ARTICLE IN PRESS

Computer Networks xxx (2009) xxx–xxx

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Analysis of an anycast based overlay system for scalable service discovery and execution

Tim Stevens [*], Tim Wauters, Chris Develder, Filip De Turck, Bart Dhoedt, Piet Demeester

Ghent University – IBBT, Department of Information Technology (INTEC), Gaston Crommenlaan 8, Bus 201, 9050 Ghent, Belgium

ARTICLE INFO

Article history:
Received 4 July 2008
Accepted 11 August 2009
Available online xxxx
Responsible Editor: I.F. Akyildiz

Keywords:
Anycast
Overlay network
Scalable state dissemination
Distributed hash table
Threshold-based updates

ABSTRACT

Support for anycast in the IP network layer allows one source node to contact a single member out of a group of destination nodes configured with the same IP address. Due to the stateless nature of the IP protocol, subsequent packets from the same source node targeted at the same anycast group may arrive at different group members. Consequently, native IP anycast cannot be applied directly to support distributed session-based services. For this reason, an anycast overlay architecture combining the transparency offered by native anycast with support for stateful communications has been proposed. In this paper, we investigate the operational impact of deploying this overlay architecture. Performance evaluation of a data plane prototype implementation for an anycast overlay node shows that high throughput and small latency can be achieved. Additionally, we show how threshold-based update triggering, in combination with an appropriate inter-proxy update strategy, delivers control plane accuracy with minimal network overhead.

© 2009 Elsevier B.V. All rights reserved.

1. Introduction

IP anycast enables communication between a source host and one member of a group of target hosts, usually the one nearest to the source [1]. As such, anycast is considered a powerful tool for realizing transparent, scalable and reliable communications with stateless distributed network services. The use of replicated DNS root servers listening to a common—anycast—IP address is an example application where anycast has proven useful [2].

At present, there are limitations that prevent widespread adoption of IP anycast in general, and its adoption for network service provisioning more specifically. First, session-oriented services (including all applications implemented on top of TCP) cannot take advantage of this addressing mode, because subsequent packets from the same source host (and session) may be routed towards a different target host. Another anycast limitation is its poor global routing scalability due to the fact that routes to anycast groups cannot be aggregated: widespread adoption of end-to-end native IP anycast would undoubtedly lead to huge and unmanageable routing tables. Possible solutions for this issue have been proposed by Katabi and Wroclawski [3] and Ballani and Francis [4].

Inspired by PIAS [4], we introduced ASTAS: a proxy-based Architecture for Scalable and Transparent Anycast Services [5]. ASTAS proxies are regular routers augmented with anycast-specific packet processing capabilities, including intelligent forwarding of client requests and registering anycast resources. Using the ASTAS architecture, distributed network services can be scaled to a large number of consumers and resources, and this in a transparent way from an end-user perspective. In addition to network state and metrics, the proxy infrastructure uses server state information to forward service requests to the most suitable location, which is not possible using only IP anycast. ASTAS overlay nodes are stateful to be able to individually assign sessions to target servers in a flexible way.

In this paper we focus on the operational impact of deploying the ASTAS overlay, i.e., we investigate data plane

[*] Corresponding author. Tel.: +32 (0) 93314900.
E-mail address: tim.stevens@intec.ugent.be (T. Stevens).

1389-1286/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2009.08.008

scalability for ASTAS proxies and control plane network overhead related to system accuracy.

Based on the Click modular router [6] model, we discuss how the data plane of regular routers can be augmented to an ASTAS proxy in an incremental way. Therefore, ASTAS-specific Click components are introduced and potential issues for the forwarding path are raised. Performance evaluation results on regular PC hardware indicate that embedding ASTAS functionality in routers does not significantly influence forwarding latency and throughput for regular unicast packets. Even for stateful anycast communications, a packet forwarding rate close to 400,000 packets per second can be achieved.

ASTAS nodes have to be informed about the local resources' state and the remote aggregated resource state to be able to forward new session requests to the most suitable location. For this reason, we assess control plane scalability for both resource-proxy and inter-proxy updates. In order to reduce the update frequency with controllable accuracy, we propose a threshold-based update triggering mechanism for resources and proxies, which is then evaluated using a mathematical model based on continuous time Markov Chains (CTMC). Next, analytical and simulation results for xDSL and cable operator networks show how proxy location and inter-proxy update strategies impact overall state dissemination scalability.

This paper is structured as follows: Section 2 provides an overview of the anycast overlay architecture. Next, router extensions for overlay proxy nodes are discussed and data plane performance is evaluated in Section 3. In Section 4, we present an analysis of the two orthogonal problems related to ASTAS control plane scalability: resource update triggering conditions and inter-proxy update strategy. Section 5 summarizes the main results of this paper.

2. ASTAS architecture

2.1. IP anycast limitations

Because IP anycast forwards packets to the nearest member of an anycast group, it could prove useful as a transparent *service discovery* primitive. For single request-response services such as DNS, it can even support the entire service and increase service scalability by means of implicit coarse-grained load balancing between the anycast group members.

Despite these promising features, the use of IP anycast is not widely adopted and production use is essentially limited to DNS root server replication [2]. According to Ballani and Francis [4], the main reason for this is the *lack of IP routing scalability* inherent to native anycast. First, IP anycast routes cannot be aggregated and widespread adoption would lead to an explosive growth of IP routing tables. Since anycast group members—using the same IP address—can be scattered all over the Internet, a distinct routing entry is needed per anycast group. Secondly, anycast group dynamics (i.e., joining and leaving members) necessitate frequent changes to a relatively slowly converging IP routing configuration, possibly leading to network instability. In general, intra-domain and inter-domain routing protocols and algorithms are designed to operate in environments with a quasi stable underlying network configuration. As such, these protocols are not optimized to handle frequent changes to the network topology.

With the intention to use IP anycast for transparently providing scalable remote service execution, two additional limitations arise:

(i) IP anycast does not support session-based communications.
(ii) IP routing in the Internet is static (shortest path) and does not support multiple constraint routing or traffic engineering in general.

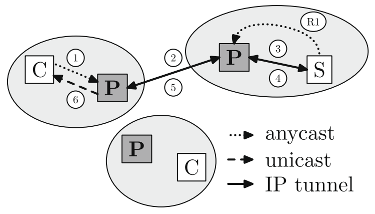
Today, most Internet traffic originates from TCP-based communications. Due to limitation (i), these services cannot take advantage of IP anycast apart from the service discovery feature. Limitation (ii) implies that anycast targets cannot be selected based on volatile network (e.g., congestion) and/or target conditions (e.g., current server load).

Taking into account both the strengths and weaknesses of IP anycast, we proposed ASTAS, an Architecture for Scalable and Transparent Anycast Services [5] that is based on PIAS [4]. For a scalable service delivery platform, the main advantage of ASTAS over PIAS is the stateful nature of all proxy components, which allows a fine-grained distribution of service requests over the available resources. For an in-depth discussion of the ASTAS architecture and a detailed comparison with PIAS, we refer to [5]. However, for the sake of clarity, the remainder of this section provides an overview of the ASTAS platform.

2.2. ASTAS overview

The ASTAS overlay infrastructure consists of a combination of two types of nodes: *client proxies* (CP) and *server proxies* (SP). Both client proxies and server proxies are special routers advertising their proximity to the anycast IP range into the routing substrate. By doing this, the proxy routers force IP packets with an anycast destination address to pass through the overlay. When a client initiates a new session to an anycast destination, the closest client proxy registers the new session and selects an appropriate server proxy to forward the request to. The server proxy receiving the new session then selects the most suitable server to handle the request.

Fig. 1 depicts the steps involved in setting up a session between a client and a target anycast server through the proxy system. Step R1 registers a server with unicast address **S** for the service offered by anycast address **A** and port **b**. In order to inform the nearest server proxy (SP) of its presence, the server addresses the registration message at the anycast address **A** and registration port **c**. Note that this registration uses native anycast to reach the *closest* server proxy. At this point, the SP configures an IP tunnel (IP-in-IP encapsulation, see [7]) to the unicast address **S**. Next, a client can initiate a session by sending a packet addressed to the anycast service of choice (step 1). When the packet arrives at the *closest* client proxy (CP), it is tunneled to a suitable SP (step 2), where it is tunneled again towards



Step	Packet headers (From → To)
R1	S:b → A:c
1	C:a → A:b
2	CP:C:a → SP:A:b
3	SP:C:a → S:A:b
4	S:A:b → SP:C:a
5	SP:A:b → CP:C:a
6	A:b → C:a

Fig. 1. Anycast communication through the proxy system. In the table capitals refer to IP addresses and lowercase characters point to the TCP/UDP port used.

a target server (step 3). The return path (steps 4, 5 and 6) is realized in the same way. *Stateful tunneling* occurs twice in each direction (in CP and SP) and is necessary to guarantee session continuity. The IP tunnels cannot be avoided on the return path because both the CP and SP have to monitor the session state, for which packets have to traverse the system in both directions. This also implies that target servers need to be aware of the ASTAS infrastructure, since an IP tunnel is maintained between each target and its SP, in both directions. However, target servers do not discover the SP unicast IP address and tunnel packets towards the anycast address (step 4).

Stateful communications are not explicitly supported by the proposed overlay mechanism since steps 1 and 4 in Fig. 1 cannot guarantee that subsequent packets from the same session arrive in the same proxy. However, in practice there are two reasons why the overlay infrastructure suffices to support stateful communications. First, the number of proxies is relatively small compared to the number of network nodes, meaning that a single link or router failure is unlikely to cause a client (or server) to swap to another proxy node. Secondly, the distance between a client (or server) and its closest proxy node is usually significantly smaller than the end-to-end distance between a client and a server, thereby reducing the chances for a failure on the path segment between client (or server) and proxy node.

3. ASTAS data plane

3.1. Router extension design

The anycast overlay architecture discussed in Section 2 requires extra functionality in network routers that are upgraded to anycast proxies. No upgrade is necessary for regular routers (i.e., routers not aware of the overlay) to be able to forward the anycast packets. For non-anycast (unicast) traffic, proxies behave just like regular routers.

Fig. 2 depicts a modular decomposition of a simplified (IPv6) router forwarding plane with ASTAS extensions, based on *Click* router components [6]. When the router receives a packet on one of its ingress interfaces (module PollDevice), a first classifier determines whether the packet

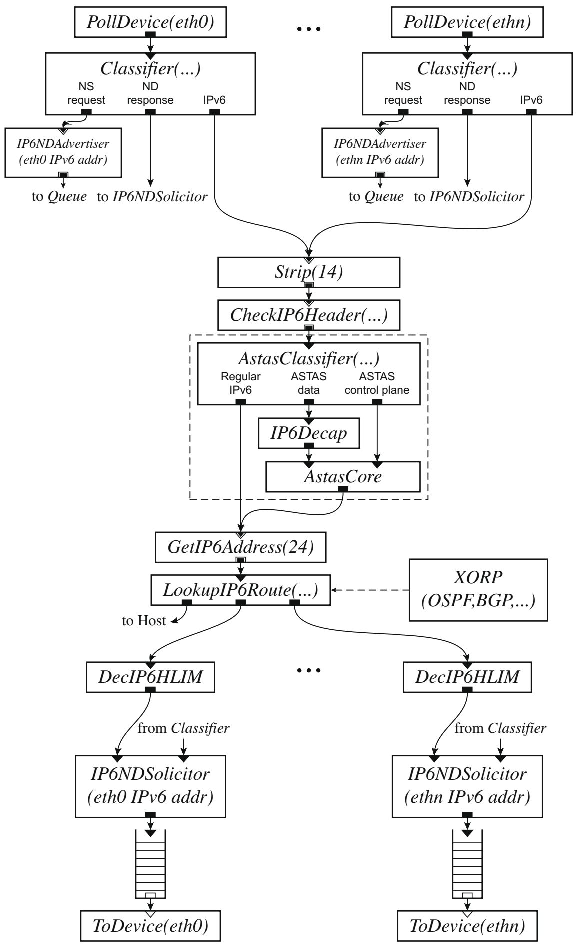


Fig. 2. Modular decomposition of an ASTAS router.

received is related to the IPv6 neighbor discovery protocol (IPv4 address resolution protocol (ARP) equivalent) or a regular IPv6 datagram (third output port from the left). Following the path of the regular IPv6 packet, the next module strips the first 14 bytes of the packet to discard the Ethernet header. As its name suggests, the CheckIP6Header module performs some sanity checks on the IPv6 header (e.g., valid IP version number and packet length) and drops the packet if the check fails. Once the packet passes the sanity checks, it arrives at the ASTAS components grouped in the dotted box. There, another classifier (AstasClassifier) differentiates between regular (non-anycast) IPv6 traffic, ASTAS data plane and ASTAS control plane traffic. Again following the path for regular packets, the next module (GetIP6Address) extracts the destination address from the IPv6 header (offset equals 24 bytes) for the actual routing element, LookupIP6Route. According to its routing table, this component forwards the packet to the correct egress interface or the host operating system for local delivery. Before pushing the packet to the egress interface, the IPv6 hop limit (IPv4 time to live (TTL) equivalent) is decremented by module DecIP6HLIM, whereafter the Ethernet header is prepended by the egress neighbor discovery module IP6NDSolicitor. The following paragraphs discuss the ASTAS components in detail and present concluding remarks for the ASTAS router extension design.

3.1.1. *ASTAS classifier*

This component distinguishes between four different types of packets: regular IPv6 packets, data packets to an anycast destination address, control plane packets originating from other proxies or resources, and packets from ASTAS IPv6 tunnels that are terminated in the router. Fig. 1 shows in which situation each type of packet can be expected. ASTAS-related data plane packets are forwarded to an IPv6 tunnel decapsulation module, before they arrive in the ASTAS core module.

3.1.2. *IPv6 tunnel decapsulation*

If an anycast data packet is encapsulated in another IP packet targeted at the proxy IP address, it is decapsulated by the IP6Decap module before it is pushed to the core component. Packets that are not encapsulated remain unchanged.

3.1.3. *ASTAS core module*

Fig. 3 depicts the flow diagram of a packet traversing this module. Upon packet arrival, its source and destination IP addresses and TCP/UDP ports are extracted from the IP and transport protocol headers. This four-tuple is the key used to lookup the session in the database of established sessions, analogous to a stateful firewall. If the packet matches a previously initiated session, the destination address for the IP encapsulation header is retrieved and the packet is encapsulated (if applicable). For a packet initiating a new session, a target proxy/resource is selected from the service repository and a new session is created.

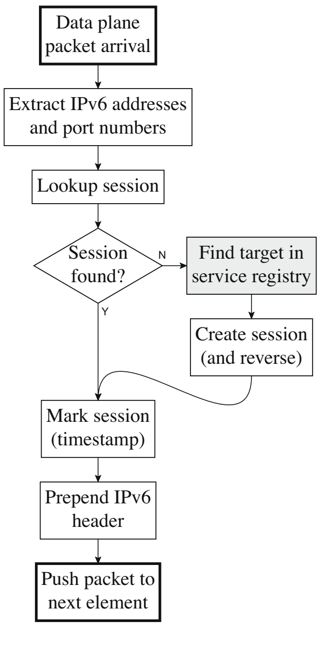

Data plane packet arrival
Extract IPv6 addresses and port numbers
Lookup session
Session found?
N
Y
Find target in service registry
Create session (and reverse)
Mark session (timestamp)
Prepend IPv6 header
Push packet to next element

Fig. 3. AstasCore element flow diagram.

If the packet was decapsulated previously by the IP6Decap module, the source address of that outer IP header is stored in the session record to track the return path. Furthermore, sessions that time out are removed from the system.

3.1.4. *Router design remarks*

The discussion in this section is entirely based on the Click router model and, consequently, the ASTAS extensions are introduced as Click components. Even though the resulting description is platform specific, we believe that the modular and generic approach of the Click model is well suited to provide a general overview of the changes necessary to make a regular router ASTAS compliant. Moreover, conceptual Click routers can be implemented and evaluated on regular Linux machines (both in user space and kernel space). As illustrated in Fig. 2, Click can also cooperate with dynamic routing protocols (e.g., OSPF, BGP) running on the host operating system to manage the data plane routing table. For this purpose, XORP [8] software can be used.

IPv6 is the network layer protocol of choice for the router extension design description and performance evaluation (see Section 3.2). There are no limitations that prevent ASTAS to be deployed in an IPv4 context, however. As such, the trends that will be shown in the performance evaluation part can be mapped directly to IPv4. Due to the manipulation of smaller addresses in the IPv4 case (32 bits instead of 128 for IPv6), the relative overhead of the ASTAS overlay will even decrease.

The duration of the target selection step in the ASTAS core element flow diagram (see shaded box in Fig. 3) is of crucial importance for the overall proxy session setup rate that can be achieved. Depending on the location of the target repository, this step might take just a few microseconds for an efficient local repository but several milliseconds for a remote repository. In this case, the control plane design (i.e., proxy state dissemination strategy) will have direct impact on data plane performance.

The ASTAS overlay relies on the use of IP tunnels to transport packets transparently to the anycast targets via the proxy system. This means that an IP header is prepended to each packet entering the overlay. In the IPv4 case, this results in an overhead of 20 bytes per-packet. Due to the larger header size, an IPv6 data plane increases the per-packet overhead to 40 bytes. For applications using relatively small packet sizes, this overhead is significant. For large packets, prepending the header should not lead to packet fragmentation (for reasons of performance). To overcome this issue, proxies can manipulate the IPv6 path MTU discovery process [9] and announce the IPv6 minimum path MTU of 1280 bytes [10]. This implies that underlying IPv6 links on paths between proxy pairs or proxy-resource paths must have an MTU of at least $1280 + 40 = 1320$ bytes.

3.2. *Performance evaluation*

In this section we wish to investigate the impact of the anycast overlay extensions on router forwarding performance. For this purpose, the Click router configuration de-

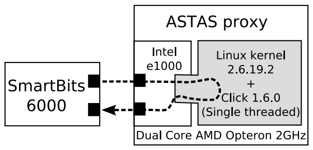

ASTAS proxy
Intel e1000
Linux kernel 2.6.19.2 + Click 1.6.0 (Single threaded)
SmartBits 6000
Dual Core AMD Opteron 2GHz

Fig. 4. Test setup and configuration.

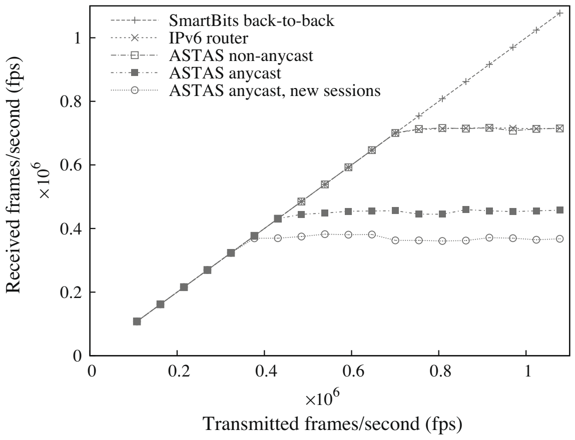


Fig. 5. ASTAS proxy throughput in Ethernet frames per second (fps).

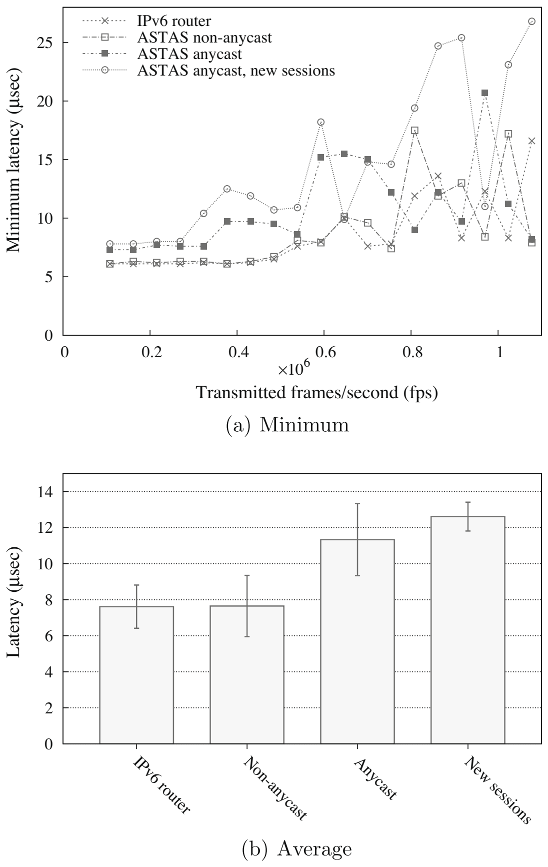


Fig. 6. ASTAS proxy latency measurements: (b) depicts the average latency at an Ethernet frame transmission rate of 215,000 fps. Error bars indicate plus and minus one standard deviation.

picted in Fig. 2 was deployed on PC hardware running Linux.

The test configuration is depicted in Fig. 4. The test machine, a dual core AMD Opteron running at 2 GHz, is equipped with a 32-bit 2.6 linux kernel patched for the Click router, that operates single-threaded in kernel space. An Intel e1000 interface card provides two gigabit Ethernet interfaces supporting the Click polling extensions.[1] As shown on the figure, the interfaces of the test machine—operating as an ASTAS proxy—are connected to a SmartBits 6000 [11] multi-port network performance analysis system.

Figs. 5 and 6 show the throughput and latency achieved with the test equipment described in the previous paragraph. The original packet size for the tests is 96 octets, which is the minimum packet size for Ethernet frames carrying TCP traffic over IPv6 in the SmartBits system.[2] Encapsulated packets—on the anycast return path—carry an extra IPv6 header and have a total frame size of 136 bytes. A total of four results is shown on the figure. SmartBits back-to-back throughput (direct connection between SmartBits interfaces) shows the maximum achievable performance by the performance analysis equipment. Subsequently, features are added to the test system in an incremental way to be able to assess a per-feature performance penalty. The results depicted in Figs. 5 and 6 lead to the following observations:

(1) The IPv6 router (Click configuration without ASTAS components) and ASTAS proxy have roughly the same performance in terms of system throughput and per-packet latency for forwarding unicast IPv6 packets. This means that introducing the extra classifier in the data plane does not significantly impact system performance for unicast traffic.
(2) Throughput decreases and packet forwarding latency increases significantly for anycast traffic, even for already established sessions. This indicates that extra processing power will be required in router devices to enable wire-speed anycast communications.
(3) When all packets belong to a new session, the session setup rate of the system is measured. Even for a local service registry (see Fig. 3), forwarding performance degrades significantly when compared to anycast throughput measurements from already established sessions. Nevertheless, a session setup rate close to 400,000 sessions per second is more than adequate for practical purposes. In case of a remote target service registry without local cache, the achievable session setup rate will be an order of magnitude smaller because it depends on network latency rather than the local lookup delay in

[1] Throughput increases significantly when interfaces operate in polling mode. At present, polling mode support is limited to a small number of interface cards.

[2] The SmartBits system adds a payload of 14 octets to the frame for packet identification and statistics.

the ASTAS router (as shown in Fig. 6b, this is just a few microseconds). Note that a remote target service registry (with or without local caching) may offer a balanced trade-off between maximum session setup rate and control plane scalability. This is discussed further in Section 4.

4. ASTAS control plane

4.1. Control plane design issues

When a client initiates a new session, the nearest anycast proxy captures the request and selects a target server if local resources are available or a remote proxy in the other case. In a perfect world, the proxy would have an accurate snapshot of every server's status in order to take well-informed decisions. Clearly, for any non-trivial deployment, this cannot be achieved in a scalable way. Remember that this is the main reason why server state is aggregated in the nearest anycast proxy. Even in this aggregated model, the dissemination of state information raises scalability concerns:

(i) Updating too frequently increases the proxy system and network load.
(ii) Updating infrequently causes inaccuracy.

Moreover, the inter-proxy state update strategy will impact scalability and possibly limit the total number of proxies in the system. Therefore, this section discusses the performance of *update triggering mechanisms* and *inter-proxy update strategies*. We consider the following performance criteria for the control plane:

(i) Network load generated by *control plane events*.
(ii) Accuracy of resource status information.

Note that the data plane performance resulting from control plane accuracy (i.e., the session assignment to a specific resource) is not taken into consideration for evaluating the control plane. That is, we study control plane overhead and accuracy in isolation, thereby neglecting their potential influence on data plane performance. In order to decouple session scheduling—a problem in its own right—from the control plane performance, we make the following assumption: *Resources are uniformly loaded, i.e., at all times the (distributed) scheduler realizes an even distribution of the workload over all resources, taking into account individual resource capacities.*

4.2. Update rate for resources and proxies

In this section, we compute the update frequency of a resource consisting of T slots to which threads can be allocated. The system sends an update when it reaches one of the $s+1$ predefined utilization states $u_i(0 \leqslant u_0 < u_1 < \cdots < u_s \leqslant T)$ coming from a neighboring update state u_j $(j \neq i)$. At the end of this section, we discuss *threshold-based updating*, a special case of this updating mechanism

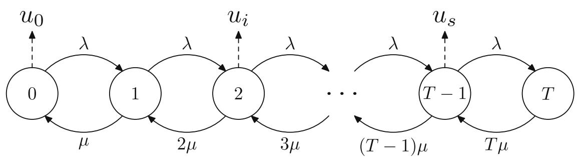


Fig. 7. Visual representation of the Markov chain defined in Eq. (1). u_0, u_i, and u_s depict possible update utilization states.

where an update is sent only if the system load differs from the load of the previous update point by the fixed *threshold*.

First, we present the mathematical modeling of a resource and derive the building blocks necessary to compute the global system update rate.

4.2.1. Mathematical model

We model the resource as a—well-known—M/M/T/T queuing system, i.e., the arrival rate for new sessions follows a Poisson distribution with parameter λ, session duration follows an exponential distribution with parameter μ, and a total of T slots for working threads are available in the system. A graphical representation of such a system, including the update states u_i, is depicted in Fig. 7. The session birth–death process for this system can then be modeled by a finite-state continuous time Markov chain $\{X(t)\}$ with infinitesimal generator Q, given by [12]:

$$Q = \begin{bmatrix} -\lambda & \lambda & 0 & & \cdots & 0 \\ \mu & -(\lambda+\mu) & \lambda & 0 & \cdots & 0 \\ 0 & 2\mu & -(\lambda+2\mu) & \lambda \quad 0 & \cdots & 0 \\ \vdots & & & \ddots & & \vdots \\ 0 & & \cdots & & T\mu & -T\mu \end{bmatrix}. \tag{1}$$

In matrix Q, a diagonal element Q_{ii} represents the parameter of the exponential staying time in state i. The transition probability from state i to $i+1$ is given by $\frac{\lambda}{\lambda+i\mu}$, and the transition probability from state i to $i-1$ is $\frac{i\mu}{\lambda+i\mu}$. The stationary distribution π of $\{X(t)\}$ is given by the left eigenvector of Q with eigenvalue 0 (Eq. (2)), satisfying the additional constraint in Eq. (3) [13]:

$$\pi^T Q = \mathbf{0}^T, \tag{2}$$

$$\sum_{i=0}^{T} \pi_i = 1. \tag{3}$$

If ρ is defined as follows:

$$\rho = \frac{\lambda}{\mu}, \tag{4}$$

then the vector components π_i of the stationary distribution $\boldsymbol{\pi}$ are given by the truncated Poisson distribution representing the Erlang loss formula for the M/M/T/T system:

$$\pi_i = \frac{\rho^i}{i! \sum_{k=0}^{T} \frac{\rho^k}{k!}}. \tag{5}$$

If the system would have triggered an update every time the system hits one of the states u_i, the results presented by Chi et al. [14] could have been employed to compute the update triggering probability, based on $\boldsymbol{\pi}$. Due to the extra condition that updates are only triggered if the previous update was sent from within another utilization state, this approach cannot be used, however. Building on the results of *conditional first-passage times* in general birth–death processes [15], our discussion below shows how the update rate for such a system can be computed.

First, we present random variables introduced by Jouini et al. in [15] that are associated with conditional first-passage times, and of great interest for the remainder of this discussion. Let θ_m^k be the first-passage time of the process $\{X(t)\}$ from state m to state $m-1$ given that the process does not visit state k $(1 \leqslant m < k)$ during this transition, i.e.,

$$\theta_m^k = Inf\{t > 0 : X(t) = m-1 | X(0) = m \text{ and no visit to } k\}. \tag{6}$$

Likewise, let τ_m^k be the first-passage time from state $m-1$ to state m given that the process does not visit k $(0 \leqslant k < m-1)$. Note that the states k and m for τ_m^k and θ_m^k are unrelated. τ_m^k is defined by:

$$\tau_m^k = Inf\{t > 0 : X(t) = m | X(0) = m-1 \text{ and no visit to } k\}. \tag{7}$$

Applying the expressions for the first order moment (i.e., the expected value) of θ_m^k and τ_m^k derived in [15] for general birth–death processes to the birth–death process $\{X(t)\}$ defined above, yields Eqs. (8) and (9):

$$\bar{\theta}_{k-1}^k = \frac{1}{\lambda + (k-1)\mu} \quad \text{and} \quad \bar{\theta}_m^k = \frac{\sum_{n=m}^{k-1} \chi_n^k}{\lambda \eta_m^k \chi_{m-1}^k}, \quad \text{for } (1 \leqslant m < k-1), \tag{8}$$

with

$$\begin{cases} \eta_{k-1}^k = \frac{(k-1)\mu}{\lambda + (k-1)\mu} \quad \text{and} \quad \eta_m^k = \frac{m\mu}{m\mu + \lambda(1 - \eta_{m+1}^k)}, \\ \chi_0^k = 1 \quad \text{and} \quad \chi_m^k = \lambda^m \prod_{n=1}^{m} \frac{\eta_n^k}{\delta_n^k}, \\ \delta_m^k = m\mu + \lambda(1 - \eta_{m+1}^k). \end{cases}$$

Similarly,

$$\bar{\tau}_m^k = \frac{\sum_{n=k+1}^{m-1} \phi_n^k}{\beta_{m-1}^k \phi_{m-1}^k} \quad \text{for } (m \geqslant k+2) \tag{9}$$

with

$$\begin{cases} \phi_{k+1}^k = 1 \quad \text{and} \quad \phi_m^k = \frac{(k+1)!}{m!\mu} \prod_{n=k+1}^{m-1} \frac{\beta_n^k}{v_{n+1}^k}, \\ \beta_{k+1}^k = \lambda + (k+1)\mu \quad \text{and} \quad \beta_m^k = \lambda + m\mu(1 - v_m^k), \\ v_{k+2}^k = \frac{\lambda}{\lambda + (k+1)\mu} \quad \text{and} \quad v_m^k = \frac{\lambda}{\lambda + (m-1)\mu(1 - v_{m-1}^k)}. \end{cases}$$

Now, based on the random variables θ_m^k we can introduce the random variables $L_{m \to l}^k$, representing the first-passage time from the transition starting in state m to state l, given no visit to state k $(0 \leqslant l < m < k)$. We define

$$L_{m \to l}^k = \sum_{n=l+1}^{m} \theta_n^k. \tag{10}$$

Similarly, let $R_{m \to r}^k$ be the random variables representing the first-passage time from state m to state r, given no visit to k $(0 \leqslant k < m < r)$. We have

$$R_{m \to r}^k = \sum_{n=m+1}^{r} \tau_n^k. \tag{11}$$

Determining the distribution function of the random variables $L_{m \to l}^k$ (respectively $R_{m \to r}^k$) involves a convolution of the random variables θ_n^k (respectively τ_n^k), but the expected values of the random variables $L_{m \to l}^k$ and $R_{m \to r}^k$ can be expressed as follows:

$$\bar{L}_{m \to l}^k = \sum_{n=l+1}^{m} \bar{\theta}_n^k, \tag{12}$$

and likewise,

$$\bar{R}_{m \to r}^k = \sum_{n=m+1}^{r} \bar{\tau}_n^k. \tag{13}$$

In the following section we apply these results to compute the update rate for *threshold-based updating*. Note that these random variables can also be used for computing the update rate for every possible combination of predefined utilization states u_i introduced at the start of this section.

4.2.2. Threshold-based updates

As a special case of the updating mechanism defined above, we consider threshold-based updating. The update mechanism is simple: whenever the load on the system expressed as $\frac{\text{active threads}}{T}$ varies by *threshold* $(0 < \textit{threshold} \leqslant 1)$, an update is sent. For a system consisting of a discrete number of thread slots T, the update states are spread evenly over the state space $[0, T]$. The distance σ between two subsequent update states is given by:

$$\sigma = \lceil \textit{threshold} \times T \rceil, \tag{14}$$

and the number of update states s not including state 0 equals

$$s = \left\lfloor \frac{T}{\sigma} \right\rfloor \tag{15}$$

When the system starts, it has 0 allocated thread slots. The first update is generated when the system reaches state σ. From that state, the next update will be sent either when the system reverts to its initial state 0 or when it pro-

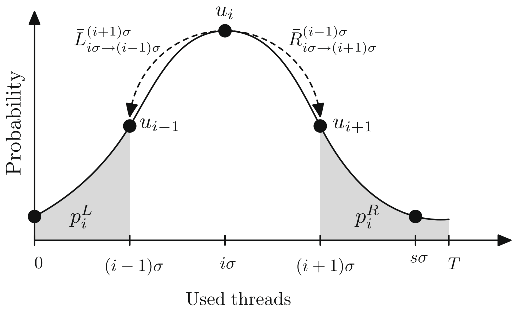


Fig. 8. Threshold-based updating illustrated.

gresses to state 2σ, and so on. Fig. 8 depicts the update states on the horizontal axis, related to their—assumed—limiting probabilities (from the stationary distribution) on the vertical axis. The last update state is $s\sigma \leqslant T$. Once the system visits update state $i\sigma$, the next update will be triggered when state $(i-1)\sigma$ or $(i+1)\sigma$ is visited. As shown in the figure, the average time needed to reach $(i-1)\sigma$ given no visit to $(i+1)\sigma$, is given by $\overline{L}^{(i+1)\sigma}_{i\sigma\to(i-1)\sigma}$. The average time to reach $(i+1)\sigma$ given no visit to $(i-1)\sigma$ is defined in a similar way. The probability p_i^L of moving to the left update state $(i-1)\sigma$ from $i\sigma$ is given by (see Fig. 8 for a visual interpretation):

$$p_i^L = \sum_{l=0}^{(i-1)\sigma} \pi_l.$$

Likewise, the probability p_i^R of moving to the right update state $(i+1)\sigma$ is

$$p_i^R = \sum_{r=(i+1)\sigma}^{T} \pi_r.$$

Consequently, in state $i\sigma$, the average update rate is

$$\frac{1}{p_i^L + p_i^R}\left(\frac{p_i^L}{\overline{L}^{(i+1)\sigma}_{i\sigma\to(i-1)\sigma}} + \frac{p_i^R}{\overline{R}^{(i-1)\sigma}_{i\sigma\to(i+1)\sigma}}\right).$$

As shown in Fig. 8, the probability of actually residing in state $i\sigma$ is given by the stationary distribution. Knowing that the system is continuously evolving from one update state to the next or previous update state (by construction), the average update rate for the entire system U_t can be computed by a normalized sum of the individual update state rates as follows:

$$U_t = \sum_{i=1}^{s-1} \frac{\pi_{i\sigma}}{(p_i^L + p_i^R)\sum_{j=0}^{s}\pi_{j\sigma}}\left(\frac{p_i^L}{\overline{L}^{(i+1)\sigma}_{i\sigma\to(i-1)\sigma}} + \frac{p_i^R}{\overline{R}^{(i-1)\sigma}_{i\sigma\to(i+1)\sigma}}\right) + \frac{\pi_0}{\sum_{j=0}^{s}\pi_{j\sigma}}\overline{R}^{-1}_{0\to\sigma} + \frac{\pi_{s\sigma}}{\sum_{j=0}^{s}\pi_{j\sigma}}\overline{L}^{-1}_{s\sigma\to(s-1)\sigma}. \quad (16)$$

In Eq. (16), the second and last term represent the unconditional update rate from state 0 to σ and from state $s\sigma$ to $(s-1)\sigma$, respectively. Expressions to compute the unconditional mean first-passage times can be found in [13,15]. The leading factor $\frac{\pi_{i\sigma}}{\sum_{j=0}^{s}\pi_{j\sigma}}$ for each term relates the probability $\pi_{i\sigma}$ of actually starting in state $i\sigma$ to the probabilities $\pi_{j\sigma}$ of residing in one of the other update states $j\sigma$.

Fig. 9 depicts the computed and simulated (i.e., measured) update rate for an increasing update threshold in a system with $T = 20,\ \lambda = 2,\ \mu = \frac{2}{15}$. As expected, increasing the update threshold decreases the update rate. Since the average system accuracy is roughly equal to $\frac{threshold}{2}$, this reduction of the update rate can only be achieved at the expense of decreasing system accuracy. The irregularity (both measured and computed) in the graph at $threshold = 0.35$ can be explained as follows. For $threshold = 0.3$, the update states are $\{0,6,12,18\}$. For $threshold = 0.4$, the update states are $\{0,8,16\}$. Since the update states for $threshold = 0.35$ are $\{0,7,14\}$, and the average system load $\rho = 15$, reaching an update event is more likely for both thresholds surrounding 0.35.

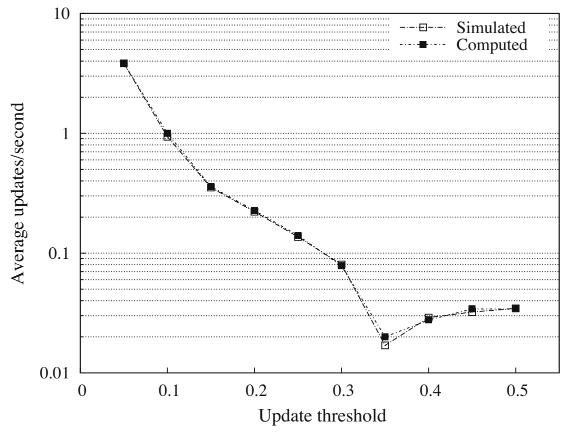


Fig. 9. Update rate related to the update threshold.

The slight discrepancy between the simulated and computed update rate originates from the use of an infinite-state Markov chain for the computation of the conditional first-passage times in [15]. Instead of the finite-state birth–death process defined by the generator matrix Q, Jouini et al. consider a general birth–death process with $\lambda_S > 0$ for state $S \geqslant T$. Fig. 10 illustrates this discrepancy for $T = 20,\ \lambda = 2,\ \mu = \frac{2}{15}$. This minimal error is acceptable for stable systems, i.e., systems that are not overloaded.

Computing the update rate for proxies using a threshold-based updating mechanism is not straightforward, because their rate of change depends on the attached resources and their update threshold. Nevertheless, Eq. (16) can be used to evaluate the update rate for an increasing number of thread slots in the system. Fig. 11 depicts these results. Note that $\mu = \frac{2}{15}$ is fixed, while the arrival

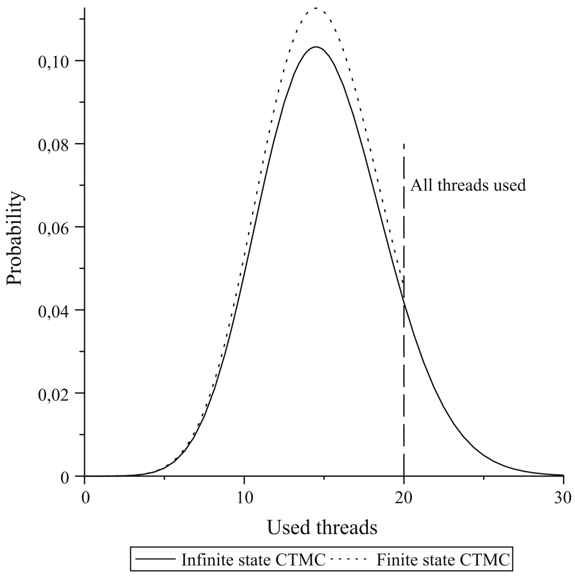


Fig. 10. The discrepancy between the stationary distributions for the finite and infinite CTMC is small for stable systems $\left(\frac{\lambda}{\mu T} \ll 1\right)$. A comparison is depicted for $\lambda = 2,\ \mu = \frac{2}{15},\ T = 20$.

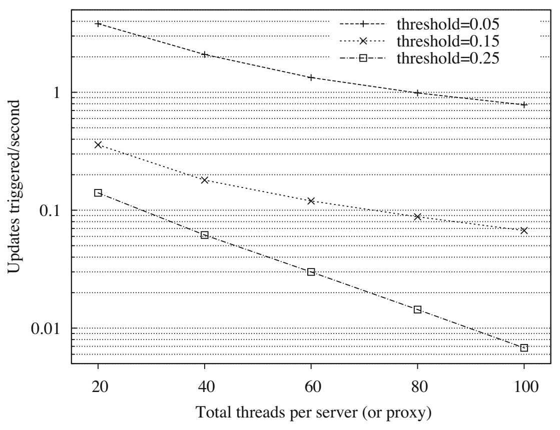


Fig. 11. By allowing more threads per proxy (i.e., aggregating more), the update frequency can be reduced without sacrificing accuracy.

rate λ is adjusted to the number of thread slots to maintain a system load of 75% as the number of thread slots increases, i.e., $\lambda = 0.75\mu T$. Since proxies aggregate the state of all connected resources, they can be modeled as super-resources with a number of thread slots equal to the sum of thread slots of all individual resources. The results clearly indicate that the aggregation phase by the proxies increases scalability, by decreasing the update rate without loss of accuracy.

4.3. Inter-proxy update strategy

If the aggregated state of the resources for a specific service behind a proxy changes significantly (see previous section), the proxy decides to update the other proxies. In general, this can be achieved by disseminating the update information to *all* proxies or to a *subset* of proxies (possibly a single proxy) or dedicated repositories. In the first case, all proxies have an accurate view on the state of all participating proxies, while in the latter case proxies have to query the (quasi-)centralized repository in order to get updated. In this paper, we consider four updating strategies: broadcasting, flooding, a single centralized repository, and a distributed hash table (DHT) repository.

Using a *broadcasting* strategy is the fastest way to diffuse an update to all other proxies. Update packets are sent individually to all recipient proxies and are forwarded over the shortest path. Unfortunately, the sending proxy needs to be aware of all other (recipient) proxies, which is unpractical or even impossible for large deployments.

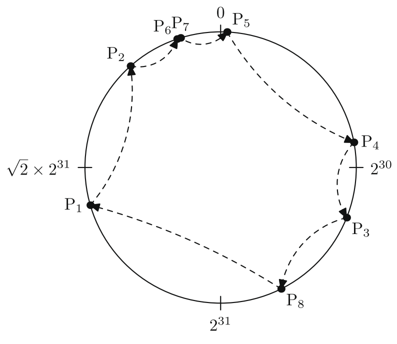
Flooding is also a diffusion strategy. Contrary to the broadcasting approach, only neighbor proxies are addressed directly. Upon arrival of an update, a neighbor in turn forwards the update to its neighbors. Duplicate updates are discarded. Note that the neighbor-relationship is defined at the application level and is not necessarily related to physical proximity.

Updating a *central repository* implies that a single node is responsible for maintaining the proxies' state. Upon request or on a timely basis, the central repository informs the requester or all remote proxies about the other proxies' state. By distributing the repository over multiple nodes (or even all proxies), a single point of failure and performance bottleneck can be avoided, albeit at the expense of configuration manageability and operational simplicity. A natural solution to overcome these shortcomings is provided by *distributed hash tables* [16,17].

The general purpose of a DHT is to map a *key* to a specific node. In the context of this paper, an anycast IP address—possibly augmented with the TCP destination port—uniquely identifies an anycast service and can be used as a lookup key. As such, each node participating in the DHT becomes responsible for maintaining the state of a subset of anycast services for all proxies. For the remainder of this paper, a simplified version of the Chord DHT [17] is used. To concretize and clarify the use of a DHT for the distributed anycast service state management, we first recapitulate basic Chord working principles.

As shown in Fig. 12, Chord maps all nodes and lookup keys on a circular key space. For the simplified version used here, each node and service identifier are mapped to a 32-bit hash value. By design, each Chord node is responsible for the keys with a hash value between that of its predecessor on the ring and itself. Assuming that each node knows its successor, a lookup can be forwarded over the ring until the responsible node for that key is reached. To speedup this linear lookup process, Chord provides a finger table in each node to be able to bypass most nodes on the ring. Considering the 32-bit version used in this paper, a node with hash value n has 32 finger table entries: $\text{finger}[k] = (n + 2^{k-1}) \bmod 2^{32},\ 1 \leqslant k \leqslant 32$. Then, for each entry finger[k], the node closest to but smaller than $(n + 2^{k-1}) \bmod 2^{32}$ is stored to allow direct jumps to this node. For an in-depth discussion of Chord, including a discussion on its operation under churn, we refer to [17].

In the following sections we discuss the performance of each update strategy in terms of their generated network load. First, analytical calculations are shown for rings and trees, next simulation results for more complex—but realistic—operator networks are presented.



Proxy ID	32-bit hash
1	3028713910
2	3798448204
3	1332612764
4	946348061
5	32663278
6	4071828276
7	4090419040
8	1831203572

Fig. 12. Chord example.

4.4. Update strategy performance in rings and binary trees

Due to their regular structure, ring and tree networks facilitate direct comparison between the different update strategies.

In a ring with $n = 2^d$ nodes and $p = 2^i$ $(1 \leqslant i \leqslant d)$ proxies *spread evenly* over all ring nodes, the average distance between the proxies is given by

$$\overline{D}_{ring} = \frac{n \cdot p}{4 \cdot (p-1)}. \qquad (17)$$

Using broadcast, all p proxies update $p-1$ other proxies directly. The corresponding average link load, expressed as $\frac{\text{update messages}}{\text{link}}$, is

$$\overline{B}_{ring} = \frac{p \cdot (p-1) \cdot \overline{D}_{ring}}{n} = \frac{p^2}{4}. \qquad (18)$$

For the flooding scenario, we assume that each updating proxy sends an update to its left and right neighbor on the ring. Since duplicate updates are discarded, both updates will visit half of the ring nodes, which results in the following average link load:

$$\overline{F}_{ring} = \frac{2 \cdot p \cdot \frac{n}{2}}{n} = p. \qquad (19)$$

In the case of a single proxy operating as the central repository for the status information related to an anycast service, there are $p-1$ proxies updating this central node. This translates to the following average link load:

$$\overline{Ce}_{ring} = \frac{(p-1) \cdot \overline{D}_{ring}}{n} = \frac{1}{4} \cdot p. \qquad (20)$$

When a Chord DHT is used to increase manageability and achieve load balancing as the number of services increases, the average link load increases because there is no relationship between the physical proxy location and its place on the Chord ring. This is depicted in Fig. 13, where dotted arrows show the Chord ring of Fig. 12 mapped to the physical ring topology. Additionally, a Chord update requires about $\frac{1}{2} \cdot \log_2(p)$ steps on the Chord ring to reach its destination [17]. The average link load is then given by

$$\overline{Ch}_{ring} = \frac{(p-1) \cdot \frac{1}{2} \cdot \log_2(p) \cdot \overline{D}_{ring}}{n} = \frac{1}{8} \cdot p \cdot \log_2(p). \qquad (21)$$

For the binary tree network with $n = 2^{d-1}$ leaf nodes and $p = 2^i$ $(1 \leqslant i \leqslant d-1)$ proxies spread evenly over the leaf nodes, average link load can be computed in a similar way. The average distance between proxies is given by

$$\overline{D}_{tree} = \sum_{i=1}^{d-1} \frac{2 \cdot \lfloor p \cdot 2^{-i} \rfloor \cdot (d-i)}{p-1}, \qquad (22)$$

which yields the following average link load for broadcast:

$$\overline{B}_{tree} = \frac{p \cdot (p-1) \cdot \overline{D}_{tree}}{2^d - 2}. \qquad (23)$$

For the flooding scenario, neighbors are selected as shown in Fig. 14. The average link load is given by

$$\overline{F}_{tree} = \frac{p \cdot \sum_{i=1}^{\log_2 p} 2^i \cdot (d-i)}{2^d - 2}. \qquad (24)$$

For the central repository, the average link load is

$$\overline{Ce}_{tree} = \frac{(p-1) \cdot \overline{D}_{tree}}{2^d - 2}, \qquad (25)$$

and similarly, the Chord average link load is given by

$$\overline{Ch}_{tree} = \frac{(p-1) \cdot \frac{1}{2} \cdot \log_2(p) \cdot \overline{D}_{tree}}{2^d - 2}. \qquad (26)$$

In Figs. 15 and 16 the average link load is related to the number of proxies in the ring or tree, respectively. From

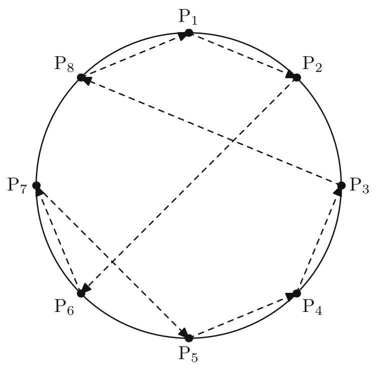


Fig. 13. Chord ring (dotted line) drawn on top of the physical ring topology.

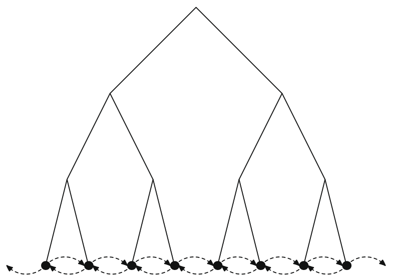


Fig. 14. Binary tree with depth $d = 4$ and $2^{d-1} = 8$ leaf nodes. Dotted arrows show the circular flooding path.

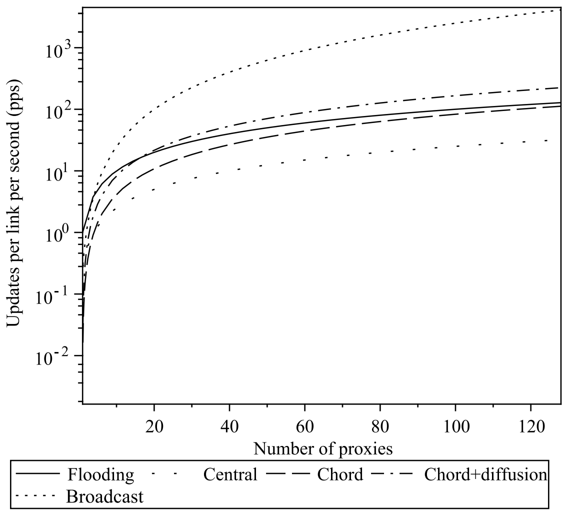


Fig. 15. Link load in a 128-node ring.

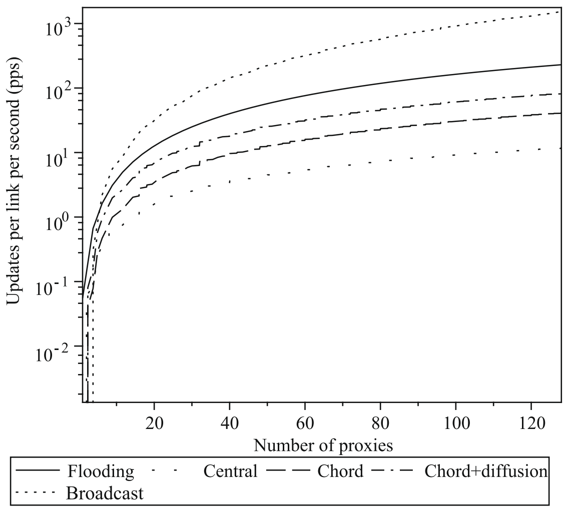


Fig. 16. Link load in a tree with depth 7 and 128 leaf nodes (potential proxies).

a network perspective, the centralized repository is the most efficient solution for both the ring and tree network. This is not a surprise, since the many-to-one relationship implies linear scaling in the number of proxies for the average network load. As such, it can be seen as an lower bound for the Chord updating cost, where all updates for the same anycast service eventually arrive in the single node responsible for the service key. As calculated by Eqs. (18) and (23) the network load generated by the broadcast strategy scales quadratically in the number of proxies. Due to its poor scaling in terms of network load and manageability, broadcasting is not further considered for the operator network simulations in the next section.

For the ring network (see Fig. 15), flooding is more efficient than Chord updating for a large number of proxies, especially when we assume that Chord also diffuses the aggregated status information at the same rate to the other proxies (Chord + diffusion). This is remarkable, because in Chord only a single node is updated (many-to-one) and for flooding all nodes are updated (many-to-many) and no further diffusion is necessary to get the status information in the proxies where the tunnels are set up. There are two reasons why flooding is more efficient here: (i) the average distance between two arbitrary nodes is relatively large (scales linearly in the number of nodes, see Eq. (17)), and (ii) the distance between two flooding neighbors is constant (and minimal). In a tree, these conditions are not fulfilled and Chord performs better than flooding, even if diffusion to all other proxies over the Chord ring at the same rate is taken into account (see Fig. 16). In the next section, simulation results for cable and xDSL operator networks—that often consist of a combination of rings and (aggregation) trees—are discussed.

4.5. Simulations for operator networks

For next-generation access and aggregation networks, operators and telecommunications equipment vendors investigate the possibility to shift IP and/or service awareness from the edge of the access network closer to the end-user [18]. In such an access network, operators could implement value-added services to generate more revenue or to differentiate from their competitors. When the ASTAS overlay is deployed in an IP-aware operator network, it could be used in a peer-to-peer mode where subscribers offer services to other subscribers lacking the CPU-power or other resources to accomplish certain tasks. With a growing number of resource-constrained portable devices, such a scenario is not unthinkable.

In this section, two typical access and aggregation network topologies are selected as use cases: a mesh of trees topology, primarily used in xDSL deployment, and a ring of rings topology, used in cable Internet access deployment, both consisting of 280 nodes. In both topologies, all nodes are assumed to be IP routers of a service-aware access and aggregation network and thus candidate for an anycast proxy upgrade.

The mesh of trees topology is depicted in Fig. 17. The 250 access nodes (L_2) are located at the leaf nodes of five trees of depth two, aggregated per 10 at depth two and per five at depth one. The five trees are interconnected by a full mesh between the root nodes. The ring of rings topology is depicted in Fig. 18. It consists of five unidirectional six-node secondary rings. One node connects the secondary ring to a unidirectional primary ring. The other five secondary ring nodes each connect to ten access nodes in a logical star topology. For the simulations, we assume that 100 Mb/s is available for anycast traffic on each link and two resources are attached to each access node. The simulation parameters are summarized in Table 1.

Simulations in this section are implemented using the NS-2 simulator [19] augmented with NS-MIRACLE [20] multi-interface extensions. Two new application modules implement anycast resource and proxy functionality. Furthermore, the assumption introduced in Section 4.1 is also valid for the NS-2 simulations: the active threads in each resource are modeled independently (for each resource) as a birth–death process with exponential inter-arrival times (parameter λ) and exponential duration (parameter μ).

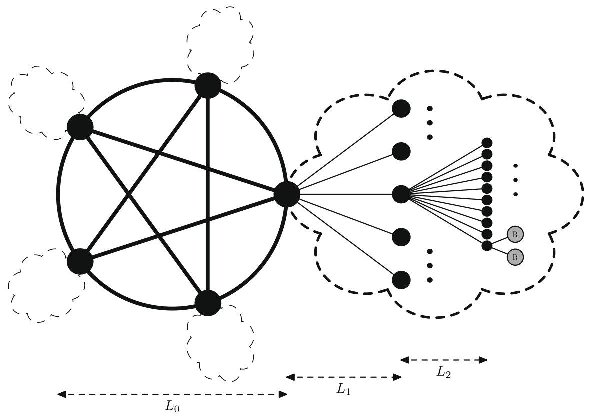


Fig. 17. Simulation topology for xDSL networks.

Table 1
Simulation setup.

Parameter	Value
Link delay	0.1 ms
Link bandwidth	100 Mb/s
Network interface buffer size	100 packets
Control plane packet size	64B
New sessions per resource per second	Exp($\lambda = 2$)
Session duration (s)	Exp($\mu = \frac{2}{15}$)
Thread slots per resource	20
Resources	$2 \times 250 = 500$
Update threshold	0.15

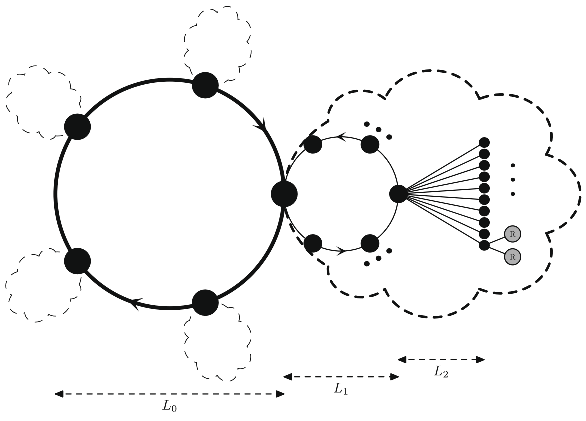


Fig. 18. Simulation topology for cable networks.

The simulation results for both the xDSL and cable like use case for a single anycast service are depicted in Figs. 19 and 20, respectively. Results are depicted for proxies placed at the three levels defined in Figs. 17 and 18, i.e., L_0, L_1, and L_2. Then, for each proxy deployment scenario, the global average link load and the average link load at each level are measured. The results lead to the following conclusions:

(1) Proxy placement (at L_0, L_1, or L_2) has greater impact on the control plane overhead than the update strategy for both xDSL and cable aggregation topologies. Placing proxies at L_0 implies that—frequent—individual resource updates travel over L_1 and L_2 links, while placing proxies at L_2 causes many inter-proxy updates due to the small number of resources (2) attached to each proxy (see Section 4.2.2, Fig. 11).
(2) In the xDSL aggregation network, the average distance between nodes is relatively small, meaning that Chord outperforms flooding for proxies deployed at all layers. Nevertheless, the difference in performance between Chord and the central repository update strategy for proxy deployment at L_1 and L_2 gives a clear indication of the overhead generated by traversing the Chord ring.
(3) In the cable aggregation network, the—unidirectional—rings increase the average distance between nodes. When proxies are placed at L_2 nodes, this translates in significant Chord overhead on the L_0 links. In this case, Chord is even outperformed by flooding when considering L_0 links only. When considering the fact that Chord still needs to diffuse the gathered information in a second phase, flooding is

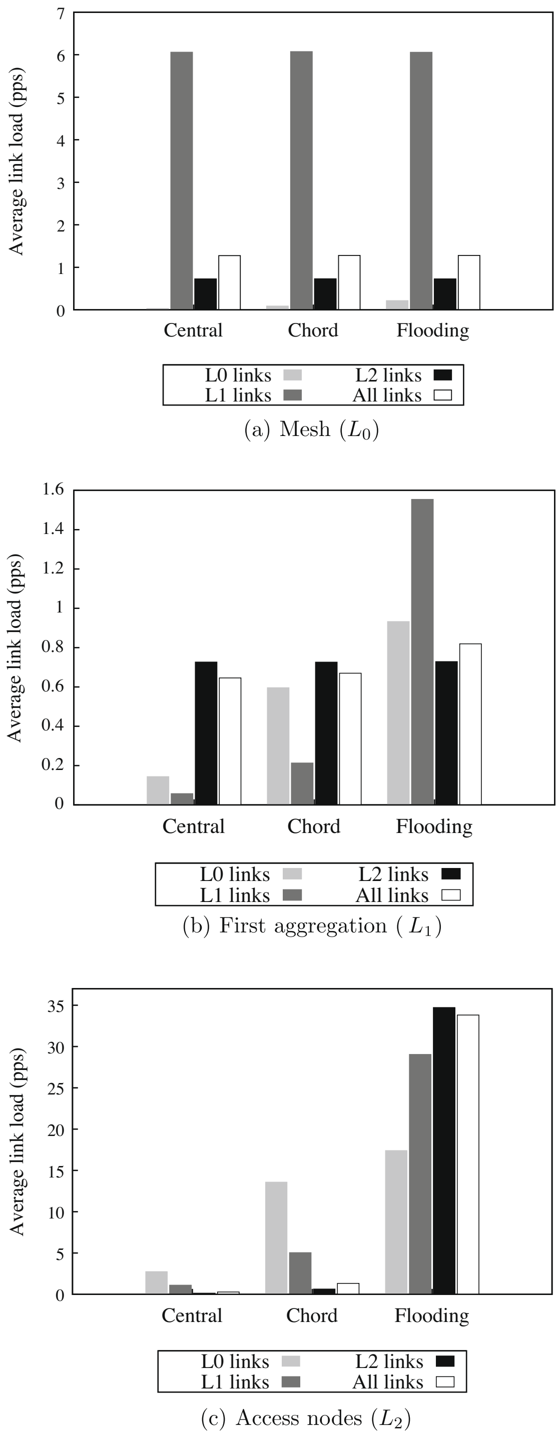


Fig. 19. Average link load for xDSL-like aggregation networks.

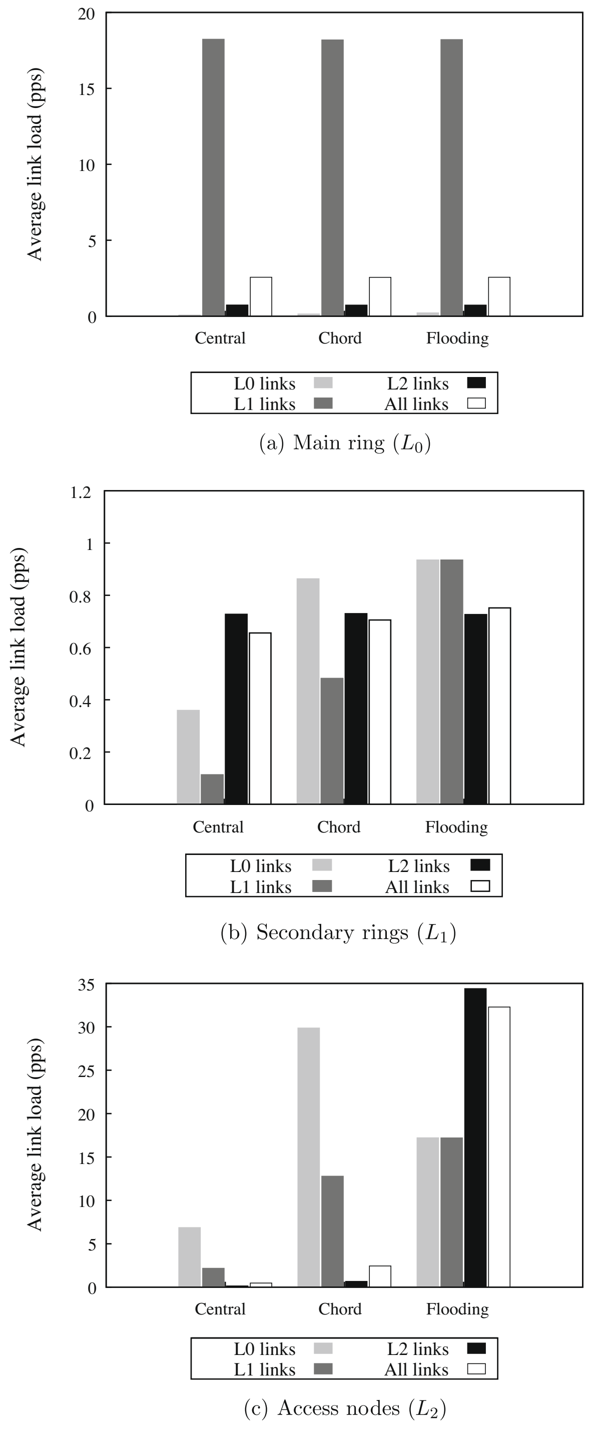


Fig. 20. Average link load for cable-like aggregation networks.

likely to be the optimal strategy for proxies placed at L_1. These results match the results of Section 4.4 for rings.

(4) The results depicted in the figures suggest that updating a central repository is the most efficient strategy. For an increasing number of services and a single repository, this approach scales poorly, however. Additionally, assigning a central repository by hand for each new service to overcome scalability issues quickly becomes unmanageable. Distributed hash tables—like Chord—solve these issues elegantly at the expense of introducing little extra network load, provided that the average distance between two network nodes is relatively small. Fortunately, even for large realistic networks like the Internet—often modeled by scale-free graphs—this condition holds true [21].

5. Conclusion

First of all, this paper studied the data plane scalability of the anycast overlay concept ASTAS. Experimental validation of an ASTAS implementation for the Click extensible router platform showed that an extra classifier to identify anycast packets did not significantly decrease the unicast packet forwarding rate of the router. Furthermore, a packet forwarding rate close to 65% of the maximum unicast forwarding rate could be maintained for anycast traffic without extra hardware acceleration.

The second part of this paper presented an analysis of two orthogonal problems related to the ASTAS overlay control plane scalability. The first analysis related the resource and proxy update rate to the desired system accuracy. The proposed mathematical model clearly indicated the gain in scalability resulting from threshold-based updating and resource state aggregation in the overlay proxies. The second analysis investigated several options for the inter-proxy update strategy. In general, we have shown that a repository based on distributed hash tables realized control plane scalability in terms of the number of proxies and services, although other update strategies such as neighbor flooding may generate less overhead for specific network conditions or may be more appropriate to fulfill alternate operator optimization criteria (e.g., minimize load on specific links). Finally, simulation results have shown that proxy placement is of greater importance for control plane scalability than the actual update strategy for ASTAS deployment in xDSL and cable operator networks.

Acknowledgments

The work presented in this paper is supported by the EU through the IST Project Phosphorus (http://www.ist-phosphorus.eu). The Phosphorus project is funded by the European Commission under the FP6 Contract No. 034115.

Tim Wauters, Chris Develder, and Filip De Turck acknowledge the F.W.O.-V. (Fund for Scientific Research Flanders) for their support through a postdoctoral fellowship.

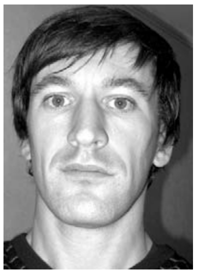
References

[1] C. Partridge, T. Mendez, W. Milliken, RFC 1546: Host Anycasting Service, November 1993.
[2] S. Sarat, V. Pappas, A. Terzis, On the use of anycast in DNS, SIGMETRICS Performance Evaluation Review 33 (1) (2005) 394–395.

[3] D. Katabi, J. Wroclawski, A framework for scalable Global IP-Anycast (GIA), ACM SIGCOMM Computer Communication Review 30 (4) (2000) 3–15.
[4] H. Ballani, P. Francis, Towards a Global IP Anycast Service, ACM SIGCOMM Computer Communication Review 35 (4) (2005) 301–312.
[5] T. Stevens, M. De Leenheer, C. Develder, F. De Turck, B. Dhoedt, P. Demeester, ASTAS: architecture for scalable and transparent anycast services, Journal of Communications and Networks 9 (4) (2007) 457–465.
[6] E. Kohler, The click modular router, Ph.D. Thesis, Massachusetts Institute of Technology, Co-Supervisor-Robert Morris and Co-Supervisor-M. Frans Kaashoek, 2001.
[7] C. Perkins, RFC 2003: IP encapsulation within IP, October 1996.
[8] XORP, 2008. <http://www.xorp.org/>.
[9] J. McCann, S. Deering, J. Mogul, RFC 1981: path MTU discovery for IP version 6, August 1996.
[10] S. Deering, R. Hinden, RFC 2460: internet protocol, version 6 (IPv6), December 1998.
[11] Spirent Communications, 2008. <http://www.spirent.com/>.
[12] L. Kleinrock, Queueing Systems, Theory, vol. 1, Wiley-Interscience, 1975.
[13] D. Yao, First-passage-time moments of Markov processes, Journal of Applied Probability 22 (1985) 939–945.
[14] C. Chi, D. Huang, D. Lee, X. Sun, Lazy flooding: a new technique for information dissemination in distributed network systems, IEEE/ACM Transactions on Networking 15 (1) (2007) 80–92.
[15] O. Jouini, Y. Dallery, Moments of first passage times in general birth–death processes, Mathematical Methods of Operations Research, 2007 (published online).
[16] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Schenker, A scalable content-addressable network, SIGCOMM Computer Communication Review 31 (4) (2001) 161–172.
[17] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. Kaashoek, F. Dabek, H. Balakrishnan, Chord: a scalable peer-to-peer lookup protocol for internet applications, IEEE/ACM Transactions on Networking 11 (1) (2003) 17–32.
[18] Multi Service Access Everywhere, 2008. <http://www.ist-muse.eu/>.
[19] The Network Simulator – ns2. <http://www.isi.edu/nsnam/ns/>.
[20] N. Baldo, F. Maguolo, M. Miozzoy, M. Rossi, M. Zorzi, ns2-MIRACLE: a modular framework for multi-technology and cross-layer support in network simulator 2, in: Proceedings of ACM NSTools 2007, Nantes, France, 2007.
[21] R. Albert, A. Barabási, Statistical mechanics of complex networks, Reviews of Modern Physics 74 (1) (2002) 47–97.

Tim Stevens received his M.Sc. degree in Computer Science from Ghent University, Belgium, in June 2001. In January 2009, he obtained his Ph.D. degree in Computer Science from the same university. Before joining the Department of Information Technology (INTEC) of Ghent University in August 2003, Tim Stevens was a database and system administrator for the Flemish public broadcasting company (VRT). At present, he is a postdoctoral researcher affiliated with INTEC. His main research interests include future access network architectures, IPv6, Quality of Service (QoS) and traffic engineering in IP networks, and anycast-based services..

Tim Wauters received his M.Sc. degree in electrotechnical engineering in 2001 from the University of Ghent, Belgium. Since September 2001, he has been working on the design of content distribution and peer-to-peer networks in the Department of Information Technology (INTEC), at the same university. His research led to a Ph.D degree in January 2007. His work has been published in more than 25 scientific publications in international conferences and journals.



Chris Develder received the M.Sc. degree in computer science engineering and a Ph.D. in electrical engineering from Ghent University (Ghent, Belgium), in July 1999 and December 2003, respectively. From October 1999 on, he has been working in the Department of Information Technology (INTEC), at the same university, as a Researcher for the Research Foundation – Flanders (FWO), in the field of network design and planning, mainly focusing on optical packet switched networks. In January 2004, he left University to join OPNET Technologies, working on transport network design and planning. In September 2005, he re-joined INTEC at Ghent University as a post-doctoral researcher, and as a post-doctoral fellow of the FWO since October 2006. Since October 2007 he holds a part-time professor position at the same institute. He was and is involved in multiple national and European research projects (IST Lion, IST David, IST Stolas, IST Phosphorus, IST E-Photon One). His current research focuses on dimensioning, modeling and optimising optical Grid networks and their control and management. He is an author or co-author of over 45 international publications.

Filip De Turck received his M.Sc. degree in Electronic Engineering from the Ghent University, Belgium, in June 1997. In May 2002, he obtained the Ph.D. degree in Electronic Engineering from the same university. At the moment, he is a part-time professor and a post-doctoral fellow of the F.W.O.-V., affiliated with the Department of Information Technology of the Ghent University. He is author or co-author of approximately 135 papers published in international journals or in the proceedings of international conferences. His main research interests include scalable software architectures for telecommunication network and service management, performance evaluation and design of new telecommunication services. He is in the program committee of several conferences and regular reviewer for conferences and journals in this field.

Bart Dhoedt received a degree in Engineering from the Ghent University in 1990. In September 1990, he joined the Department of Information Technology of the Faculty of Applied Sciences, University of Ghent. His research, addressing the use of micro-optics to realize parallel free space optical interconnects, resulted in a PhD degree in 1995. After a two-year post-doc in opto-electronics, he became professor at the Faculty of Applied Sciences, Department of Information Technology. Since then, he is responsible for several courses on algorithms, programming and software development. His research interests are software engineering and mobile and wireless communications.

He is author or co-author of approximately 150 papers published in international journals or in the proceedings of international conferences. His current research addresses software technologies for communication networks, peer-to-peer networks, mobile networks and active networks.

Piet Demeester finished his Ph.D. Thesis at INTEC, Ghent University, in 1988. At the same department he became group leader of the activities on metal organic vapor phase epitaxial growth for optoelectronic components. In 1992 he started a new research group on broadband communication networks. The research in this field has already resulted in more than 300 publications. In this research domain he was and is a member of several program committees for international conferences, such as ICCCN, the International Conference on Telecommunication Systems, OFC, ICC, and ECOC. He was Chairman of DRCN '98. He was chairman of the Technical Programme Committee ECOC '01. He has been Guest Editor of three special issues of IEEE Communications Magazine. He is also a member of the Editorial Board of Optical Networks Magazine and Photonic Network Communications. He was a member of several national and international Ph.D. Thesis commissions. He is a member of ACM and KVIV. His current research interests include multilayer networks, QoS in IP networks, mobile networks, access networks, grid computing, distributed software, network and service management, and applications (supported by FWO-Vlaanderen, the BOF of Ghent University, the IWT, and the European Commission). He is currently a full-time professor at Ghent University, where he is teaching courses in communication networks. He has also been teaching different international courses.