The exponential growth of web services have necessitate the evolution of network infrastructures to meet this challenge. We envision the myriad of Internet connected devices coming together to provide a robust and reliable service network. We propose an Autonomous Swarm Network to provide autonomic capabilities to achieve our service quality goals while coping with complex and changing requirements of today's web services particularly cost-effectiveness versus service assurance. To create a high-resilient network, we incorporated features of self-management, self-configuration, self-optimization and self-healing strategies. Using a combination of fast-flux service networks, autonomic management and swarm algorithms, it becomes possible to build cost effective assurance for existing web services. We demonstrate the feasibility of our solution using the Nanyang Analytics Supercomputer with more than 20,000 agents against varying loads. We've also simulated algorithms and reconfiguration strategies. We subsequently developed a prototype swarm network of up to 500 machines.**

***Index Terms*—Autonomic Computing, Collaborative Networking, Self-Configuration, Self-Optimization, Swarm Networks, Overlay Networks, Fast Flux Service Networks**

I. Introduction

Mission critical services includes web search, on line banking and content delivery services depend on their service availability for their continued business. Any disruptions to their service would result in substantial and financial loss.

Threats towards service availability includes (1) Distributed Denial of Service Attacks (DDoS) and (2) flash crowds. In DDoS attacks, an attacker attempts to prevent legitimate users from accessing services by overwhelming the victim with an abnormally large volume of traffic. It overloads the victim and severely degrades its serviceability. A massive attack launched against Spamhaus Project (March 2013) has caused rippling slowing effects around the globe. VeriSign reports Botnets for hire at a low cost of $67.20 (per day). Combined with easy access to popular DDoS programs such as Low Orbit Ion Cannon the threat of DDoS is on an exponential rise.

Studies have shown that many organizations have chosen to over provision their bandwidth to account for potential flash crowds and DDoSs. However, bandwidth over provisioning is far from economical or effective. Recent attacks are on the order of millions of attack packets per second. This tremendous amount of attack can overwhelm even well-provisioned (Tier 1,2) networks. For example, when Michael Jackson passed away (2009), news of his death immediately results in a flash crowds that brought Google, Wikipedia, BBC, CNN, Twitter and TMZ's servers to a crawl. None of the service providers would have been able to predict this traffic outburst. This phenomenon continues to surface in a much lesser degree, but on a daily basis [1]. When such incidents occur, massive amounts of traffic shifts in the Internet. Some links becomes overloaded, and others underutilized. These traffic requirements are complex and varies greatly. Advances in Quality of Service [2], buffering [3] and Service Overlays [4] may provide respite to traffic sensitive applications. We believe that we can provide an alternative strategy to combat the above problems.

We propose the use of an autonomic network made of up collaborating network agents to maintain a reliable overlay service in spite of flash floods and attacks.

II. Our contributions

Our Autonomic Swarm Networks (ASN) follows the paradigm of Autonomic computing [5]. Autonomic computing focuses on designing a system capable of managing and administrating itself. In designing our autonomic computing system, we consider the following behaviors. [6].

1) **Self-management** - Node recruitment, retirement and promotions.
2) **Self-configuration** - Dissemination of service and swarm directives
3) **Self-optimization and Healing** - Adjusting neighbor size, update intervals and service level.

We designed a Water-Flow Algorithm, a swarm routing algorithm suitable for a tightly collaborative network. These algorithms are used for neighbor selection and packet forwarding. We then implement it using fast flux service networks on a swarm overlay infrastructure and demonstrate its effectiveness against flash crowds.

Our paper is organized as following. Section 3 provides a design overview and construction of our Autonomous Swarm Network. Section 4 details autonomic capabilities developed. Section 5 describes the network mechanics of swarm networking and how it performs packet delivery. Section 6 presents our simulation and experimental findings. Section 7 presents discussions raised during this study. Section 8 lists related work and Section 9 is our conclusions.

978-1-936968-92-3 © 2013 ICST

DOI 10.4108/icst.collaboratecom.2013.254108

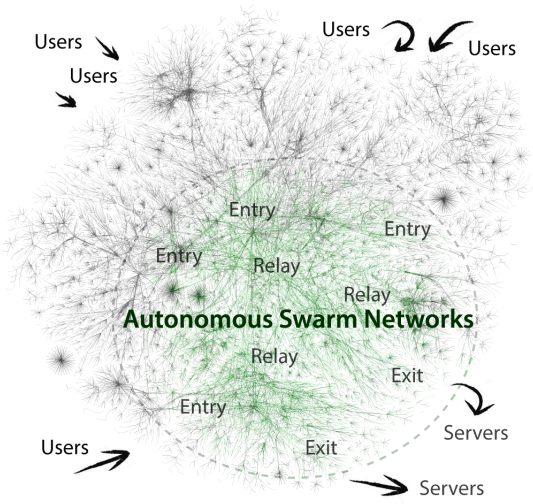

Figure 1. Internet and ASN

III. Autonomous Swarm Networks

A. Strategic positioning

We propose creating a network made up of nodes that are deployed throughout the Internet. These nodes are made up of (1) dedicated (2) client and (3) crowd-source hardware. We defend our users by positioning their servers behind our swarm network as shown in Fig. 1[1]. Our system provides a robust and reliable overlay network. Our autonomous nodes mitigate flash crowds and malicious attacks by the use of self-optimization and self-healing techniques. Our positioning also allows us to cloak our user's servers to avoid direct bandwidth or application attacks. We only allow traffic flows that correspond to our user's expectations and type to be forwarded to them. With sudden traffic increases, we recruit nodes to compensate and maintain an agreed service level. If the legitimate traffic exceeds the capacity of the user's server, we can provide waiting rooms, queues or distributed content retrieval to mitigate outrages. Our network aims to provide simultaneous protection for large number of varying servers and services. We provide certain level of 'insurance' by dynamically allocating overlay resources to compensate for any short fall in performance.

B. Design Goals

We propose a network inspired by the homeostasis process. The network maintains a predictable and reliable packet delivery under flash crowds or service attacks.

We set design goals as follows.

1) **Autonomous management and configuration** - Maintaining a control loop to ensure that the system remains within specified service level and take actions to achieve and maintain it.
2) **Functional re-composition (Group level)** - An architectural design that enables formation of large-scale networks in which multi-tiered communities are also composed in a similar fashion. This allows flexibility in reorganizing them in any fashion and levels we require.
3) **Atomization (Individual level)** - Nodes are to be as simple as possible to allow for various mix of functions to perform their roles and functions.

Swarm nodes coordinate and relay messages between clients and servers. The network performs self-organization and uses swarm algorithms to constantly improve its relay solutions. We developed translators that allows the use of established communication protocols (HTTP, FTP, etc). We use fast-flux service networks [7] to manage how clients and servers communicate with the swarm network. It allows us to direct client's request to any specific gateway node in the swarm. We then design a swarm algorithm that can be implemented in a distributed arrangement across all participating nodes. These nodes forwards data, optimized by a shared objective function. We make use of Water-flow algorithm as the basis of our swarm algorithm. Each node in the swarm is functionally indistinguishable to another node in the network. As more nodes join the network, each node will configure itself to serve the network. When the number of nodes has increased beyond a certain threshold, some of it will split away to form neighboring communities. When subsets of nodes are disrupted, neighboring nodes will fill the operational gap. This minimizes any possible disruption that the swarm network would experience under a heavy load or malicious attack.

To demonstrate the feasibility of such an autonomic swarm network, we subjected it to varying magnitudes of attacks and report our findings.

IV. Autonomic Capabilities

A. Self-management

1) Recruitment and Retirement (Network elasticity): When a node registers itself with its neighbors, they will respond with its current network statistics. The node will determine if it should take up any particular role (Seed/FDNS, Gateway (Entry/Exit) and Relay roles). As the network experiences heavier consumption or DDoS-like attacks, its load will increase correspondingly. When the load exceeds a threshold, it will contact its inactive peers and recruit them to join the network. Similarly, when traffic load of the nodes drops beneath the threshold, some of them will retire. Determination of this threshold is described later in Section IV-B Aggressiveness of recruitment and retirement must also be balanced among certain factors. These factors include (1) apparent rate of load increase and (2) estimated performance of the recruited and retiring nodes. The rate of load increase can be broadly categorized into "expected" and sudden flash floods. In 'expected' ($1-10\%$ load increase) events, we make use of addictive increase / multiplicative decrease (AIMD) to converge on the optimal number of nodes. When flash floods occur (load increase of a factor of > 2), we use multiplicative-increase to recruit as many nodes as possible to withstand the flash flood. Special care must be taken in the

[1]Image generated and modified from the opte project, 2013

TABLE I
SERVICE DIRECTIVES

Parameter	Content Type
Domain Name *(Used for a particular service)*	e.g. asn.myserver.com
Server Location(s)	IP Address(es)
Service Ports	Ports and ranges
Service Types *(Protocols to be used, can be customized further)*	HTTP, FTP, etc.
Service Rates *(Thresholds and expected patterns)*	Requests / sec

TABLE II
SWARM DIRECTIVES

Parameter	Content Type
Target Utilization *(Based on sampled max performance)*	Percentage or score
Small-increase	Recruitment rate or coefficient
Large increase	(Larger) rate or coefficient
Update interval	Timings, lower and upper bound.
Cluster size *(Max nodes in a cluster per seed)*	e.g. 8,16
Neighbor size *(Max neighbor nodes)*	8,16

retirement process to prevent large portion of the network from dropping out simultaneously, creating large gaps that disrupts its serviceability (e.g. net splits). Hence, we retire nodes with the least number of recently used links to avoid disruptions.

As we scale our system across various hardwares, we have to factor in different performance and network characteristics when we select which host to retire. In our implementation, we favor higher performance systems as compared to lesser machines. We assume that generally higher performing machines have more reliable and consistent network performance (e.g. desktops, servers) then lesser devices (e.g. laptops, tablets, cell phones). Hence, we make use of lesser performance machines to address sudden increases and rarely to form the network backbone. Mobile devices with an unreliable connection will not maintain persistent connections with its neighbors, but employ a store and forward mechanism instead. Software routers are categorized separately and are never retired.

2) Role polymorphism: Each node has complete access to all the functions required for any particular role. This allows it to morph into any role as required. These roles are as follows (but not limited to).

1) **Fast-Flux Domain Name Servers** (Session binders, Network seeds)
2) **Gateway nodes** (Inter-swarm communications)
3) **Relay nodes** (Forwarding payloads through the network)

Each role has a target performance measure that is tuned to provide optimal efficiency. Any shortfall experienced by the network will encourage appropriate role recruitment and change.

B. Self-configuration

1) Service and Swarm Directives: Different network services have different QoS requirements. We introduce QoS directives for multiple traffic requirements. This allows us to multiplex our network to support multiple servers. We craft a QoS directive with the following (but not limited to) parameters shown in Table I

The swarm network itself uses the following (but not limited to) parameters shown in Table II.

C. Policy Dissemination

Our network does not have a centralized command and control architecture, we perform a peer-to-peer update mechanism. The directives are sent to any node in the network (with authentication). When a node receives a service directive, it will spread it to the neighbors. In turn, each neighbor will spread the service update recursively throughout the swarm.

D. Self-Optimization and Healing

Each node tries to maximize its contribution and forwarding reliability to the network by following simple directives[2].

1) Maintaining optimal number of neighbors: During the lifetime of the swarm network, some nodes drop out unexpectedly. Each node has to constantly maintain a mix of its nearby and far neighbor list. This ensures that payloads will not be trapped in localized clusters of nodes easily. We facilitate this by defining a node horizon which determines the number and type of neighboring nodes in each node's contact list. Maintaining a large neighbor list reduces the number of hops required to traverse the network. However, with n neighbors, n^2 updates are required at each refresh interval, rapidly increasing the network overhead. We determine the optimal number of neighbors using a target utilization[3].

2) Cooperative maintenance of service level: As load of each node increases, it shares traffic statistics with nearby nodes periodically. These directives allow the swarm to repair the network and maintain its service level despite of disruptions from unexpected outrages or unreliable/undedicated host machines. When the target utilization cannot be achieved, more nodes are recruited to share the workload. Fig. 2 illustrates a portion of the swarm network. The square node in the center represents the FDNS/Seed node. The colors indicate the load experienced by the node normalized against its maximum performance capability. The size of the circle indicates the relative performance of the nodes. As observed as some of the nodes turn red (darker) indicating heavy load, neighboring nodes starts to share its workload. There is a bias for the workload to be shared among higher performance nodes, thus they get loaded first as shown.

E. Self-protection

When traffic pattern exceeds certain specified rates, it classifies the incoming traffic as suspicious. When the traffic is

[2]To prevent oscillations as the nodes try to achieve its target performance characteristics, we designate a $\pm 10\%$ boundary.

[3]In this study, we use a neighbor size of 8 and 16.

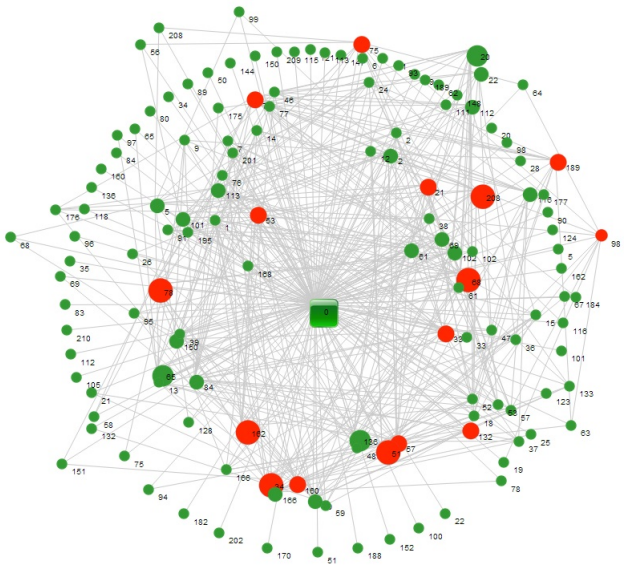

Figure 2. Autonomous Reconfigurations

classified as suspicious, more gateway nodes will be recruited in preparation for an incoming flood/attack. When suspicious user traffic is detected on certain nodes, network puzzles will be deployed to challenge some users before continuing to forward their service requests. For the remaining users, there will be no service disruption. As the traffic continues to grow, the network classifies itself as under an attack. In this scenario, larger numbers of nodes are recruited with an increased deployment towards gateway roles. Some relay nodes will morph into gateway nodes. Existing users are challenged with a network puzzle. Upon solving the puzzles, the gateway node will only forward their requests and drop all other requests. This is similar to an immune response as the system tries to distinguish between legitimate users and malice external agents while increasing its capacity to deal with the threat.

Classification of traffic patterns is performed by measuring unique source-destination pairs. Global request/connections to destination addresses are also used to determine if a flash flood / distributed attack is underway. Our network protects multiple server assets simultaneously. In some cases, attackers are only targeting a number of servers belonging to a specific group. Our gateway nodes are able to filter and direct traffic flows in a manner that minimizes the impact to non-targeted servers, minimizing collateral harm across the entire swarm.

V. Network Mechanics

Autonomous Swarm Networks can be described functionally with the following types of interactions.

1) **Inter-swarm** - Fast Flux Domain Name Service
2) **Intra-swarm** - Routing through swarms, self-organization and self-configuration for elastic networking.

A. Fast Flux Domain Name Service

Fast Flux Service Network provides a mechanism to implement a session binding system. Unlike conventional fast-flux networks, we do not host the content in the network. We manipulate return IP addresses from our fluxing name servers to manage how clients connect to the swarm. This arrangement does not require any modifications to existing web client and servers.

B. Swarm Algorithms

1) Collaboration and Shared Objectives (System View): We have learn from Wardrop Equilibria [8], that if traffic in a network distributes itself according to the individual optimal of each nodes, network flows as a whole is optimized. The pattern of traffic resulting from these decisions behaves in the same way as if a central intelligence was present to direct them. The objective function is defined in the distributed swarm parameters in Section IV-B. A popular example of distributed flow control is TCP congestion avoidance. Our approach optimizes the configuration of relay nodes (between source and destination hosts), and provides multiple solutions for packet delivery to achieve our service level.

We propose modeling consumption of individual link capacities so as to efficiently make use of available resources at any given time. In a typical setup, network devices are not explicitly collaborating with one another. In our swarm network, we are able to control our devices more tightly. Through sharing of each node's network statistics, we are able to sense/predict link congestion much earlier. By estimating consumption of the links, we are able to fairly distribute traffic flows to avoid congestion on any particular link. Consumption is expressed by a "soil" parameter as described in the next Section V-B2

2) Water-flow Algorithm (Node-to-Node routing): We studied swarm algorithms used for network routing like Ant-based Control (ABC) [9] and AntNet, Ant Colony Optimization [10]. AntNet uses an agent to determine the best route. However, we require a node (junction) based solution that is compatible with our payloads requirements. We do not keep the entire path history to prevent path-tracing. We formulate an alternative routing strategy with our Water-flow Algorithm, which takes its inspiration from a similar concept of Intelligent Water Drops [11]. The movement of water in nature underpins the inspiration for this algorithm. It is evident in the observation of flowing water, that water always finds the path of least resistance (less soil[4]). WFA is described as follows.

1) Graph of the local community is initialized in each node. The quality of the global-best T^{Best} solution is set to an arbitrary large negative number. The number of water drops is set to the number of nodes in the community. Every node has a visited node list V_C(WFA), which is initially empty. The velocity of the WFA is set to an initial value.

[4]Soil values are initially scored using (1) Latency (RTT) and (2) Utilization

2) As all the nodes are distributed into various communities, each will be a partial solution to the global optimization problem. At each node i, it will choose the next node j, within the constraints of the problem and not in the visited node list with the following probability model.
$P_i^{\text{WFA}}(j) =$

$$P_i^{\text{WFA}}(j) = \frac{f(\text{soil}(i,j))}{\sum_{k \in V_c(\text{WFA})} f(\text{soil}(i,k))} \quad (1)$$

Where,

$$f(soil(i,j)) = \frac{1}{\varepsilon_S + g(\text{soil}(i,j))} \quad (2)$$

$$g(soil(i,j)) = \begin{cases} soil(i,j) \text{if} \min_{k \notin V_{c(\text{WFA})}}(soil(i,l)) \leq 0 \\ soil(i,j) - \\ \min_{k \notin V_c(\text{WFA})}(\text{soil}(i,l)) \text{otherwise} \end{cases} \quad (3)$$

3) For each WFA moving from one node to another, the velocity is updated.

$$v^{\text{WFA}}(t+1) = v^{\text{WFA}} + \frac{a_v}{b_v + c_v \text{soil}^2(i,j)} \quad (4)$$

4) For every WFA that moves from one node to another, the change in soil is described as follows.

$$\Delta\text{soil}(i,j) = \frac{a_s}{b_s + c_s t^2(i,j,v^{\text{WFA}}(t+1))} \quad (5)$$

where b_s, c_s are arbitrary values to describe the velocity and momentum of WFA.

$$time(i,j,v^{\text{WFA}}(t+1)) = \frac{\text{penalty function}}{v^{\text{WFA}}(t+1)} \quad (6)$$

A penalty function is defined to achieve different desired optimizations.

5) The soil parameter is then updated as follows.

$$\text{soil}(i,j) = (1 - p_n \cdot \text{soil}(i,j) - p_n \cdot \Delta\text{soil}(i,j) \quad (7)$$

6) The optimal path T^{Best} is calculated by

$$T^{\text{Best}} = \max_{\forall T} \text{WFA}(T^{\text{WFA}}_{q(\cdot)}) \quad (8)$$

where $q(\cdot)$ is the quality function that returns a measure of the quality of the solution T^{Best} is the global best solution.

7) The soil on the path of the optimal solution is then updated.
8) This algorithm is repeated as the system operates and up-to-date optimal paths will be generated.

WFA also exhibits an advantage over other optimization solutions by virtue that it does not require access to the entire problem space to start optimization. WFA can perform partial localized optimization without knowledge of the entire network. Large changes in the network are captured in the soil values between the links. WFA adapts to the changing soil values quickly with little to no disruption. WFA can be easily scaled to support large networks. The speed and available bandwidth of nodes are used as features for WFA. The amount of water in each flow is abstracted from the size of the packet flows. Unlike the original IWD algorithms, WFA is designed to be executed on a massively parallel and distributed system.

Using WFA's soil parameter, we are able to model consumption of links between refresh intervals. When a refresh interval occurs, traffic statistics from neighboring nodes are updated and soil values are reseted accordingly.

C. Using the System

Fig. 3 shows how service requests and responses take place.

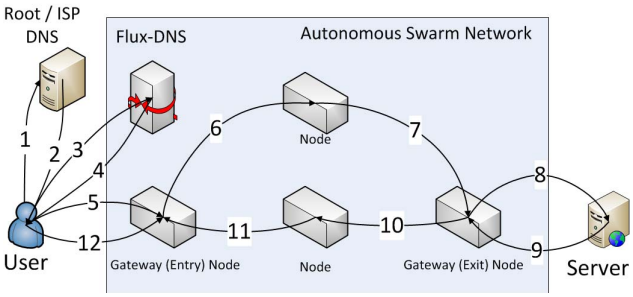

Figure 3. Forwarding mechanism

The following occurs when a user visits our web services.

- The client resolves the DNS by querying the ISPs DNS (1).
- DNS returns the IP address of our Flux DNS (2).
- The client's computer contacts the Flux DNS to resolve the domain (3).
- The FDNS assigns and returns a gateway node to the client for this session (4).
- Client contacts the gateway note with it's service request. E.g. HTTP, FTP etc (5).
- The client's request is forwarded through the swarm network utilizing WFA (6,7).
- The client request arrives at a gateway node and the payload is then deciphered. The service request is then forwarded to the server (8).
- The server responds and the response is forwarded back to through the network to the requesting client (9).
- The response is forwarded through the swarm network utilizing WFA for neighbor selection. However, return routes may not be the same. (10,11).
- The gateway node deciphers the payload and replies the user with servers response (12).

It is also noted that paths taken from client to server and server to client are explicitly made different. This reduces man-in-the-middle attacks.

TABLE III
SIMULATION PARAMETERS

Role / Agent	Allocation
Total number of nodes	24,000
ASN nodes	10,000
- FDNS / Seed nodes	625
- Entry (region) nodes	2343 (25% of 10k)
- Relay (Inter-link) nodes	4686 (50%)
- Exit (region) nodes	2346 (25%)
User-clients	4000
Malicious agents	10,000

TABLE IV
LEAST-COSTS VS WFA

	Least-costs	WFA
Outbound requests *(Simulates HTTP GET messages)*	400,000 0.38 TB	400,000 0.38TB
Received responses *(Acknowledgment of successful point to point connection)*	392717 2.996 TB	399613 3.049 TB
Failed transmission *(On first attempt, no repeats)*	7283	387 (-94.69%)

VI. PERFORMANCE EVALUATIONS

A. Super-computer simulation (Nanyang Analytics)

Simulations were carried out to determine whether WFA provides expected benefits as opposed to commonly used forwarding algorithms in a swarm network. WFA uses different inputs to determine link quality as opposed to previous IWD-based approaches. Hence, we also need to ensure that we are able to achieve similar performances.

We build a simulator consisting of 10,000 swarm nodes. These nodes are clustered together in groups of up to 16 nodes. Each cluster promotes a FDNS/Seed node. Each node randomly picks up 16 nodes to form its neighbor list. The neighbor list within each node does not necessarily contain the addresses of nodes in the same cluster. After allowing the network to boot up and reach a steady state, the link configurations are then stored in a list. Using this list, we re-evaluate the same network terrain with different algorithms while maintaining the same network configuration as a control. We collected up to 6 network configurations to be used in our simulations. While network configurations differ in individual node's neighbor list, the allocation of node roles remains largely identical. The parameters are shown in Table III.

The 4000 user-clients sends requests at 156.25 Mbps to ASN. After a short waiting period, an additional load of 10,000 nodes are used to simulate a flash crowd at an additional 300 Mbps.

A valid neighbor is defined as one nearer to the destination address then the node's current location. We considered the following selection strategies.

1) **Round-robin selection** - Valid neighbors are selected one after another.
2) **Random selection** - Neighbors are selected randomly.
3) **Nearest neighbor** - Nearest neighbor to the current node is selected.
4) **Furtherest neighbor** - Furthest neighbor to the current node is selected. However, furthest from current node does not mean nearer to the destination node.
5) **Least cost** - Valid neighbor that has the best traffic characteristics, or lowest link cost
6) **Least cost with probability model** - Lowest link cost has highest probability of being selected.
7) **Water-flow Algorithm** - Selection based on best available performance using probability model with the addition of consumption modeling and periodic network updates.

Simulations were performed on Nanyang Analytics Supercomputer. The system is described as follows.

- IBM System x iDataplex
- 300 Nodes of 2-Socket Intel Xeon 5500 series
- Quad Data Rate InfiniBand for Node Interconnect
- IBM General Parallel File System (GPFS), DCS9900 Storage System

Results have shown that "Least cost" and "Least cost with probability model" are similar and provides the lowest failed transmission. We excluded geographical selection as the system runs on a localized machine and will not be affected by 'spatial distribution' of the nodes. This is then compared with WFA in Table IV

B. Prototype

To show feasibility of such a system, we prototyped our network using 500 host machines. Each of the 500 hosts is an independent swarm agent and communicates through TCP/IP on LAN. Each host has its own IP address, in a subnet of the university's network. Internet browsers (e.g. Mozilla Firefox) are used as clients. Service request can be either GET, LIST messages sent from client's browsers. The requests received by servers indicate the number of service requests successfully forwarded to the servers. The server then chooses which request to respond to. The network is created with arrangement shown in Fig. 4. Up to 10 users are created to provide periodic service requests. By varying the rate of service requests from the user base, we can achieve different load conditions. Up to 500 hosts are used to form the swarm network (FDNS, Seed and Relay nodes). Guarantees[5] are absent to show how the swarm network performs self-configuration and management to maintain its service level. When guarantees are in place, the occurrence of timeouts becomes negligible.

The results are shown in the following Table V.

Approximate requests per second correspond to the request rate from the 10 clients. It does not include intermediate forwarded packets. We adjust the load on the network by

[5]Guarantees are used to ensure that requests and responses will be re-attempted in quick succession to reduce the failures. (1) Entry nodes perform a re-request if response is not received in time. (2) Nodes returns an acknowledgment when they successfully forwarded their payload.

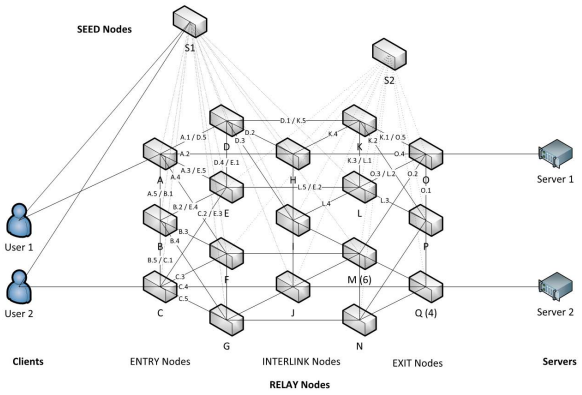

Figure 4. Network arrangement

TABLE V
PERFORMANCE STATISTICS

Type	Low-load	High-load	Extended
Requests *(Sent from the user-agents)*	31,800	56,000	440,075
Received *(Received by the servers)*	31,766	55,985	440,083
Responses *(Received by the user-agents)*	31,753	55,985	440,071
Timeouts *(Failed connections detected)*	32	436	104
Approx request/s	5	25	15
Avg upload (Kbps)	290.74	1378.46	804.65
Avg download (Kbps)	290.43	1378.09	804.70

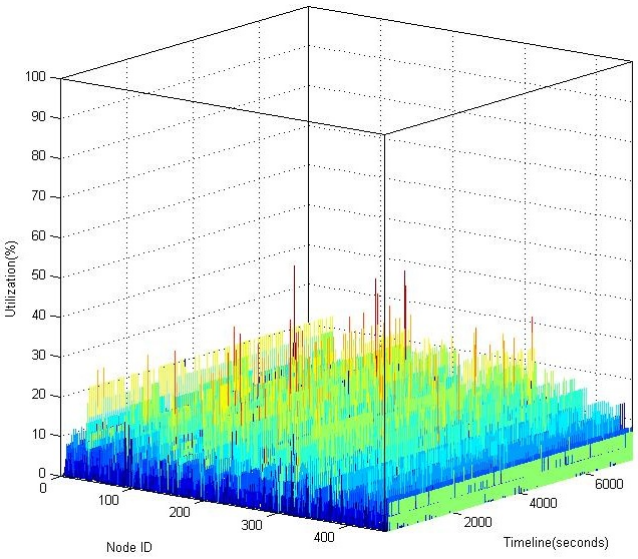

Figure 5. Low-load utilization

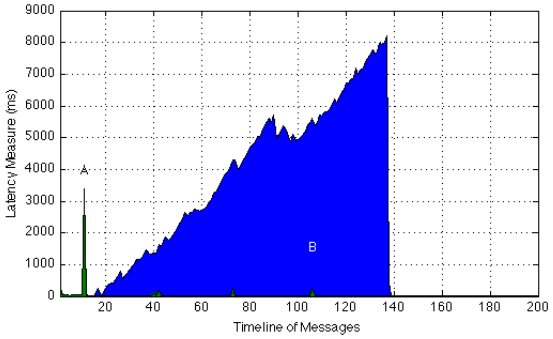

Figure 6. Latency of WFA vs "Least-cost" selection

varying the number of requests made by the clients. Each request and reply (message) was approximately 8 KB.

1) Low-load scenario: We ran the swarm network for over 2 hours under a low load scenario. The results is shown under "Low-load" in Table V

We illustrate the utilization of each node over the entire experiment in Fig. 5.

The Y-axis (Utilization) is calculated based on the maximum performance of the node. This is determined prior to this run by increasing the load until it failed[6]. The X-axis (Node ID) is used to identify each node. The Z-axis (Timeline) refers to the time elapsed which in Fig. 5 is approximately 2 hours.

The load is distributed relatively fairly over the entire network with the use of WFA. Most of the nodes generally experience up to 20% utilization. If least-cost selection was used, the link would aggregate and result congestion. Without the combination of consumption modeling and link updates, traffic flows will not be distributed as effectively.

To understand the impact of WFA and the benefits of autonomic real-time re-configuration of the network, we focus on one node. We show the congestion experienced by the node using both WFA (A) and "Least-cost" selection (B) in Fig. 6.

The latency measure in Fig. 6 refers to the time taken for a valid response to a forward request is received. It can also similar to Round-Trip-Time experienced by inter-node communications. As the node experiences congestion, WFA and autonomic responses triggers a re-organization. This results in (A), a spike that lasts momentarily and affects only a few service requests. Without these features, the node has no ability to share its workload with its neighbors. Hence, the load builds up and delays increases (B). When the delay has reached a specified timeout, the client (and network) has to re-negotiate another node (and path) to be used, and the utilization drops off as the node is deprecated due to bad performance (or through a link update).

To compare utilization of nodes across a variety of hardware, we determine the maximum number of requests each node is able to process without any failures[7]. Instantaneous

[6]This includes software exceptions, or thread unresponsive

[7]Failures occur when nodes throw exceptions when they are unable to bind to sockets fast enough. There are also errors from constraints of the operating systems used.

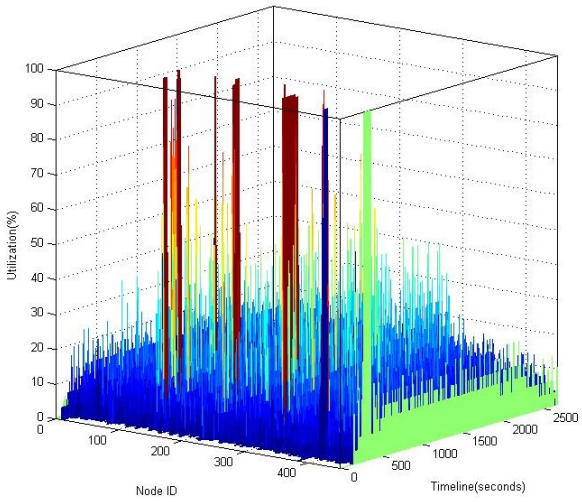

Figure 7. High-load utilization

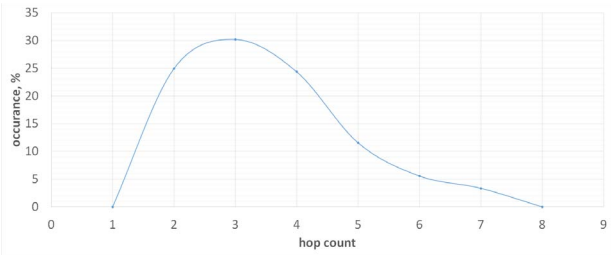

Figure 8. Hop count statistics

utilization is then normalized against the individual maximum of each node.

2) High-load scenario: We then increase the load by a factor of 5 to exert more pressure on the system. With a much higher loading factor, we expect to observe traffic aggregations and weakness of the system. This experiment is conducted over a much shorter period of 40 minutes. Results are shown under "High-load" in Table V.

We then illustrate the load experienced by the network in Fig. 7.

We observe that some nodes experience heavy utilization. This occurs because they are initially high performance nodes. WFA performs with a probability model that is biased towards higher performing nodes. This results in a sudden aggregation at these nodes. As these nodes are loaded, autonomic responses triggers a re-organization that distributes the workload and thus reducing the congestive effects.

The number of aggregation junctions that occur in this scenario is dependent on the number of desired hops we have specified. In this scenario, we instruct the network to perform forwarding within 5 hops (across 4 nodes). To achieve this goal, the network configures each node to have up to 8 neighbors. Statistics of all hops is shown in Fig. 8 in reference to the percentage of their occurrence.

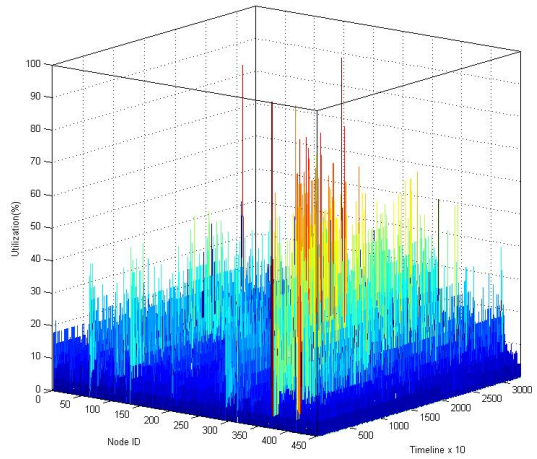

Figure 9. Extended utilization

3) Extended scenario: To understand how this network will perform over longer periods of time, we operated the network for 10 hours with up to 440,000 service requests. Results are shown under "Extended" in Table V.

Utilization is shown in Fig. 9.

The system remains largely consistent over a long period of time[8]. When aggregation occurs, a re-configuration is performed. However, we observe that after a re-configuration is completed, aggregation is still possible as the network runs for longer period of time. Hence a series of re-configuration occurs throughout the operation lifetime. The number of requests received exceeds those that is sent, as messages are re-transmitted when an acknowledgment time-outs.

To facilitate current and future studies, we have also constructed the following virtualization cluster.

- 16 units x 2-Socket Intel Xeon E5 series processors *(256 physical, 512 logical processors)*
- 16 units x 64 GB 1600Mhz RDIMM Memory *(1TB Memory)*
- 2 units x 10GBase-T switches

VII. DISCUSSION

A. Alterations and constraints for existing services

The swarm network does not require modifications to existing protocols or web servers. However, certain provisions must be made for the system to be utilized. Service providers have to map their domain name to our fluxing name servers. They will need to provide us with the expected traffic flow and type so that we can effectively filter illegitimate traffic. We also need to know the addresses of their servers so that we can perform the mapping accordingly.

Positioning servers behind our swarm network provides anonymity that cloaks the addresses of the servers. However, this also prevents servers from gathering information directly

[8]The graph shows utilization every 10s due for scaling purposes

from the clients. From the servers view, connecting agents are the exit nodes of the swarm network and not the users themselves. We do not inherently forward the client's identity across the swarm network unless explicitly specified. To maintain sessions, the servers have to read meta-data from the forwarded packets or requests to retrieve user information.

B. Resource impact on infrastructure

To facilitate autonomic functions of the swarm network, majority of the overhead comes from (1) Updating neighbor list and (2) Link updates.

1) Updating neighbor list: To maintain integrity and reliability of the network, the neighbor list needs to be current. Neighbor lists are updated at various intervals depending on the network load. Interval t_{low} is used when network utilization is below 50%. Interval t_{high} is used when utilization is above 50%. This number is chosen as a conservative limit for our study. At each update, each node will contact all it's neighbors and request their neighbor list. Nodes will then return their neighbor list so that each node can acquire it's designated number of neighbors. The expected outgoing overhead is described in Equation 9.

$$\text{Overhead}_{(\text{neighbor})} = N \cdot \frac{(n \cdot r_{\text{data}})_{\text{req}} + (n \cdot (n \cdot h_{\text{data}}))_{\text{res}}}{t_{[\text{low,high}]}} \quad (9)$$

N = total number of nodes in the network.
n = number of current neighbors of a particular node.
$n = [0, ..., n_{\text{max}}]$
n_{max} = max size of neighbor list defined in swarm directive
r_{data} = request (control) message $\leq$ 1KB
h_{data} = host data (IP address, etc.) $\leq$ 2 KB

2) Link updates: Each node periodically broadcasts it's link status to allow the network to perform better link selection. This occurs periodically at intervals of T. T varies according to the variance between expected link status provided by the consumption model and the actual link status. If the variance is large, more frequent updates are required.

The link update overhead is described in Equation 10.

$$\text{Overhead}_{\text{link update}} = N \cdot \frac{n \cdot s_{\text{data}}}{T} \quad (10)$$

s_{data} = status of link (utilization, latency, RTT etc.) $\leq$ 1 KB

Under normal conditions, optimizing re-configurations can take place over hours or days. However, when a DDoS attack starts, re-configuration has to take place within minutes to minimize the service disruptions.

To prevent a cascade failure whereby an sudden increase volatility of the network results in large number of updates, we have capped the maximum updates to 10% of maximum utilization of each node.

To show that the additional resources required is kept to a minimal, we delayed the start of client requests. Fig. 10 shows 480 relay nodes, with a maintenance[9] period from 0 to 45. After which 10 users send requests at 100ms intervals.

[9]Updating of neighbor and links, without any external traffic

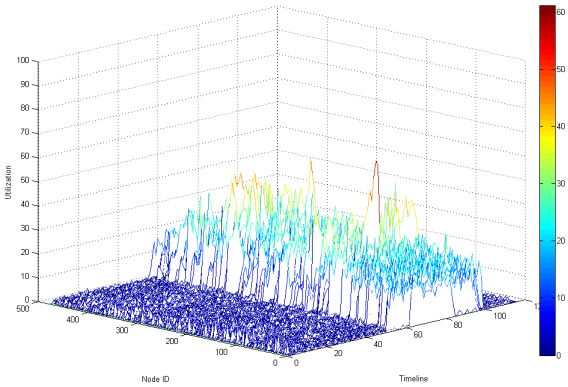

Figure 10. Maintenance and Overhead

The requests stops at 100, and the swarm returns to it's maintenance mode. This shows that we are able to keep the overhead to a minimal (1-3%) of total traffic.

C. Further works

Understanding hostile attackers and their strategies allows us to explore game theory as a means to generate effective counter-strategies [12]. In some conditions, it may be preferable to fragment the network to isolate portions of network under attack. While in others, directing packets to black holes may be appropriate to avoid catastrophic overload. We can use it to determine the appropriate arrangement of the network to best weather the attack. Using computational and bandwidth cost, we can formulate games and strategies to our advantage. We also ported our system to run on mobile devices to create larger and more pervasive networks.

VIII. RELATED WORK

Generally DDoS defense mechanisms can be categorized as (1) Preventive, (2) Reactive and (3) Alternate Topologies [13]. Preventive measures such as resource accounting attempts to provide fair service for users. QoS regulators are proposed to reside between public networks and end servers. Fairness policies can be implemented that provides good user protection. Authenticated tokens can also be used to send requests to the servers. Resource brokerage can also be used to negotiate user/server agreements. Reactive measures detects network anomalies and respond accordingly. D-WARD, Pushback [14] and many other techniques perform malicious traffic detection and rate limiting. However these implementation faces economic barriers as service providers seldom view protecting competitor's network as in line with their self-interest and profits.

Reconfiguration approach changes the topology of the victim network by either adding more resources or isolating systems under attack. (1) Overlay networks and (2) Indirection infrastructure are such techniques. Reconfigurable overlay networks allows distributed applications to detect and recover from path outrages and periods of degraded performance.

Indirection Infrastructure such as I3: Internet Indirection Infrastructure and Phalanx[15] provides rendezvous based communication. This prevents a direct attack on servers but require modifications to existing services.

Our Autonomous Swarm Networks provides a robust reconfigurable overlay network. Similar to Resilient Overlay Network, Tapestry, SOS, and OASIS[16], we provide a reconfigurable network topology. However, these are mostly a target-side solution which can be still potentially overwhelmed by a bandwidth attack at its access points. Our ASN provides autonomous reconfiguration of all access points. Recruiting gateway nodes as necessary, or enlarging relay capacity to perform traffic shaping for our users. We do not use centralized monitoring and reporting tools, but depend on swarm algorithms to achieve overall optimization. Node list and links are determined autonomously and adapts to various network conditions. Traditional overlay networks does not actively and consistently seek out the best topology and hence, not as effective or efficient. P2P networks such as Skype and BitTorrent etc, do not have our extensive control over each node, changing their roles as required. We have also design our system as a fast-fail network. Each node recovers quickly in the event of a flash flood. We have multiplexed our infrastructure to allow multiple web servers to operate simultaneously. Each node compartmentalize each virtual routing circuit and prevents the overloading of one service network to affect another.

Overlay networks also requires strict ownership over their nodes. This provisioning of such dedicated hardware resources might be cost prohibited for any single party. Our swarm network can be deployed on multiple platforms across dedicated and non-dedicated hardware. We provision for various types of hardware and score them accordingly to be used by the network. ASN can be weaved into existing Internet applications to further the common good, as shown by many crowd sourced grid computing softwares. The robust nature of our fast-flux service network provides a high level of serviceability while using unreliable hosts. This reduces the dependency on dedicated hardware for its continued operation. Our design allows the use of our network with only minor modifications to domain name records. This allows it to be easily deployable with little to no disruption to existing services.

IX. CONCLUSION

In this paper we present our concept of an Autonomous Swarm Network and its implementation. We discussed on various autonomic features used to create a self-administrating network infrastructure. These features demonstrate self-management, self-configuration, self-optimization, self-healing and self-protection capabilities. We based our swarm algorithms on the self-interest of individual nodes used to optimize flows.

Fast-flux techniques increases the network robustness by maintaining very high service availability while using unreliable hosts. The swarm network constantly re-configures itself through a combination of autonomic responses and our Water-flow Algorithm. All these techniques allow us to extend the serviceability of cloud services under varying loads and DDoS attacks. To demonstrate the feasibility of our Water-flow Algorithm, simulations were performed at the High Performance Computing Cluster at NTU. Up to about 20,000 agents and 400,000 service requests show that WFA provides a competent mechanism to forward payloads. We then deploy a swarm of up to 500 hosts. Each host possessing its own unique IP address and performance characteristics. We then subjected it to varying loads and periods of time. Our results illustrate benefits and expected characteristics of our system.

REFERENCES

[1] A. Srivastava; B. B. Gupta; A. Tyagi; A. Shamn; A. Mishra, "Recent Survey on DDoS Attacks and Defense Mechanisms," *Advances in Parallel Distributed Computing, Communications in Computer and Information Science*, vol. 203, pp. 570-580

[2] A. Kumar V; S. G. Thorenoor, "Analysis of IP Network for different Quality of Service ", *International Symposium on Computing, Communication, and Control (ISCCC 2009)*, vol.1, pp. 79-84

[3] R. Bahl; C.M. Sharma; M.K. Malik, "Comparative Study and Analysis of Different Types of Buffering in Go-Back-2 Network in NS2," *UkSim 13th International Conference on Computer Modeling and Simulation (UKSim), 2011*, pp.498,500, 2011

[4] M. Kodialam; T. V. Kakshman; J. B. Orlin; S. Sengupta, "Oblivious routing of highly variable traffic in service overlays and IP backbones," *25th IEEE International Conference on Computer Communications*. vol. 17 no. 2, April 2009 pp. 459-472

[5] Kephart, J.O.; Chess, D.M., "The vision of autonomic computing," *Computer* , vol.36, no.1, pp.41,50, Jan 2003

[6] S.R. White; J.E. Hanson; I. Whalley; D.M. Chess; J.O. Kephart, "An architectural approach to autonomic computing," *International Conference on Autonomic Computing*, pp.2 , 9 , 17-18 May 2004

[7] J. Nazario; T. Holz, "As the net churns: Fast-flux botnet observations," in *Malicious and Unwanted Software, MALWARE 2008*, pp. 24-31, 2008

[8] J. Wardrop, "Some theoretical aspects of road traffic research", *Road Engineering Division Meeting*, 1952

[9] R. Schoonderwoerd; O. E. Holland; J. L. Bruten; L. Rothkrantz, "Ant-based load balancing in communications network," *Adaptive Behavior,* vol. 5, no. 2, pp. 169-207, 1997

[10] G. D. Caro; M. Dorigo, "Antnet: Distributed stigmergetic control for communications networks," *Journal of Artificial Intelligence Research*, vol. 9, pp. 317-365, 1998

[11] S. Hosseini; S. Behesthi, "Problem solving by Intelligent Water Drops," *IEEE Congress on Evolutionary Computation*, pp. 3266-3231, 2007

[12] A. Bensoussan; M. Kantarcioglu; C. Hoe, "A Game-Theoretical Approach for Finding Optimal Strategies in a Botnet Defense Model," *Lecture Notes in Computer Science,* vol. 6442, pp. 135-148, 2010

[13] A. Keshariya and N. Foukia, "DDoS defense mechanisms: A new taxonomy," *Lecture Notes in Computer Science (Data Privacy Management and Autonomous Spontaneous Security)*, vol. 5939/2010, pp. 222-236, 2010.

[14] Y Xu; R Gurin, "On the robustness of router-based denial-of-service (DoS) defense systems" *SIGCOMM Computer Communications Review*. Vol. 35, No. 3 pp. 47-60 July 2005

[15] C. Dixon, T. Anderson, and A. Krishnamurthy, "Phalanx: withstanding multimillion-node botnets," *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation, 2008.*

[16] J. Kurian; K. Sarac. "A survey on the design, applications, and enhancements of application-layer overlay networks." *ACM Computing Surveys CSUR*. Vol. 43, 1, No. 5, December 2010.