UNIVERSITÉ DE MONTRÉAL

PROTECTION CONTRE LES ATTAQUES DE DÉNI DE SERVICE PAR GESTION DYNAMIQUE DU DÉLAI D'INACTIVITÉ

DANIEL NICOLAE BOTEANU
DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION DU DIPLÔME DE
MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)
MAI 2008

Your file Votre référence
ISBN: 978-0-494-46035-1
Our file Notre référence
ISBN: 978-0-494-46035-1

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

PROTECTION CONTRE LES ATTAQUES DE DÉNI DE SERVICE PAR GESTION DYNAMIQUE DU DÉLAI D'INACTIVITÉ

présenté par : BOTEANU Daniel Nicolae
en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées
a été dûment accepté par le jury constitué de :

M. GAGNON Michel, ing., Ph.D., président
M. FERNANDEZ José Manuel, ing., Ph.D., membre et directeur de recherche
Mme. BELLAÏCHE Martine, Ph.D., membre

À mon amour, Alexandra

REMERCIEMENTS

Plus qu'à tout le monde, je tiens à remercier José Fernandez, mon directeur, pour tout le dévouement dont il m'a fait preuve à plusieurs instances, pour ses conseils autant académiques que personnels, pour les nombreuses rencontres en dehors des heures de travail et plus généralement, pour l'initiation dans le monde de la recherche.

Les coauteurs des articles précédents, Édouard Reich, John Mullins et John McHugh, méritent aussi ma gratitude pour leurs idées et leur aide qui m'ont aidé avancer plus rapidement dans ce projet.

Finalement, je remercie mes parents, Maria et Nicolae-Daniel, pour m'avoir envoyé sur ce chemin. Sans leur aide spirituelle et matérielle, je n'aurai jamais pu vivre cette aventure d'études supérieures à l'étranger.

RÉSUMÉ

Le but d'une attaque de déni de service (DoS) est de rendre un service réseau indisponible pour les usagers légitimes. Nous adressons le problème des attaques de déni de service sur les protocoles orientés connexion où l'attaquant essaie d'épuiser les connexions du serveur en initiant la communication avec le server et en l'abandonnant par la suite. Ainsi, les utilisateurs légitimes ne peuvent plus initier des nouvelles connexions avec le système. L'attaque la plus exploitée de cette catégorie est l'attaque *SYN-Flood* mais d'autres attaques qui utilisent la même approche dans des protocoles à état rentrent dans la même catégorie. La stratégie la plus simple et évidente pour se protéger contre ce type de comportement malicieux est d'avoir un mécanisme de délai d'inactivité. La méthode traditionnelle est d'utiliser un délai d'inactivité fixe, mais l'intuition est que l'utilisation d'un délai d'inactivité dynamique offre une meilleure performance. Ceci est notre hypothèse de base et la vérifier a été l'inspiration de ce travail. Nos buts sont d'une part de développer un modèle mathématique pour pouvoir analyser le compromis entre les ressources de l'attaquant et du défenseur et d'autre part d'offrir des mécanismes de prévention qui peuvent être utilisés contre des attaques dans cette catégorie. Nous modélisons la file des connexions du serveur en utilisant des chaines de Markov pour établir une relation entre la capacité du serveur, le taux d'attaque et l'impact sur le niveau de service. Nous analysons deux méthodes d'ajustement du délai d'inactivité, *threshold* et *linear*, et nous couplons ces méthodes avec trois politiques d'assignation du délai d'inactivité aux connexions : la politique *deterministic*, la politique *deferred* et la politique utopique *Poisson*.

Les résultats que nous avons obtenus confirment nos intuitions. Premièrement, le modèle théorique montre que pour toutes les stratégies, il existe un compromis linéaire entre le taux d'attaque et la taille de la file du serveur cible, dans le sens où lorsque le premier augmente, le deuxième doit être augmenté aussi pour maintenir la même qualité de service pour les utilisateurs légitimes. Cependant, le rapport qui doit être gardé entre ces valeurs diffère entre les stratégies; dans ce sens, certaines sont meilleurs que d'autres. Plus particulièrement, le modèle théorique indique aussi que la stratégie du délai d'inactivité dynamique *linear deferred* est très similaire du point de vue de la performance à la stratégie *linear Poisson*, qui à son tour surpasse toutes les autres stratégies du délai d'inactivité dynamique. Les stratégies du délai d'inactivité dynamique surpassent toujours la méthode classique du délai d'inactivité *fixed*.

Notre modèle est très général et peut être utilisé pour décrire le comportement de la file du serveur durant des attaques d'épuisement des connections à des niveaux différents de la pile des protocoles TCP/IP. Nous confirmons ces résultats par des simulations stochastiques et des expériences réseau des attaques *SYN-Flood*. Nous montrons aussi comment le modèle peut être utilisé pour analyser une attaque d'inondation avec connections TCP ou une attaque d'inondation avec des réservations des billets. Les stratégies que nous suggérons sont robustes faces à des changements dans le modèle d'attaque et notre implémentation est très efficace et transparente par rapport au serveur et aux applications qu'elle essaie de protéger. Les stratégies pourront donc être facilement intégrées dans des systèmes d'exploitation et des applications, ou être implémentées dans des équipements réseau.

ABSTRACT

The purpose of a denial-of-service (DoS) attack is to render a network service unavailable for legitimate users. We address the problem of DoS attacks on connection oriented protocols where the attacker tries to deplete the server connections by initiating communication with the server and then abandoning the communication. Thus, legitimate users can no longer initiate new connections with the system. The most exploited attack in this category is the *SYN-flood* attack but other attacks using the same approach in stateful communication protocols also fall into this category. The simplest and most obvious strategy to protect against this type of malicious behaviour is to have a timeout mechanism in place. The traditional method is to use a fixed timeout but the intuition is that using a dynamic timeout offers better performance. This is our base hypothesis and verifying it was the inspiration to this work. Our goals are to develop a mathematical model allowing us to analyse the trade-off between the attacker and the defender resources on one hand, and to offer prevention mechanisms that can be used to defend against this category of attacks, on the other hand. We model the server queue of connections using Markov chains to establish a relationship between the server capacity, the attack rate and the impact on the service level. We analyse two methods of adjusting the timeout, *threshold* and *linear*, and we couple them with three policies of assigning the timeout to connections: the *deterministic* policy, the *deferred* policy and the utopian *Poisson* policy.

The results we obtained confirm our intuitions. First, theoretical modelling shows that for any given strategy there exists a *linear* trade-off between the attack rate and targeted server queue size, in the sense that when the first one increases, the second one must be increased as well to maintain the same quality of service for legitimate users. However, the ratio that needs to be kept between these values differs between strategies; in that sense some are better than others. In particular, theoretical modelling also indicates that the *linear deferred* timeout strategy is very similar in performance to the *linear Poisson* timeout strategy, which in turn outperforms all the other dynamic timeout strategies. The dynamic timeout strategies always outperform the classical *fixed* timeout method.

Our model is very general and can be used to describe the behaviour of the server queue during connection depletion attacks at various levels in the TCP/IP protocol stack. We confirm the theoretical findings using stochastic simulations and network experiments of SYN-flood attacks. We also show how the model can be used when analysing a TCP connection establishment flood or a ticket reservation flood. The protection strategies we suggest are robust to changes in the attack model and our implementation is very efficient and transparent with respect to the server and applications it tries to protect. The strategies could therefore be easily integrated into existing Operating Systems (OS) and applications, or implemented in network devices.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES ANNEXES

LISTE DES SIGLES ET ABRÉVIATIONS

$\triangleq$	Egal par définition
ACK	Acknowledgement
AP	Attraction-Point
BIT	Burst Interarrival Time (Temps entre les arrivées des rafale)
CDF	Cumulative Distribution Function (Fonction de répartition)
cnx	Connexion
DDoS	Distributed Denial of Service (Déni de sérvice distribué)
DoS	Denial of Service (Déni de sérvice)
EXP	Experience
FIFO	First in First Out (Premier arrivé, premier sorti)
FILO	First In Last Out (Premier arrivé, dernier sorti)
FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
ISP	Internet Service Provider (Fournisseur de services Internet)
MOL	Modified-Offered-Load
OS	Operating System (Système d'exploitation)
OSI	Open Systems Interconnection
PDF	Probability Distribution Function (Densité de probabilité)
QG	Queue Guardian
QoS	Quality of Service (Qualité de sérvice)
RFC	Request For Comment
RST	Reset
SIM	Simulation
SIP	Session Initiation Protocol
SYN	Synchronization
SS	Steady-State
SSL	Secure Sockets Layer

TCP	Transmission Control Protocol
TH	Théorique
TLS	Transport Layer Security
VPN	Virtual Private Network (Réseau privé virtuel)

DESCRIPTION DES VARIABLES UTILISÉES

c	Capacité
E_k	État de la file avec k positions occupées
S	Seuil
v	Virulence de l'attaque
$\|i\|$	Connexion qui occupe la position i dans la file
A_i	État de la file à l'arrivé de la connexion i
τ_i	Temps d'arrivé de la connexion i
T_{out}	Délai d'inactivité (*timeout*)
$T_{\text{out}}^{(k)}$	Délai d'inactivité dans l'état E_k
T_0	Délai d'inactivité à file vide
T_1	Délai d'inactivité à file pleine
$\tilde{t}$	Temps moyen de service
$\tilde{t}_S$	Temps moyen de service à seuil
t_{l}	Temps moyen de service des connexions légitimes
$t_{\text{l}}^{(k)}$	Temps moyen de service des connexions légitimes dans l'état E_k
$t_{\text{l}}^{(h\to k)}$	Temps moyen de service des connexions légitimes dans l'état E_k si l'état precedent est E_h
t_{m}	Temps moyen de service des connexions malicieuses
$t_{\text{m}}^{(k)}$	Temps moyen de service des connexions malicieuses dans l'état E_k
$t_{\text{m}}^{(h\to k)}$	Temps moyen de service des connexions malicieuses dans l'état E_k si l'état precedent est E_h
$C(t)$	Fonction de répartition du temps de service
$C_{\text{l}}(t)$	Fonction de répartition du temps de service des connexions légitimes
$C_{\text{m}}(t)$	Fonction de répartition du temps de service des connexions malicieuses
$G(t)$	Densité de probabilité du temps de service

$G_{\mathrm{l}}(t)$	Densité de probabilité du temps de service des connexions légitimes
$G_{\mathrm{l}}^{(k)}(t)$	Densité de probabilité du temps de service des connexions légitimes dans l'état E_k
$G_{\mathrm{l}}^{(h\rightarrow k)}(t)$	Densité de probabilité du temps de service des connexions légitimes dans l'état E_k si l'état precedent est E_h
$G_{\mathrm{m}}(t)$	Densité de probabilité du temps de service des connexions malicieuses
$G_{\mathrm{m}}^{(k)}(t)$	Densité de probabilité du temps de service des connexions malicieuses dans l'état E_k
$G_{\mathrm{m}}^{(h\rightarrow k)}(t)$	Densité de probabilité du temps de service des connexions malicieuses dans l'état E_k si l'état precedent est E_h
λ	Taux d'arrivé
$\lambda^{(k)}$	Taux d'arrivé dans l'état E_k
λ_{l}	Taux d'arrivé des connexions légitimes
λ_{m}	Taux d'arrivé des connexions malicieuses
μ	Vitesse de service
μ_0	Vitesse de service avant seuil
μ_1	Vitesse de service après seuil
μ_S	Vitesse de service des connexions à seuil
$\mu_{\lvert i \rvert}$	Vitesse de service de la connexion qui occupe la position i dans la file
$\tilde{\mu}_{\lvert i \rvert}$	Approximation de la vitesse de service de la connexion qui occupe la position i dans la file
μ_{l}	Vitesse de service des connexions légitimes
$\mu_{\mathrm{l}}(t)$	Vitesse de service des connexions légitimes à l'instant t
$\mu_{\mathrm{l}S}$	Vitesse de service des connexions légitimes à seuil
μ_{c}	Vitesse d'achèvement des connexions légitimes
μ_{m}	Vitesse de service des connexions malicieuses
$\mu_{\mathrm{m}}(t)$	Vitesse de service des connexions malicieuses à l'instant t
$\mu_{\mathrm{m}S}$	Vitesse de service des connexions malicieuses à seuil
$\mu_{MOL}(t)$	Vitesse de service à l'instant t d'après l'approximation MOL
$\mu^{(k)}$	Taux de service des connexions dans l'état E_k
$\tilde{\mu}^{(k)}$	Approximation du taux de service des connexions dans l'état E_k

$\mu^{(k)}_{MOL}(t)$	Taux de service à l'instant dans l'état E_k d'après l'approximation MOL
n_{ss}	Nombre des connexions qui arrivent avant que la file soit stable
$m_\infty(t)$	Nombre moyen des positions occupées dans une chaine $M_t/G/\infty$ à l'instant t
$m_{\text{l}\infty}(t)$	Nombre moyen des positions occupées par des connexions légitimes dans une chaine $M_t/G/\infty$ à l'instant t
$m_{\text{m}\infty}(t)$	Nombre moyen des positions occupées par des connexions malicieuses dans une chaine $M_t/G/\infty$ à l'instant t
$p^{(k)}$	Probabilité d'être dans l'état E_k à état stable
$\tilde{p}^{(k)}$	Approximation de la probabilité d'être dans l'état E_k à état stable
$p^{(k)}(t)$	Probabilité d'être dans l'état E_k à 'instant t
$p^{(h\rightarrow k)}$	Probabilité d'être dans l'état E_k sachant que l'état precedent est E_h
p_{le}	Probabilité d'expiration d'une connexion légitime si elle est rentrée dans la file
p_{li}	Probabilité d'expiration instantanné d'une connexion légitime après transition
$p^{(k)}_{\text{le}}$	Probabilité d'expiration d'une connexion si elle est rentrée dans la file sachant que le délai d'inactivité est $T^{(k)}_{\text{out}}$
p_{me}	Probabilité d'expiration d'une connexion malicieuse si elle est rentrée dans la file
p_{mi}	Probabilité d'expiration instantanné d'une connexion malicieuse après transition
$\phi(t)$	Probabilité d'échec de la connexion arrivée à l'instant t
$\phi_{\text{r}}(t)$	Probabilité de rejet de la connexion arrivée à l'instant t
$\phi_{\text{e}}(t)$	Probabilité d'expiration de la connexion arrivée à l'instant t
Φ	Probabilité moyenne d'échec des connexions légitimes arrivées à l'état stable
$\tilde{\Phi}$	Approximation de la probabilité moyenne d'échec des connexions légitimes arrivées à l'état stable
Φ_{r}	Probabilité moyenne de rejet des connexions légitimes arrivées à l'état stable

$\tilde{\Phi}_{\mathrm{r}}$	Approximation de la probabilité moyenne de rejet des connexions légitimes arrivées à l'état stable
Φ_{e}	Probabilité moyenne d'expiration des connexions légitimes arrivées à l'état stable
$\tilde{\Phi}_{\mathrm{e}}$	Approximation de la probabilité moyenne d'expiration des connexions légitimes arrivées à l'état stable
φ	Pourcentage des connexions échouées
φ_{r}	Pourcentage des connexions rejetées
φ_{e}	Pourcentage des connexions expirées
$B(c)$	Formule Erlang-B pour un serveur de capacité c
$\tilde{B}(c)$	Approximation de la formule Erlang-B pour un serveur de capacité c
$\delta(t)$	Fonction delta de Dirac
$\Gamma(z)$	Fonction Gamma d'Euler
$\Gamma(a, z)$	Fonction Gamma incomplète
$E_n(z)$	Fonction exponentielle intégrale
$M/G/c/c$	Chaine de Markov avec processus d'arrive de Poisson, processus de service general, c positions dans la file et maximum c connexion servies en parallel
$M_t/G/c/c$	Chaine de Markov avec processus d'arrive de Poisson variant dans le temps, processus de service general, c positions dans la file et maximum c connexion servies en parallel
$M_t/G/\infty$	Chaine de Markov avec processus d'arrive de Poisson variant dans le temps, processus de service general, sans limite des positions dans la file et sans limite de connexion servies en parallel

CHAPITRE 1

Introduction

La sécurité informatique a comme but de préserver les caractéristiques suivantes d'un système ou d'un service informatique : confidentialité, intégrité et disponibilité. La confidentialité signifie qu'il est possible d'utiliser le service sans que personne à part le client et le serveur ne soit capable de savoir quelle information a été échangée. L'intégrité signifie que chaque partie est capable de vérifier que les données qu'elle a reçues proviennent de la bonne source et qu'aucune modification intentionnelle ou accidentelle n'ait eu lieu. Finalement, la disponibilité signifie que les données que le client demande sont présentes et les services fonctionnent dans les paramètres de qualité de service (QoS) préalablement définis. Les aspects de confidentialité et intégrité sont en partie solutionnés par l'usage de la cryptographie. Cependant, quand le système que nous considérons est l'Internet, la gestion de la confiance et des clés de chiffrement s'avère difficile. La disponibilité est encore plus difficile à assurer à cause de l'architecture de l'Internet qui permet la communication entre tous les participants sans pouvoir offrir une qualité de service autre que localement. Ceci ouvre la porte aux attaques de déni de service (DoS) qui peuvent avoir comme but de diminuer la disponibilité d'un service d'une entreprise, de tous les services d'une entreprise et même d'un pays entier.

Nous identifions trois grandes catégories d'attaques de déni de service :

1. Attaques d'innondation (*flooding*), qui visent à saturer les canaux de communication réseau.
2. Attaques qui exploitent des vulnérabilités dans l'implémentation des protocoles de communication.
3. Attaques qui exploitent des faiblesses dans la logique des protocoles de communication, dues au fait que lors de la conception des protocoles, la sécurité n'a pas été le souci principal.

Le premier type d'attaque, l'inondation, est difficile à combattre mais pour qu'une attaque réussisse il faut que l'attaquant dispose d'une connexion Internet avec une capacité supérieure de celle de la victime. Très souvent, l'utilisation de la technique d'innondation du canal de communication, est utilisé dans des attaques de déni de service distribuées (DDoS). Dans ce cas, l'attaquant emploie plusieurs machines et équipements réseau, pour avoir une capacité de communication agrégée qui dépasse rapidement celle de la cible, s'il s'agit d'un seul serveur attaqué. Cependant, ce n'est plus intéressant pour l'attaquant d'utiliser cette méthode pour rendre indisponible un grand nombre des serveurs si une architecture distribuée est adoptée par l'entreprise ou l'organisme cible, parce que la force de l'attaque est répartie sur chaque serveur attaqué.

Le deuxième type d'attaque, qui vise à exploiter les vulnérabilités dans l'implémentation des protocoles de communication est très facile à réaliser si une telle vulnérabilité est connue. La plupart des fois, les vulnérabilités sont des débordements de tampon, cependant les outils et méthodes de développement sont arrivés à un niveau de maturité qui rend plus rares les vulnérabilités d'implémentation. De plus, une fois qu'une vulnérabilité est connue, le constructeur de l'application en question publie des mises à jour qui rendent la vulnérabilité inexploitable. La fenêtre de temps d'utilisation des attaques de ce type est donc limitée.

Le troisième type d'attaque, qui vise à exploiter les faiblesses dans la logique des protocoles de communication vient avec un avantage pour l'attaquant : même si une faiblesse dans la logique d'un protocole est trouvée, il est très probable que le protocole ne changera pas s'il est déjà adopté par la majorité des acteurs sur Internet.

Dans le reste de ce mémoire nous nous intéressons aux attaques de déni de service de la troisième catégorie. La faiblesse logique que nous considérons est présente dans la plupart des protocoles de communication orientés connexion : un client malicieux peut simuler l'établissement d'une connexion avec le serveur de sorte que le serveur alloue des ressources mémoire pour garder l'état de la connexion. Parce que l'attaquant ne doit pas établir la connexion avec le serveur mais seulement convaincre le serveur que la connexion est établie ou en cours d'établissement pour que le serveur alloue des ressources, ce type d'attaque est très avantageux pour l'attaquant. L'attaque la plus connue de ce type est l'attaque *SYN-Flood*, où l'attaquant envoie des messages TCP avec le drapeau SYN ce qui déclenche l'établissement des connexions du coté serveur. Cependant, l'analyse que nous faisons et les solutions que nous allons proposer s'appliquent aussi aux autres protocoles orienté connexion, par exemple HTTP, FTP, SSL/TLS, SIP, SMTP.

Une attaque d'épuisement des ressources mémoire du serveur est relativement facile à détecter, l'indicateur le plus accessible à consulter étant le nombre de connexions établies ou en cours d'établissement. Si cette valeur est beaucoup plus élevée que la valeur attendue, alors il est probable que le serveur soit sous attaque. Même s'il est possible de détecter une attaque, il est très difficile, voire impossible, de discriminer le trafic légitime du trafic malicieux avant de tenter d'établir la connexion parce que, comme dans le cas de l'attaque *SYN-Flood*, le trafic malicieux peut copier toutes les caractéristiques du trafic légitime. Entre autre, la distribution dans le temps des paquets peut être très facilement copiée par une attaque et la distribution géographique des adresses IP n'est pas une limite non plus. Un attaquant peut utiliser la technique de *IP spoofing* pour envoyer des messages qui proviennent de la même source mais qui semblent venir des adresses distinctes. De plus, un attaquant pourrait contrôler un *botnet*, un réseau d'ordinateurs infectés sur internet, ce qui rend toute méthode de filtrage encore plus difficile, car dans ce cas tout le trafic provient de machines avec des vraies adresses IP et avec un comportement légitime la plupart du temps.

Les stratégies de protections suggérées jusqu'à présent ne sont malheureusement pas si efficaces et faciles à déployer que prévu par leurs auteurs. Ceci est prouvé par le fait que de déni de service est toujours un problème d'actualité. Parmi les attaques les plus récentes, nous identifions l'attaque de type *SYN-Flood* contre GoDaddy, la plus grande compagnie d'enregistrement des noms de domaines et d'hébergement web, qui a eu lieu en mars 2007 (Murphy, 2007). L'attaque a enregistré des taux de 30 Mbps et a rendu le site inaccessible pendant 5 heures. En septembre 2007, plusieurs sites dédiés à la protection contre le spam ont subi des attaques de déni de service de différents types, certains enregistrant des taux de l'ordre 10 Mbps durant plusieurs heures (Spamnation, 2007). En octobre 2007, le site des guides financiers pour les consommateurs MoneySavingExpert.com a été inaccessible durant une fin de semaine dû à une attaque de déni de service (Leyden, 2007). L'attaque la plus puissante enregistrée à date est l'attaque de déni de service contre différentes sites web administratifs de l'Estonie arrivée en mai 2007 (Nazario, 2007). Cette attaque qui a duré plus de deux semaines et a atteint des taux de 100 Mbps durant plusieurs heures consécutives, a prouvé que le déni de service est une arme viable dans la guerre cybernétique. Plus récemment, en février 2008, une attaque de déni de service qui consiste à inonder les serveurs avec des demandes de pages HTTP a rendu 32 sites web des casinos en ligne inutilisables pendant 8 jours (Adair, 2008). Cette dernière attaque est un exemple concrèt de l'utilisation de la même technique d'épuisement des ressources contre un protocole de plus haut niveau, HTTP.

Malgré les efforts qui ont été faits par le monde académique et l'industrie pour résoudre le problème du déni de service, il n'existe pas de mesure de protection efficace facilement utilisable contre les attaques d'épuisement de connections. De plus, il n'existe pas de modèle théorique qui permette d'évaluer quantitativement l'impact des attaques sur la qualité de service. L'objectif de ce mémoire est d'adresser le problème des attaques d'épuisement de ressources en proposant un modèle mathématique qui permet d'évaluer quantitativement l'impact des attaques sur la qualité de service. Ensuite, nous visons à proposer des stratégies de gestion de la file des connexions du serveur dans le but de la rendre plus résistante aux attaques de déni de service. Ces stratégies ne doivent pas se baser sur la discrimination du trafic légitime et malicieux et nous devons pouvoir appliquer ces stratégies facilement à tous les protocoles orienté connexion vulnérables aux attaques d'épuisement de ressources. La performance des stratégies doit être évaluée à l'aide du modèle mathématique construit préalablement et ensuite validée par des simulations et expérimentalement.

Ce mémoire suit la formule de présentation d'un mémoire par articles et est organisé comme suit : le chapitre 2 présente une revue critique de la littérature concernant les travaux antérieurs de modélisation, de détection et de protection contre des attaques de déni de service. Par la suite, le chapitre 3 décrit la démarche de l'ensemble du travail de recherche. Le chapitre 4 présente le modèle mathématique, les stratégies du délai d'inactivité dynamique ainsi que les résultats, tel que soumis dans l'article *An Exhaustive Study of Queue Management as a DoS Counter-Measure* à la revue *International Journal of Information Security* (publiée par l'éditeur scientifique Springer). Ensuite, le chapitre 5 offre une discussion des résultats obtenus, les conclusions tirées des travaux effectuées ainsi que des directions de recherche pour des travaux futurs. L'annexe A présente des résultats préliminaires du modèle mathématique avec la politique d'assignation du délai d'inactivité *Poisson* et des résultats de simulation, tel que publiés dans les actes de la conférence *Information Security Conference (ISC)* en octobre 2007. L'annexe B présente les résultats expérimentaux des stratégies du délai d'inactivité dynamique, tel que publiés dans les actes de l'atelier *ACM Quality of Protection (QoP) Workshop*, organisé dans le cadre de la conférence *ACM Communications and Computer Security Conference (CCS)* en octobre 2007.

CHAPITRE 2

Modélisation et protection contre le déni de service

Dans ce chapitre nous faisons un survol de littérature par rapport aux attaques de déni de service (DoS). Premièrement, nous regardons les travaux de modélisation des attaques et la mesure de performance des serveurs sous attaque. Ensuite nous analysons les mécanismes de protection contre les attaques d'épuisement des ressources qui ont été proposées par le monde académique et l'industrie.

2.1 Modélisation et mesure de performance

Le premier modèle mathématique qui formalise le compromis entre les ressources de l'attaquant et celles du défendant a été introduit par Meadows (1999, 2001). Ce modèle n'est pas suffisant pour mesurer le degré auquel un protocole est vulnérable parce qu'il est difficile de définir des fonctions de coût concrètes pour les opérations élémentaires, par exemple le refus d'un message ou le calcul d'une signature digitale. De plus, ce modèle ne considère pas les implications pour un défendant d'une qualité de service réduite dans des situations où le service est encore fonctionnel, durant ou après une attaque de déni de service.

Les chaines de Markov à temps discret sont un outil très utilisé dans la détection des anomalies et des intrusions. Un tel modèle qui prend en considération la tolérance du serveur face à différentes attaques a été proposé par Madan *et al.* (2002). Cependant, l'application du modèle aux attaques d'épuisement des ressources implique l'estimation de variables difficile à mesurer, par exemple la probabilité qu'une attaque ne soit pas détectée, la probabilité qu'un système résiste à une attaque ou le temps moyen d'un système pour devenir vulnérable.

Lui *et al.* (2004) proposent un modèle basé sur des chaines de Markov à temps continu pour montrer que les protocoles à *état dur* ont une meilleure performance que les protocoles à *état mou* si les conditions du réseau sont parfaitement définies. Cependant, ce sont les protocoles à *état mou* qui sont plus résistants aux attaques et fluctuations non attendues. Le *degré de dureté* d'un protocole est évalué par rapport à la manière dont le délai d'inactivité (T_{out}) est utilisé. La plupart des protocoles orientés connexion utilisés sur Internet sont des protocoles à *état dur*, qui utilisent le délai d'inactivité seulement comme mesure de protection en cas d'échec de communication.

D'autres modèles se contentent de détecter la présence des attaques de déni de service d'épuisement des ressources. Khan et Traoré (2005) observent que le taux de remplissage de la file des connexions est un bon indicateur pour détecter les attaques de *SYN-Flood*. Wang *et al.* (2002) utilisent la méthode *change-point* pour détecter un changement brusque dans le nombre des paquets SYN qui n'ont pas de correspondent FIN. Des travaux de recherche dans la même direction ont été faits par Tartakovsky *et al.* (2006) pour utiliser la méthode *change-point* avec différents algorithmes de détection. Divakaran *et al.* (2006) utilisent la technique de prédiction linéaire pour détecter les attaque de *SYN-Flood*, en regardant la différence entre le nombre de paquets de type SYN reçus et le nombre de paquets SYN/ACK envoyés par le serveur. L'utilisation des algorithmes de détection d'anomalies est étudiée par Siris et Papagalou (2004) pour détecter les attaques de *SYN-Flood*. Finalement, différents algorithmes de détection ont été implémentés et testés par Beaumont-Gay (2007) pour observer que la performance des algorithmes varie beaucoup en fonction du jeu de test utilisé. Même s'il est important de pouvoir détecter qu'un serveur est sous attaque pour savoir où concentrer les efforts, la détection des attaques est loin de résoudre le problème de déni de service.

Mirkovic *et al.* (2007a, 2006) introduisent la notion de transaction comme étant l'agrégation d'un ensemble de paquets qui ont comme but une tâche de haut-niveau, par exemple, l'établissement d'une connexion TCP ou le transfert d'une page HTML. Une transaction est considérée échouée si les paramètres de qualité de service ne sont pas satisfaits. La qualité de service globale est calculée en fonction du pourcentage de transactions échouées. Ceci est un premier essai de définition d'une métrique de déni de service qui prend en compte la qualité de service perçue par l'usager.

La plateforme d'essai DETER (Mirkovic *et al.*, 2007b) permet de reproduire des scénarios d'attaque et de mesurer expérimentalement la performance de serveurs durant les attaques. Malheureusement, il n'existe pas de modèle mathématique équivalent qui permet d'estimer la performance du serveur sachant les différents paramètres de configuration et du trafic. Ceci est un des problèmes que nous essayons de traiter.

2.2 Mécanismes de protection

La RFC4732 (Handley et Rescorla, 2006) est une introduction aux attaques de déni de service qui vise à offrir aux concepteurs des protocoles des patrons et des solutions partielles pour résoudre ce problème. Une autre très bonne introduction au déni de service est le livre *Internet Denial of Service : Attack and Defence Mechanisms* (Mirkovic *et al.*, 2004) qui vise à expliquer le problème et les actions qui peuvent être prisent par les administrateurs des réseaux. Mirkovic et Reiher (2004) proposent une taxonomie des mécanismes d'attaques et de protection par rapport au déni de service en général. La caractérisation des mécanismes de protection est faite par rapport aux critères suivants : le niveau d'activité (préventif ou actif), le niveau de coopération (autonome, coopératif ou interdépendant), le lieu de déploiement (réseau de la victime, réseau intermédiaire ou réseau source) et la stratégie de réponse (identification de l'agent, limitation de la bande-passante, filtrage ou reconfiguration). Cette taxonomie s'avère très utile pour la classification mécanismes de protection contre les attaques de déni de service en général. Par contre, nous nous intéressons seulement aux attaques de déni de service d'épuisement des ressources. Dans ce cadre nous identifions quatre grands axes de recherche des mécanismes de protection : la modification de la logique des protocoles, la modification de l'implémentation des protocoles, la discrimination du trafic malicieux et la protection collaborative.

2.2.1 Protection par modification de la logique des protocoles

Parce que la vulnérabilité qui est exploitée dans les attaques d'épuisement des connexions est la logique des protocoles, plusieurs propositions ont été faites pour apporter des modifications à la logique des protocoles de manière à rendre les attaques très couteuses ou difficiles pour l'attaquant.

Les *client puzzles* (Juels et Brainard, 1999) consistent à donner au client un problème cryptographique à résoudre avant que le serveur alloue les ressources pour la connexion. La supposition derrière ce mécanisme est que les clients malicieux établissent beaucoup de connexions depuis les mêmes machines physiques. La vitesse de résolution des problèmes cryptographiques est limitée par la puissance de calcul et donc un client malicieux aura un taux d'attaque utile très limité. Les *network puzzles* proposés par Feng *et al.* (2005) consistent à utiliser la même approche que les *client puzzles*. Cependant, ce mécanisme est conçu pour être déployé au niveau réseau de la pile des protocoles du modèle OSI, dans le but d'offrir un mécanisme de punition contre tout type d'attaque d'inondation. Le principal désavantage de cette méthode est l'impact ressenti par les clients légitimes qui doivent aussi résoudre les problèmes cryptographiques. L'impact est encore plus important si la puissance de calcul est limitée, fait qui rend cette méthode très couteuse pour les ordinateurs et les téléphones portables.

Un autre mécanisme de protection, les *SYN cookies* (Bernstein, 2003; Zuquete, 2002), propose une manière d'établissement des connexions TCP sans garder l'état et sans allouer de ressources. L'approche consiste à chiffrer les paramètres de connexion et de les encoder dans les numéros de séquence du message SYN+ACK envoyé au client. Si le client répond avec le message ACK, les paramètres sont récupérés du numéro de séquence et la connexion est établie. Cette approche a le désavantage que la taille du numéro de séquence ne permet pas d'encoder toutes les options TCP. De plus, le protocole TCP requiert la retransmission des messages qui n'ont pas eu un accusé de réception, or cela n'est pas possible durant l'établissement de la connexion avec le mécanisme de *SYN cookies* parce qu'aucun état n'est gardé. Pour enlever les limitations des *SYN cookies*, Lemon (2002) propose le mécanisme *SYN cache*, qui consiste à garder l'état durant l'établissement des connexions mais de n'allouer que le strict nécessaire des ressources pour sauvegarder les paramètres. Le options restantes sont, similairement aux *SYN cookies*, encodées dans le numéro de séquence TCP.

Une autre approche proposée pour protéger les serveurs web des attaques de déni de service consiste à rediriger les clients légitimes vers un autre serveur via un message de redirection HTTP (Xu et Lee, 2003). C'est ce deuxième serveur qui offre le service que les clients attendent. En partant du principe que les attaques ne sont pas ciblées et adaptées pour une certaine victime, Al-Duwairi et Manimaran (2006) proposent un mécanisme de protection contre les attaques de *SYN-Flood* qui consiste à ignorer le premier paquet SYN provenant de chaque adresse IP. En plus d'avoir un impact négatif sur la performance des clients, ces mesures naïves de protection n'ont aucun effet contre les attaques ciblées. Dans ce cas, il est probable que l'attaquant découvre le système de protection et configure les paramètres des attaques de sorte à contourner les mécanismes de protection.

Plus récemment, des propositions ont été faites par Ghavidel et Issac (2007) pour remplacer le protocole TCP avec un protocole équivalent mais qui précède l'établissement de la connexion par une phase d'authentification. Encore une fois, ce mécanisme implique une détérioration de performance pour les clients légitimes dû au calcul cryptographique et aux messages supplémentaires d'authentification qui doivent être transmis. De plus, pour des raisons politiques, de compatibilité et du cout élevé d'opération dans les premières phases, la migration du protocole IP vers IPv6 est en cours depuis 15 ans et n'est pas encore achevée. Les mêmes types de problèmes sont envisageables pour la migration vers le protocole TCP sécurisé proposé par Ghavidel et Issac (2007) et il n'est donc pas raisonnable de croire que ce protocole soit adopté sur Internet dans le futur proche.

Nous avons vu que, généralement, les mécanismes de protection qui consistent à apporter des modifications à la logique des protocoles ont comme principe de défense l'introduction d'une charge que seules les clients légitimes peuvent supporter. Ceci n'est pas encourageant parce qu'aujourd'hui la majorité des attaques de déni de service sont réalisées à l'aide des *botnets*, des grands réseaux d'ordinateurs personnels compromis qui sont sous le contrôle de l'attaquant. Il est estimé que le ver informatique Storm a été utilisé pour la construction d'un *botnet* qui contient des millions d'ordinateurs (Smith, 2008). L'utilisation d'un tel réseau pour des attaques de déni de service contourne les mesures de protection qui visent à pénaliser les clients malicieux parce que chaque client qui participe à l'attaque se comporte individuellement comme un client légitime.

2.2.2 Protection par modification de l'implémentation des protocoles

Une autre stratégie de protection contre les attaques de déni de service est de modifier l'implémentation des protocoles ou d'ajuster les paramètres de configuration pour rendre le serveur plus résistant aux attaques. Les meilleures pratiques de gestion des systèmes d'exploitation modernes conseillent toujours d'augmenter la taille de la file des connexions ouvertes TCP, comme mesure de protection contre les attaques *SYN-Flood*. Cependant, dans la pratique, la taille maximale de la file est limitée par la quantité de mémoire disponible sur le serveur, surtout si le serveur offre plusieurs services réseau.

Les protocoles orientés connexion implémentent un mécanisme de délai d'inactivité des connexions, connu en anglais comme *timeout*. Si une réponse n'est pas reçue dans l'intervalle de temps attendu, la connexion est rejetée. Une méthode de protection évidente contre les attaques d'épuisement de ressources est de diminuer le délai d'inactivité mais cette méthode bloque de façon permanente l'accès aux clients sur un réseau avec un temps de réponse plus long que le délai d'inactivité. Dans le système d'exploitation Windows Server (Microsoft TechNet, 2003), un mécanisme de détection d'attaque est implémenté. Si une attaque est détectée, un délai d'inactivité plus restrictif est utilisé durant l'attaque. Cependant, il n'est pas clair quelles sont les valeurs des seuils qui déclenchent la détection des attaques, quel délai d'inactivité doit être utilisé durant une attaque et comment appliquer le délai d'inactivité dynamique aux connexions déjà présentes dans la file, pour maximiser la qualité de service que les clients perçoivent. Si les bonnes valeurs ne sont pas choisies, cette méthode a le désavantage d'offrir la possibilité à un attaquant de déclencher une attaque avec des ressources limitées, seulement pour mettre le serveur dans un état restrictif dans lequel les clients plus lents sont bloqués.

Une méthode différente de contrôle du délai d'inactivité a été proposée par Schuba *et al.* (1997) sous le nom de *SYNkill*. Ceci consiste à écouter sur le réseau pour détecter les connexions ouvertes sur le serveur. Si la connexion a passé plus de temps sur le serveur que le moniteur le permet, un paquet RST est envoyé par le moniteur au serveur pour fermer la connexion. La même idée est citée et testée par Nakashima et Oshima (2006) et Nakashima et Sueyoshi (2007). Cette méthode a l'avantage d'être indépendante de la plateforme qu'elle doit protéger mais a le même désavantage que la diminution du délai d'inactivité : les clients sur un réseau lent sont toujours refusés.

Il a été suggéré par Schuba *et al.* (1997) et Ohsita *et al.* (2005) d'utiliser un mécanisme de relai TCP dans le pare-feu. C'est le pare-feu qui accepte la connexion du client, et seulement après que la connexion est établie, le pare-feu ouvre une connexion avec le serveur. Ceci a l'effet de déplacer la file sous attaque sur le pare-feu et est implémenté par les pare-feu Checkpoint (Noureldien et Osman, 2000). Cependant, si la file du pare-feu est inondée, l'effet est plus grave parce qu'aucun des serveurs protégès ne sera plus disponible. De plus, comme mentionné par Noureldien et Osman (2000), une attaque de saturation de la table d'état du pare-feu avec des paquets ACK est envisageable.

Les travaux de recherche de modification des implémentations des protocoles semblent prometteurs, parce que de telles mesures sont faciles à implémenter et à déployer et elles sont indépendantes de la plateforme à protéger. Cependant, il ne faut pas que des telles mesures ouvrent la porte aux autres types d'attaque ou que les clients légitimes perçoivent un impact négatif de la qualité de service pendant le fonctionnement sans attaque. De plus, cette approche de renforcement de sécurité de la cible est compatible et complémentaire avec les autres approches de protection, par discrimination du trafic malicieux ou par protection collaborative. Les stratégies de gestion dynamique du délai d'inactivité que nous proposons se retrouvent dans cette catégorie.

2.2.3 Protection par discrimination et filtrage du trafic malicieux

Plusieurs mécanismes de discrimination du trafic malicieux se basent sur l'analyse statistique du trafic. Dans ce sens, Cheng *et al.* (2002) proposent l'utilisation d'une méthode d'analyse spectrale pour détecter les flots de données malicieux. Feinstein *et al.* (2003) suggèrent l'analyse des distributions des adresses IP source pour détecter quelles sont les adresses IP des clients malicieux. Ayres *et al.* (2006) utilisent la technique PacketScore (Kim *et al.*, 2006) pour calculer un score et ensuite utiliser un seuil sur le score pour décider si un paquet est malicieux ou pas. Lim et Uddin (2005) étudient l'implémentation des méthodes d'analyse statistique de détection des attaque de type *SYN-Flood* sur des processeurs réseaux.

D'autres méthodes de discrimination visent à combattre la technique de *IP spoofing* qu'un attaquant pourrait utiliser, technique qui consiste à envoyer des paquets IP avec des fausses informations dans le champ d'adresse de provenance. Varanasi *et al.* (2004) proposent l'enregistrement des adresses IP des routeurs d'entrée et de sortie pour les paquets IP afin d'utiliser un modèle de chaines de Markov cachées pour dépister le trafic IP venant des fausses adresses. La technique *Hop-Count Filtering* (Jin *et al.*, 2003) consiste à comparer le nombre de routeurs qu'un paquet de type SYN en provenance d'une adresse IP a traversé par rapport à la même information dans les traces des connexions réussies. Zou *et al.* (2006) proposent la configuration adaptive des paramètres de la technique *Hop-Count Filtering* en fonction de la sévérité de l'attaque afin de mieux protéger un serveur contre des attaques de type *SYN-Flood.* Une idée explorée par Xiao *et al.* (2005) est d'utiliser la technique *Delay Probing Method* pour mesurer la congestion dans le réseau et décider si la réponse tardive du client est dûe aux conditions du réseau ou à une attaque de *SYN-Flood.*

Nous avons vu différentes types de méthodes de discrimination du trafic malicieux. Ces méthodes supposent que le trafic malicieux a au moins une caractéristique distinctive par rapport au trafic légitime. Cependant, encore une fois, l'utilisation des *botnet* pour réaliser les attaques de déni de service peut rendre le trafic malicieux indistinguable du trafic légitime, parce que les machines utilisées pour l'attaque ont toutes les caractéristiques des machines légitimes. Il est donc futile d'essayer de construire un mécanisme de protection par discrimination du trafic malicieux tant que les *botnets* sont actifs sur Internet, un problème qui est probablement plus difficile que le déni de service.

2.2.4 Protection par collaboration

Étant donnés les effets négatifs pour les clients légitimes que les approches de protection par des modifications de protocoles introduisent, et l'infaisabilité d'une mesure de protection par discrimination du trafic malicieux devant les *botnets*, il a été suggéré que seule une mesure de protection collaborative peut combattre efficacement les attaques de déni de service. Le premier effort significatif dans ce sens est la RFC2267 (Ferguson et Senie, 1998) qui spécifie que les routeurs des fournisseurs d'accès Internet doivent router sur Internet seulement les paquets provenant des adresses IP qui sont censées être derrière ces routeurs. Cette méthode vise à bloquer la technique de *IP spoofing*, qui permettrait à un attaquant d'envoyer des paquets SYN qui semblent venir de fausses adresses. Une amélioration de la RFC2267 proposée par Chen et Song (2005) consiste à implémenter des mécanismes de détection des attaques de déni de service sur les routeurs de fournisseurs d'accès Internet, pour identifier et limiter les éventuelles attaques provenant de leurs clients.

L'architecture DefCOM (Oikonomou *et al.*, 2006; Robinson *et al.*, 2003; Mirkovic *et al.*, 2003, 2002) consiste à faire communiquer les équipements réseau sur Internet sur un canal sécurisé dans le but de partager l'information concernant le trafic d'attaque et de diminuer la priorité de routage de ce trafic. Des suggestions ont été faites par Yang *et al.* (2005) pour transmettre dans l'entête de chaque paquet IP une mesure qui reflète si le paquet est désiré ou pas à la destination. En plus de cette mesure, Natu et Mirkovic (2007) proposent d'utiliser la réputation des clients à long terme pour identifier les clients légitimes. Ces deux dernières approches sont très couteuses en terme des ressources mais il est envisageable de les intégrer à l'architecture DefCOM pour être actives seulement durant une attaque.

L'architecture SOS (Keromytis *et al.*, 2004) emploie des fonctions de hachage cryptographique pour construire un réseau de routage secret relié à un pare-feu distribué. Le désavantage principal de cette méthode est l'augmentation du temps de latence du trafic introduit à cause du routage supplémentaire dans le réseau secret. Pushback (Ioannidis et Bellovin, 2002) est un mécanisme qui permet aux routeurs d'identifier la direction de provenance du trafic d'attaque et de demander aux routeurs en amont de limiter ce trafic. De manière similaire, Yau *et al.* (2005) suggère un mécanisme de limitation du trafic provenant des routeurs en amont qui génèrent une congestion. Zhang et Dasgupta (2003) proposent la collaboration des routeurs près de la victime pour marquer les paquets IP afin d'identifier les routeurs de provenance des attaques. Cependant, ces approches ont le désavantage de pénaliser les clients légitimes qui utilisent une route commune avec l'attaquant.

Étant donné que pour des raisons politiques et sociales, la RFC2267 n'a pas encore été adopté à échelle mondiale malgré sa facilité d'implémentation, il est peu probable qu'une des architectures plus complexes comme DefCOM ou Pushback soit déployée sur Internet dans le futur proche. De plus, les mécanismes de protection contre les attaques de déni de service par collaboration sont conçus principalement pour bloquer les attaques d'inondation et sont la plupart du temps inefficaces contre les attaques qui exploitent la logique des protocoles, réalisées à partir des *botnets*, machines dont le comportement est la plupart de temps légitime.

CHAPITRE 3

Démarche du travail de recherche

Le projet de recherche des stratégies dynamiques du délai d'inactivité a commencé avec mon projet de fin d'études (PFE) que j'ai fait sur ce sujet, dans le cadre de mon année d'échange à l'École Polytechnique de Montréal. Les fruits du PFE ont été l'idée d'utilisation d'un mécanisme dynamique de gestion du délai d'inactivité, un modèle mathématique limité pour modéliser les attaques de déni de service d'épuisement des ressources et d'une première version du simulateur stochastique. Ces résultats ont été publiées dans un rapport technique (Boteanu *et al.*, 2006). En rétrospective, la critique a apporter à ce modèle est que seulement la politique d'assignation du délai d'inactivité *deterministic* était modélisée et les résultats sont des valeurs approximatives et non pas des valeurs exactes, approximation dont la fiabilité n'avait pas été vérifiée à l'époque. De plus, les équations mathématiques étaient très complexes et pas optimisées, ce qui nous avait permis de calculer des résultats numériques pour des taux d'attaques de seulement 96 connexions par seconde et pour des tailles de la file du serveur de seulement 128 connexions.

Suite à mon retour à l'École Polytechnique de Montréal dans le cadre d'une maîtrise, dans un premier temps, nous avons apporté des corrections mineures au modèle mathématique. De plus, nous avons mieux formalisé les politiques d'assignation du délai d'inactivité que nous avons introduit : *deterministic* et *deferred*. Le simulateur stochastique, développé précédemment spécifiquement pour ce projet, a été aussi amélioré et englobe maintenant 46 classes JAVA pour un total de presque 3000 lignes de code. Suite à ces améliorations, des simulations avec un taux d'attaque de 65536 connexions par seconde contre un serveur avec une file de 8000 connexions ont été possibles. De plus, nous avons exploré un autre modèle d'attaque que celui où les paquets malicieux arrivent à des intervalles selon une distribution exponentielle : le modèle d'attaque en rafale (*burst*), où les paquets malicieux arrivent en rafales de taille déterminée et à des intervalles de temps constants. Le simulateur a été étendu pour implémenter ce nouveau type d'attaque et toutes les stratégies ont été testées contre le nouveau type d'attaque, en rafale, pour une file de taille 128 connections. Ces résultats ont fait l'objet de l'article publié à la conférence *Information Security Conference (ISC)* en octobre 2007 (Boteanu *et al.*, 2007a), présenté à l'annexe A.

L'étape suivante a été de vérifier expérimentalement si les résultats obtenus à l'aide du modèle mathématique et des simulations stochastiques étaient exactes et d'analyser le cout en terme de performance de telles mesures de protection dans un cas réel. Dans ce but, nous avons implémenté les stratégies de gestion dynamique dans une application temps réel qui résidait sur une composante tierce partie. Nous avons décidé de déployer cette application sur un ordinateur de bureau standard, mais nos algorithmes et le code source pourraient facilement être déployés sur un processeur réseau dédié. Nous avons ensuite analysé la performance des stratégies de protection dans l'environnement expérimental et nous avons comparé les résultats aux simulations stochastiques. Les deux modèles d'attaque, avec intervalles d'arrivée selon une distribution exponentielle et avec arrivées en rafale, ont été testées. Pour chaque simulation et expérience, nous avons effectué plusieurs essais afin d'obtenir une valeur moyenne statistiquement significative. Pour ce, nous avons mesuré et vérifié que les écarts type étaient suffisamment petits, ce qui a toujours été le cas. Tous ces résultats ont été publiés dans les actes de l'atelier *ACM Quality of Protection (QoP) Workshop*, organisé dans le cadre de la conférence *ACM Communications and Computer Security Conference (CCS)* en octobre 2007, présenté à l'annexe B.

Après la réalisation et la publication de ces travaux, nous avons voulu combler une des lacunes principales : le fait que le modèle mathématique offrait des résultats seulement pour la politique *deterministic*, et ceci avec des erreurs importantes. Pour cette raison, nous sommes retournés sur les concepts de modélisation et nous avons raffiné la technique de couplage du trafic légitime avec le trafic malicieux dans le modèle mathématique. De plus, suite à l'analyse détaillée de la politique d'assignation du délai d'inactivité *deterministic*, nous sommes arrivées à la conclusion que le fonctionnement de cette politique dépend de l'état passé de la file. Cette politique ne peut pas donc être résolue avec un modèle de chaines de Markov autre que par des approximations, que nous avons introduites. Nous avons aussi modélisé la politique *deferred*, qui n'avait pas été étudiée théoriquement avant, mais cette politique dépend aussi du passé, et des approximations ont été introduites pour résoudre le modèle mathématique. Étant donné que pour résoudre les politiques *deterministic* et *deferred* nous avons dû utiliser des approximations, nous avons défini encore une autre politique d'assignation du délai d'inactivité : la politique *Poissson*. Cette politique est plus facile à modéliser, et nous avons calculé les valeurs exactes des probabilités de remplissage de la file à état stationnaire pour cette politique. Encore une autre amélioration a été apportée au modèle mathématique qui consiste dans l'utilisation de la représentation récursive de la formule Erlang-B et la simplification des équations, ce qui nous a permis d'explorer théoriquement des scénarios avec des taux d'attaques et capacités similaires à ceux des simulations et expériences, car ainsi les équations étaient devenues suffisamment simples pour être évaluées numériquement de façon rapide sur une plus grande plage de valeurs. Parce que les résultats théoriques sont calculés à partir de l'état stationnaire de la chaine de Markov, nous avons analysé le temps de convergence de la chaine théoriquement à l'aide l'adaptation de l'approximation *Modiffied-Offered-Load* (MOL), ainsi que par des simulations stochastiques, afin de valider la précision de l'utilisation de résultats basés sur ces états stationnaires. Nous avons également fait une analyse détaillée du choix des paramètres de configuration optimaux pour les stratégies de protection pour arriver à des conseils pratiques qui puissent être appliques immédiatement dans l'industrie. De plus, nous avons étudié la robustesse des stratégies de protection à des variations du taux du trafic d'attaque mais aussi du taux du trafic légitime. Cette étude confirme le bénéfice d'utiliser une stratégie dynamique de gestion du délai d'inactivité. Finalement, pour rendre plus facile l'implémentation d'une telle stratégie de protection contre une attaque autre que *SYN-Flood*, nous définissons un protocole abstrait de communication qui correspond au modèle mathématique déjà existant. Ensuite, nous montrons comment ce protocole peut être instancié pour modéliser des attaques de plus haut niveau, par exemple une attaque d'inondation avec des requêtes HTTP et une attaque d'inondation avec des demandes de réservation des places d'un événement culturel. Ces résultats ont été récemment soumis à publication à la revue scientifique *International Journal of Information Security* (publiée par Springer) et font l'objet du chapitre 4 de ce mémoire par articles.

CHAPITRE 4

Article 1: An Exhaustive Study of Queue Management as a DoS Counter-Measure

4.1 Introduction

Denial-of-Service (DoS) Attacks have been and continue to be one of the most insidious threats on networked computer systems. Over the years, *crippling* DoS attacks exploiting vulnerabilities in protocols of software and achieving phenomenal results with little or no resource investment by the attacker have become less and less common. Instead, they have been replaced by *flooding* DoS attacks where a moderate amount of resources are invested by the attacker in order to create a vastly superior consumption of resources on the targeted system. Protecting against flooding DoS attacks can be particularly difficult and frustrating. At the heart of this difficulty is the presence of a constant compromise or trade-off between providing services to legitimate users of network services, while keeping malicious users at bay. In particular, counter-measures aimed at reducing the presence and effect of malicious users impact negatively the Quality of Service (QoS) experienced by the legitimate users of the system.

Very generally speaking, the research and development efforts in DoS attack protection can be divided into two broad categories: a) defensive measures that try to detect, identify and block malicious uses of the system, and b) those that try to alter the trade-off between the resources expended by the attacker and the defender, to the advantage of the latter. The protective counter-measures work described and analysed in this paper belongs in the second category.

Of course, there are several different types of resources that attacker and defender can expend in a DoS attack. On the defender's side, inordinate consumption of the memory or CPU time of the server(s) directly providing a service, or even of those intermediate proxies and servers providing other necessary subsidiary services (including protection), can have significant impact on QoS. On the network side, consumption of large portions of the available channel bandwidth can have similar quality-decreasing effects, whether it is by directly occupying bandwidth with malicious traffic, or by reducing the channel capacity by targeting network equipment or in-channel protective network appliances (firewalls, proxies, etc.).

While attacks often have significant simultaneous effect on several resources types, they are normally designed with one single target resource in mind. One of the most infamous such resource-specific attack is the *SYN-flood* attack. It is without doubt the *Mother of all DoS attacks.* In a nutshell, it consists in flooding the memory that the targeted server allocates for the TCP/IP stack by forcing it to expend all of the available slots in its TCP half-open connection table (a.k.a. TCP backlog queue), i.e. those for which a SYN packet has been received but for which no ACK packet has been received yet. The significance of SYN-flood attacks is first and foremost historic, as it has traditionally been the workhorse of large-scale distributed DoS attacks in the wild. While it is arguably not optimal for the attacker in terms of ultimate impact on the target, one of the reasons of its success is the fact that very few attacker resources need to be expended in order to mount a successful SYN-flood. This is due to the fact that in most circumstances source IP addresses in the SYN packets can be spoofed (i.e. not correspond to the originating machine); the attacker vs. defender resource is thus very advantageous to the attacker, no matter what the impact for the defender is. This situation is somewhat atypical and particular to SYN-flood, which is why we introduce the notion of a more general *connection depletion* attack, in which the targeted resource are slots representing "active" connections in some abstract connection-tracking table. This general paradigm applies in principle to several network and application layer protocols with SYN-flood just being one example amongst many possible flooding attacks of this type. The research presented in this paper concentrates on these connection-depletion attacks.

For QoS and network engineering reasons that predate and go beyond the need for protection against connection-depletion DoS attacks, protocol designers and application developers have included and implemented in their design various mechanisms for managing the queues containing information about active connections. One of the most important and prevalent such queue management mechanisms is the assignment of timeout values to individual connections, combined with a queue polling policy that removes connection entries that have timed out.

At first, it seems intuitive that such a mechanism would provide some level of DoS protection. Indeed, malicious connections that may have made their way to the queue and have been abandoned there, will timeout more often that legitimate connections attempts that make it to the queue. Thus, lowering or adaptively modifying the timeout would seem to help. In particular, having a dynamic timeout, where the timeout is high when the server queue is empty and the timeout is low when the server queue is full seems like a good strategy. While this is definitely not a new idea in itself (versions of it have even been applied to the TCP/IP stack of some commodity Operating Systems), relatively little attention has been paid to the quantitative analysis of how effective this intuitive idea really is against connection-depletion attacks. In particular, there are several questions about the parameters that should be chosen (queue sizes, timeout values) and how these queue management policies should be implemented.

In previous work, we sought to partially address this issue by combining both mathematical models and simulations (Boteanu *et al.*, 2007a), with laboratory experiments (Boteanu *et al.*, 2007b). With this in mind, we considered three types of methods for adjusting timeout values: the *fixed* timeout method, and the *threshold* and *linear* timeout adjustment methods. Furthermore, we considered two policies for enforcing the timeout and removing connections from the queue, the *deterministic* and the *deferred* timeout assignment policies.

This paper is a revised and detailed compendium of the work cited, and extends previous results in several ways. First, we extended and improved our mathematical analysis of the Markov chain models previously developed. This now allows us to more efficiently generate theoretical predictions, for a wider range of possible queue management parameters than previously possible. In addition, we extended the mathematical analysis to the deterministic policy, and to a newly introduced one, the *Poisson* timeout assignment policy. We also added a study of the convergence rates of the modelled Markov chains, in order to validate the usability of the steady-state approximations used throughout our analysis. Finally, our most significant contribution concerns our study of the robustness of the various solutions considered with respect to non-optimal choices of the parameters. In other words, we study and describe the sensitivity of the queue management strategies to poor or suboptimal choices of the timeout parameters and to varying legitimate and attack traffic rates.

The rest of this paper is organised as follows: in Section 4.2 we provide an overview of previously related work. We then introduce the connection depletion attacks and the dynamic timeout strategies in Section 4.3. In Section 4.4 we use Markov chains to model the previously described attacks and protection strategies only to validate the model by stochastic simulations in Section 4.5. With all the pieces in place, we proceed to measuring the performance of the dynamic timeout strategies in laboratory experiments in Section 4.6. Finally, we conclude in Section 4.7 by sum our findings and providing directions for future work.

4.2 Previous work

In this section we provide an overview of previous work related to DoS. This topic has been covered before in detail in several review articles and books. In terms of generic introductions to the subject, the book *Internet Denial of Service: Attack and Defence Mechanisms* (Mirkovic *et al.*, 2004) is aimed at helping network administrators understand attacks and how to act when faced with them. Even an RFC, RFC4732 (Handley et Rescorla, 2006) has been written to provide an introduction on DoS attacks to protocol designers. Finally, extensive taxonomies of DDoS attack types and counter-measures have been described by Mirkovic et Reiher (2004) and Douligeris et Mitrokotsa (2004). We refer the reader to any of the above work for more comprehensive reviews of the topic. The review of previous work covered here is oriented towards the discussion of connection depletion attacks, and as such we concentrate on efforts to model, detect and protect against them. We only cover quickly some of the other related approaches such as collaborative defences, which are not necessarily specific this kind of attack.

In principle, it is always possible to force the targeted server to expend all available resources up to the point where it becomes inoperable. However, this could come at a certain non-negligible cost for the attacker. In some cases, it is more interesting to consider the effect on the QoS as a function of the effort spent by the attacker. Alternatively, one can think of the required level of defence resources that must be spent, in order to maintain a given level of QoS, when the attacker mounts an attack of a given strength. These trade-offs between attacker vs. defender resources were first studied and formalised by Meadows (1999, 2001). This framework is, however, not sufficient for measuring the degree to which a protocol is vulnerable to DoS attacks because of the difficulty of providing concrete cost values for elementary operations (e.g. blocking a message, computing a digital signature).

Another model has been proposed where continuous Markov chains are used to analyse the performance of hard-state vs. soft-state protocols (Lui *et al.*, 2004). A protocol is considered to keep hard-state if the timeout is only used as a fail-safe mechanism. The hard-state protocols, although better performers in perfectly defined conditions, are more vulnerable to attacks and network fluctuations than the soft-state protocols.

SYN-flood being the most exploited amongst connection depletion attacks, several counter-measures have been devised to offer protection specifically to this attack. These measures try to tip the trade-off between the attacker and defender resources so that it is not worthwhile for attackers to try to inflict damage with this particular attack any more. *Client puzzles* (Juels et Brainard, 1999) and their network layer extension, *network puzzles* (Feng *et al.*, 2005) consist in requiring the client to solve a cryptographic puzzle before the service is offered. Apart from the challenge of making the new protocol adopted widely, the major drawback of this method is the negative impact that the heavy computation has on legitimate clients which is even more substantial if the clients are mobile (laptops or mobile phones). Another approach to solving the SYN-flood problem is to render the three-way handshake stateless for the server, by storing state information in the TCP sequence number, method that is referred to as *SYN cookies* (Bernstein, 2003; Zuquete, 2002). However, the TCP sequence numbers do not allow for all the TCP options to be encoded and the retransmission of unacknowledged messages, as required by the TCP protocol, is not possible in this context. To cope with these limitations, a lighter version of the SYN cookies, called *SYN cache* has been suggested (Lemon, 2002), where some state information is kept on the server and the rest is encoded in the TCP sequence numbers. Although this might resolve the issues of the SYN cookies, this is only gives a small advantage to the server, because memory is still allocated for connections.

The timeout mechanism is implemented in TCP as a fail-safe measure. The intuition is that when using a lower timeout, the server will be more resilient to attacks but tougher to use for slow legitimate clients. If the timeout value cannot be configured on the server for some reason, a method of controlling the timeout remotely by a third-party device has been proposed (Schuba *et al.*, 1997; Nakashima et Oshima, 2006; Nakashima et Sueyoshi, 2007). This device would sniffs the network and send RST packets to the server. It has even been suggested that firewalls should be used as TCP proxies (Schuba *et al.*, 1997; Ohsita *et al.*, 2005) for the purpose. From the attacker point of view, this is equivalent with moving the attacked queue from the server to the firewall with the side-effect that if the firewall queue is flooded successfully, none of the protected servers will operate anymore. In addition, this method opens the door to a table saturation attack as identified in the TCP proxy implementation by Checkpoint (Noureldien et Osman, 2000). Because a low timeout might have a negative impact on slow legitimate clients, Microsoft (Microsoft TechNet, 2003) implemented a method where a low timeout is used only during an attack. Nevertheless, the questions of whether lowering the timeout has an impact on legitimate clients, how to measure this impact, what the optimal timeout values are and what is the best method for lowering the timeout have not been answered yet. This is where we focus our efforts.

Other counter-measures consist in trying to evade the attack traffic by implementing some measures that the attackers do not expect, for example intentionally dropping the first SYN packet (Al-Duwairi et Manimaran, 2006) or redirecting legitimate clients to a different server using a HTTP redirect message. Needless to say, during a targeted attack these counter-measures can be very easily identified and the attack can be configured to bypass them. In a different perspective, a secure TCP protocol was suggested (Ghavidel et Issac, 2007) that would replace the existing TCP protocol which would have the clients authenticate before establishing connections. Given the necessary efforts needed for the migration of the IP protocol to IPv6, it is not reasonable to believe that a secure TCP protocol would be adopted at large scale in the near future.

Given that the protection against SYN-flood is such a difficult problem, some were content with detecting such an attack in the first place. In this sense, is was shown that the number of used slots in the queue is a good SYN-flood attack indicator (Khan et Traoré, 2005). Counting the number of SYN packets and comparing them with absolute thresholds (Tartakovsky *et al.*, 2006; Siris et Papagalou, 2004), the overall number of TCP packets (Shin *et al.*, 2005) or with the number of specific FIN (Wang *et al.*, 2002) or SYN/ACK packets (Divakaran *et al.*, 2006) could also provide a measure of the attack intensity. However, the performance of these types of detection algorithms varies depending on the traffic parameters (Beaumont-Gay, 2007).

Another attack at the TCP level that captured recent attention is the low-rate TCP-targeted attack which consists in blocking the TCP flow at specific times so that the TCP congestion mechanism is activated (Kuzmanovic et Knightly, 2003; Yang *et al.*, 2004; Shevtekar *et al.*, 2005; Dong *et al.*, 2006). Another low rate attack type has been suggested for servers handling connections in a serial fashion, which consists in sending packets at specific times to prevent legitimate connections from entering the queue (Maciá-Fernández *et al.*, 2007). These low-rate attack types are somewhat similar to the resonance effect that we will describe in Section 4.6.4. The attack we deal with is different however, because in our case the server is able to handle connections in a parallel fashion and the attack does not rely on blocking network traffic.

A complete different protection approach to modifying the trade-offs consists in discriminating in some manner the attack traffic and filtering it entirely. First, because most of the SYN-flood attacks make use of spoofed IP addresses, various methods have been developed to detect the use of this technique, for example the recording of the entry and exit edge routers addresses (Varanasi *et al.*, 2004) or the Hop-Count Filtering (Jin *et al.*, 2003; Zou *et al.*, 2006). Other counter-measures rely on statistical methods to detect unusual traffic profiles, based on source address distribution (Feinstein *et al.*, 2003), TCP packets arrival times (Cheng *et al.*, 2002), or specific TCP packets relative frequency (e.g. SYN vs. FIN) (Kim *et al.*, 2006; Ayres *et al.*, 2006). Propositions have been made for implementing such statistical detection mechanisms on network processors (Lim et Uddin, 2005). Also, a method for determining whether the server did not receive the ACK response packet due to network congestion was suggested by probing the network and analysing the delay in the response (Xiao *et al.*, 2005). Another suggestion has been made where edge-routers would observe the number of incomplete handshakes and block the sources that generate too many of them (Bellaïche et Grégoire, 2007). It is important to note that all the counter-measures that employ traffic discrimination rely on the fact that the malicious traffic has at least one distinctive characteristic from the legitimate traffic. Unfortunately, attackers nowadays make use of *botnets*, large networks of controlled machines on the Internet, in order to launch the attacks. In this scenario, it is futile to try to discriminate the traffic because the malicious machines can easily mimic the behaviour of legitimate machines.

Finally, it has been suggested that due to the distributed architecture of the Internet, only distributed counter-measures could offer global protection against DDoS attacks. The first effort in this direction is the RFC2267 (Ferguson et Senie, 1998) which specifies that ISP should filter all outgoing traffic with addresses not within their network, measure that would block the IP spoofing technique. Going a step further in this direction, it is thought that the ISP should implement more sophisticated detection and filtering mechanisms on their routers to block attacks coming from their network (Chen et Song, 2005). DefCOM (Mirkovic *et al.*, 2002, 2003; Robinson *et al.*, 2003; Oikonomou *et al.*, 2006) is an overlay mesh proposition in which network equipments on the Internet communicate on a secure channel with the purpose of sharing attack information and lowering attack traffic priority. Other suggestions have been made (Yang *et al.*, 2005) where the target would send feedback to the routers (called *capabilities*), on whether certain flows are to be given priority. Attack traffic would still get through, but not having obtained associated capabilities, would be be routed with lower priority. The use of capabilities along with long-term client reputation can be then used for distinguishing between legitimate and malicious clients (Natu et Mirkovic, 2007). The SOS architecture (Keromytis *et al.*, 2004) consists in building a secret routing network using cryptographic hash functions but this has the drawback of a significant latency increase. Other methods which have to be deployed on routers consist in identifying from which upstream routers does the attack traffic come from, and blocking or limiting these routers (Ioannidis et Bellovin, 2002; Zhang et Dasgupta, 2003; Yau *et al.*, 2005). Another method requiring edge router collaboration consists in having the entry routers send ARP requests to clients initiating communications in order to prevent IP spoofing (Chouman *et al.*, 2005). Unfortunately, seen that for political and social reasons, the RFC2267 has not yet been adopted world-wide in spite of its simplicity, it is very unlikely that other more complex architectures like DefCOM or SOS are to be deployed on the Internet in the near future. Furthermore, the collaborative protection measures to date have focused on blocking bandwidth consumption flooding attacks and are most of the times inefficient against connection depletion attacks.

In summary, we have seen that there are several types of counter-measures aimed at solving connection-depletion attacks. However, these solutions are aimed at dealing with one particular attack in this category which in most cases is the SYN-flood attack. Unfortunately, these SYN-flood specific solutions might not work on higher level flooding attacks. This is particularly worrying, as in recent DoS attacks, SYN-flood accounted for a relatively low percentage of the attack footprint (Nazario, 2007). In particular, a recent case documents the relatively new use of flooding attacks at higher levels of the protocol stack (Adair, 2008), in this case HTTP. Hence the development of analysis tools and defensive approaches against generic connection depletion attacks is needed.

Modifying the logic of the existing protocols or replacing them with new, more secure ones does not seem to be a feasible option because of the huge migration effort involved. Also, in theory, collaborative counter-measures seem to be very efficient against bandwidth flooding attacks but their wide adoption is questionable for non-technical, social and political reasons. If the malicious traffic has distinctive characteristics from the legitimate traffic, it can can be detected and filtered. However, in practice, the traffic filtering techniques can be evaded when attacks are generated by using botnets. It seems that one of the few options left is thus to modify the internal implementation and parametrisation of protocols and applications on the targeted servers, so that they can be made more resilient to attacks. This last method is of particular interest because it is complementary to the use of other filtering or collaborative upstream protection mechanisms that might offer some level of protection.

4.3 Dynamic timeout strategies

In this section we define what attack types we are trying to protect the server from, and describe the dynamic timeout strategies that might protect against them. In order to provide a solution as general as possible, we first define an abstract protocol that we will use for modelling and analysis and show how real-life protocols like TCP can be instantiated from this abstract protocol to model attacks like SYN-flood or TCP connection establishment flooding.

4.3.1 Abstract protocol description

Let us consider an abstract protocol defined as usual by a finite state machine, where the following events generate transitions between the protocol states:

- *connection arrival*, an entry event for every connection, meaning that the server received a connection request; If the server has enough resources, it will accept the connection by generating a *connection acceptance* event internally and then proceeding with serving the connection. Otherwise, a *connection rejection* event is generated.

- *connection rejection*, an exit event, meaning that due to insufficient resources, the server dropped the connection upon arrival. No further tracking is made of the connection.
- *connection completion*, an exit event, meaning that the client received the required service and that the connection was "closed" gracefully.
- *connection expiration*, an exit event, meaning that the server tried to offer the required service but the client did not respond in a timely manner and hence the connection was dropped. No further tracking is made of the connection.

The protocol state diagram illustrating possible states: connection arrived, connection rejected, connection accepted, connection expired and connection completed as well as the above-mentioned events generating transitions between the states is shown in in Figure 4.1.

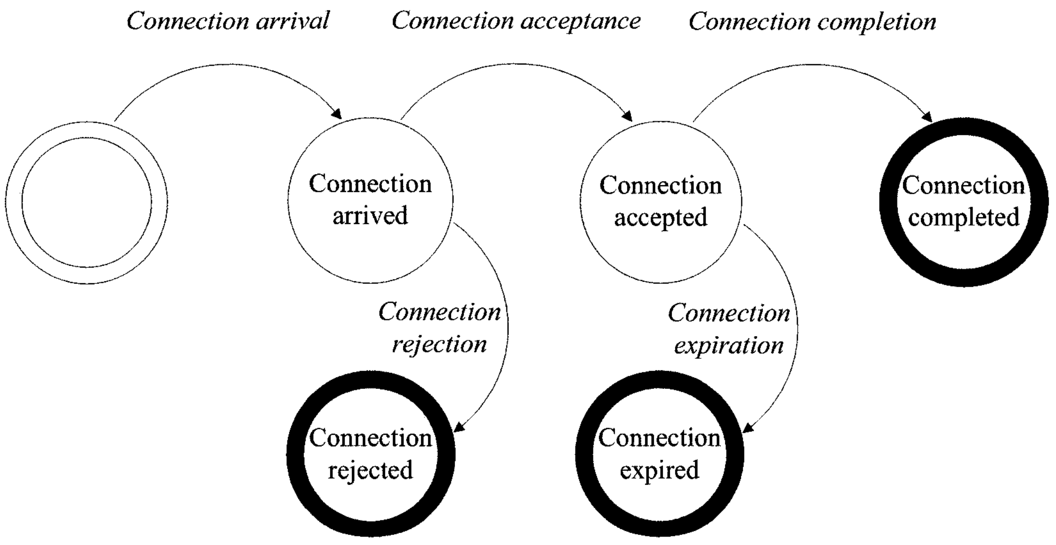

Figure 4.1 State diagram of the abstract protocol

Let us now consider under which circumstances these events can occur, in the context of legitimate and malicious of the protocol (e.g. for mounting flooding attacks). The *connection arrival* event occurs when either a legitimate or a malicious connection arrives, i.e. *connection arrival* = *legitimate connection arrival* $\vee$ *malicious connection arrival.* A connection is considered served if it was accepted by the server and it completed or expired. Hence, *connection service* = *connection completion* $\vee$ *connection expiration.* Again, we make a distinction between the legitimate and malicious service events, i.e. *connection service* = *legitimate connection service* $\vee$ *malicious connection service.* A connection is successful only if the *connection completion* event is generated. Otherwise, the connection is considered to have failed, i.e. *connection failure* = *connection rejection* $\vee$ *connection expiration.*

Usually, the service time is mostly due to the messages travelling back and forward between the server and the client, on the Internet. The only significant resource that the server uses is memory, and this, in order to keep track of the connection states. In this scenario, the connections can and will be served independently and in a parallel fashion. This might not be the case, however, when the considered server is a heavily used router with limited bandwidth or a web server with very slow disk speed. In the latter case, the connections would be served serially and/or influence each others service times.

4.3.2 SYN-flood attack

We now instantiate the abstract protocol model described earlier for the transport level protocol TCP, to capture the behaviour of the SYN-flood attack. The *connection arrival* event corresponds to SYN messages being received by the server. If the server backlog queue is full, the connection is rejected, which corresponds to the *connection rejection* event. If the backlog is not full, the server adds the connection to the backlog by generating the *connection acceptance* event internally. Then, the server proceeds to service by sending the SYN-ACK message to the client. If the client replies with an ACK message in a timely manner, the TCP handshake is successful, which corresponds to the *connection completion* event in the abstract protocol model. However, if the client does not respond with an ACK message, the server repeats the SYN-ACK message several times and then drops the connection by generating a *connection expiration* event internally. Figure 4.2 illustrates a possible sequencing of messages that would trigger the different events at the server.

One of the reasons SYN-flood is so attractive to attackers is because the default queue size c and timeout T_{out} values in modern OS are very permissive. Constructors,

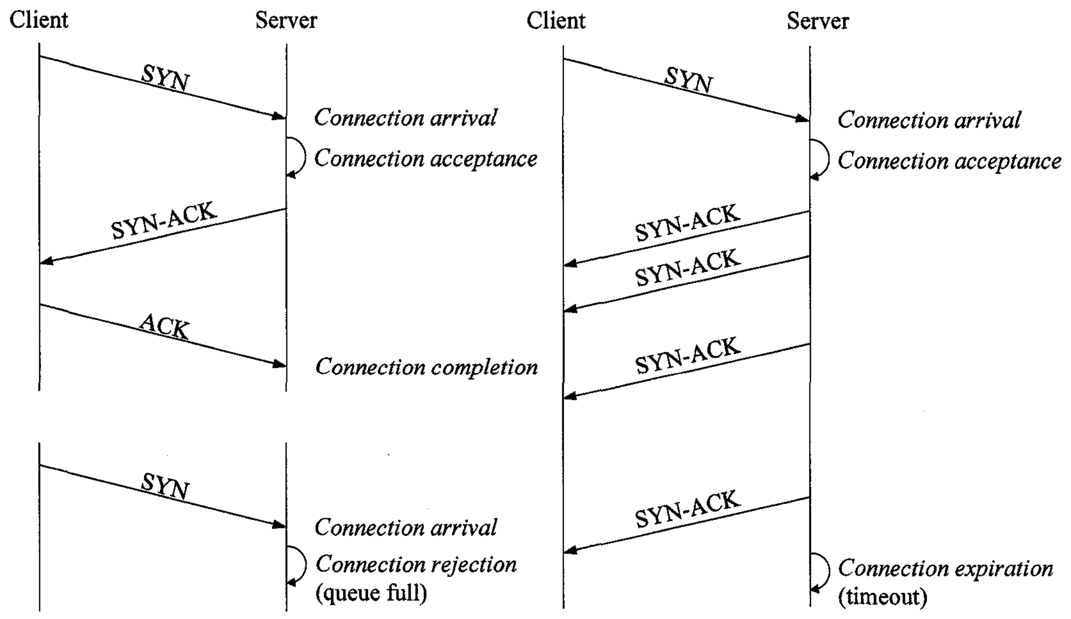

Figure 4.2 Abstract protocol instantiation for the TCP 3-way handshake

however, often suggest tweaking these parameters to harden the OS. The minimum SYN packets arrival rate λ_{m} that an attacker would have to generate to completely fill the queue of a server configured with the default parameters is illustrated in Table 4.1 (note that cnx=connections).

4.3.3 TCP connection establishment attack

Similarly to the previous section, we instantiate the abstract protocol model for a generic application level protocol that uses TCP as a transport layer. We do this in order to capture the behaviour of an attack that would try to flood the TCP established-connection queue. In real life, this upper-level protocol could be HTTP,

Table 4.1 Default queue size c and timeout T_{out} values in popular operating systems and minimum attack rate λ_{m} that fills the server queue

OS	c [cnx]	T_{out} [s]	λ_{m} [cnx/s]
Windows 2003	1000	45	22.2
Linux 2.6	1024	180	5.7
HP-UX 11.00	500	75	6.7
Solaris 10	128	180	0.7

SSH or any other connection-oriented protocol. The *connection arrival* event corresponds to the successful establishment of the TCP connection, due to the three-way handshake. If the server does not have enough application-level working threads to serve the connection, the *connection rejection* event is generated, usually by sending a RST message to the client. If the connection is accepted, the server starts a working thread and serves the client with the information required. At the end, the client can generate a *connection complete* event at the server side by closing the TCP connection gracefully. If on the contrary, the client does not respond to the ACK messages that the server sends, the server generates an *connection expiration* event internally and drops the connection. A possible sequencing of messages that would generate the above described events is illustrated in Figure 4.3.

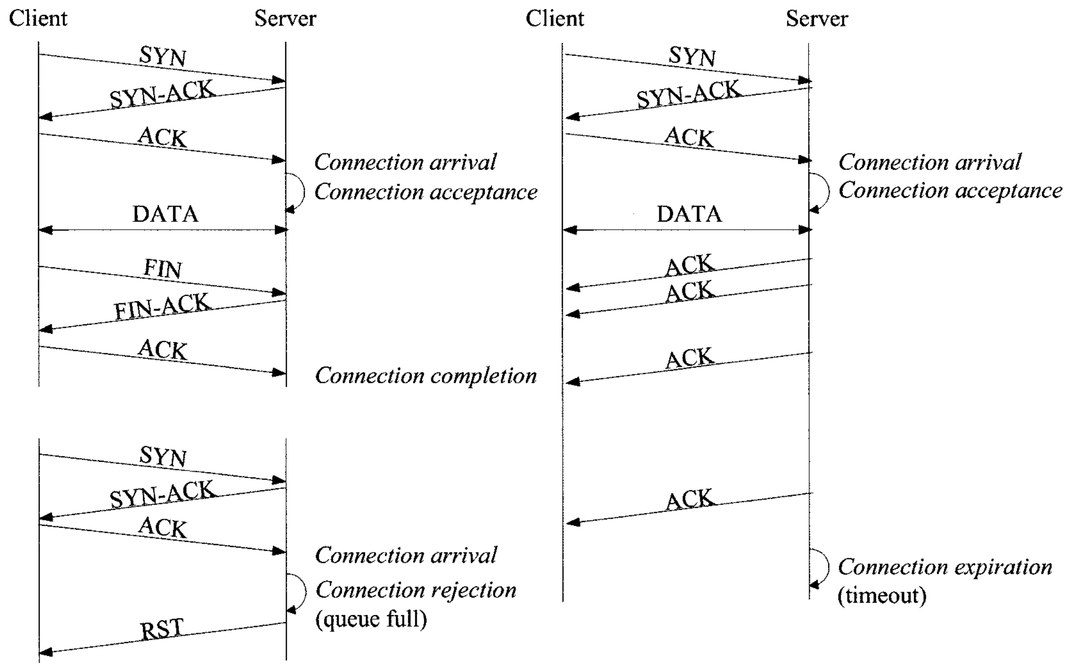

Figure 4.3 Abstract protocol instantiation for TCP connections life-cycle

Table 4.2 illustrates the default queue size c and the timeout T_{out} values in some of the commonly used server applications on the Internet as well as the minimum malicious arrival rate λ_{m} that an attacker has to generate to completely fill the server queue. The timeout values indicated in Table 4.2 are enforced at the application level but depending on the OS that is being used, lower timeout values could also be enforced at the transport level. Furthermore, an attacker could try to hold the application level connection by sending keep-alive messages regularly, depending on the protocol that is being attacked.

Let us compare the amount of memory used by the connection queue during the SYN-flood and the TCP connection establishment flooding. Each used slot in the SYN-flood attack protocol model, corresponding to a half-open TCP connection, requires roughly 100 bytes. On the other hand, each slot in the TCP connection establishment flooding protocol model, corresponding to a successfully completed TCP connection, requires between 1.5 kB and 16 kB at the TCP level, depending on the TCP implementation and configuration. More memory may be allocated at the application level protocol that is flooded.

4.3.4 Ticket reservation flooding attack

We take the previous examples of resource exhaustion attacks one step further and we illustrate how the same model can be instantiated to capture the behaviour of the server during a higher-level attack, in this case against a ticket reservation application on the Internet. As we did in the previous section, we illustrate a possible sequencing of messages that could generate the events considered in the abstract protocol in Figure 4.4. In this case, the *connection arrival* event corresponds to a client deciding to buy a ticket by clicking on the *buy* button. If there are places still available for the selected date and event, a place is temporarily reserved for the client, by generating the *connection acceptance* event internally. A web page is then sent to the client inviting it to complete the payment. If the payment is completed in a timely manner, the *connection completion* event is generated and the reserved place is permanently assigned to the client. However, the reservation is only held by the application for a limited amount of time, usually several minutes. If the client does not complete the payment during this interval of time, a *connection expiration* event is generated

Table 4.2 Default queue size c and timeout T_{out} values used in some popular server applications and minimum attack rate λ_{m} that fills the server queue

Application/Protocol	c [cnx]	T_{out} [s]	λ_{m} [cnx/s]
Apache 2.0 / HTTP	150	300	0.5
IIS 6.0 / HTTP	8000	120	66.7
IIS 6.0 / FTP	100000	120	833.3
Cisco SIP Proxy 2.0 / SIP	20	0.005	4000.0

and the temporary reservation is discarded, making the place available to other users. For example, if for a specific event there are 10000 available places (c) and each user is allowed 5 minutes (T_{out}) to complete the payment, a malicious user generating 34 reservation requests per second (λ_{m}) would block all the 10000 places, for as long as it would want.

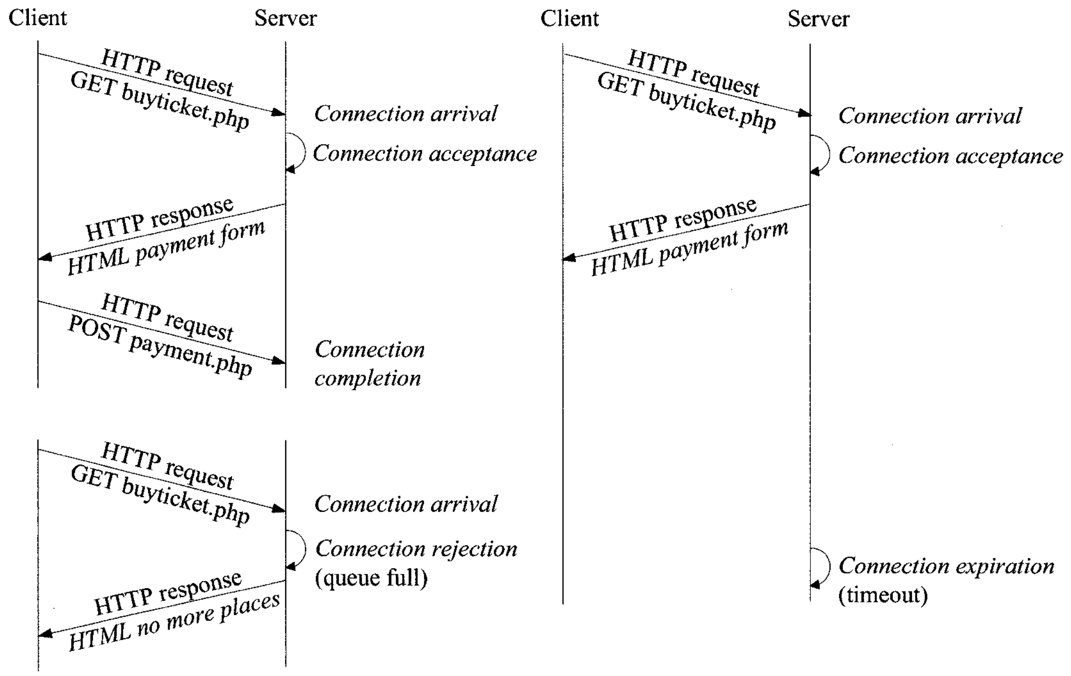
Client
Server
HTTP request GET buyticket.php
Connection arrival
Connection acceptance
HTTP response HTML payment form
HTTP request POST payment.php
Connection completion
HTTP request GET buyticket.php
Connection arrival
Connection rejection (queue full)
HTTP response HTML no more places
Client
Server
HTTP request GET buyticket.php
Connection arrival
Connection acceptance
HTTP response HTML payment form
Connection expiration (timeout)

Figure 4.4 Abstract protocol instantiation for a ticket reservation attack

4.3.5 Legitimate and malicious traffic distributions

In this section, we discuss the assumptions and simplifications that we make on the distribution of arrival times of protocol transition events, i.e. the legitimate and malicious traffic distributions.

Legitimate traffic. We will consider that all legitimate incoming connections are generated by a Poisson process, i.e. the inter-arrival times are independent of each other and exponentially distributed. This assumption is made by other DoS related research (Mirkovic *et al.*, 2006; Cheng *et al.*, 2002) and is justified by the fact that connection arrivals, just like telephone calls, are triggered by humans acting independently. We are aware of the day of week and time of day legitimate traffic variations. However, we make the simplification that the legitimate traffic rate, λ_{l}, is constant within the time scale of queue management attack counter-measures. We will, however, analyse the sensitivity of the attack counter-measures to traffic rate variations in Section 4.6.6.

In most cases, the serving speed of legitimate connections, μ_{l}, depends only of network transit times but in some cases user interaction is also a factor. It has been shown that because of the network queueing algorithms, all the IP packet traffic tends toward a Poisson process as the load increases (Cao *et al.*, 2001). We will make the supposition that the network is heavily used and hence *connections completion* events are generated at exponentially distributed intervals of time from the corresponding *connections arrival* events, with speed μ_{c}.

Malicious traffic. While we model legitimate arrivals as a Poisson process, this is not general for attack traffic as the attacker is free to use whatever strategy it wants. In order to be able to build a simpler model that we could analyse mathematically, in part of our work, we make the simplification that the malicious arrival traffic is also generated by a Poisson process, with rate constant in time, λ_{m}. In Section 4.6.4 we analyse the robustness of this model by stochastic simulation and laboratory experiments for attacks with a different traffic distribution, where packets arrive in bursts of various sizes and at various intervals of time. Moreover, the malicious traffic might or might not be distinguishable from the legitimate traffic. For our measures, and without loss of generality, we consider that the malicious and legitimate traffic are indistinguishable. If a distinction between the two could be made, a filtering counter-measure might then be used to prevent all or some of the malicious traffic from reaching the server. Our method is complimentary to such a technique and can be used to handle all the *residual attack traffic*, unfiltered by upstream defences.

Concerning the malicious traffic service speed, μ_{m}, the strategy of the attacker is to exhaust the server resources using the smallest effort possible. This is achieved by generating the *connection arrival* events and then abandoning the communication without any notice to the server. Malicious connections will eventually all expire and generate *connection expiration* events at T_{out} intervals of time from the *connection arrival* events.

4.3.6 Timeout adjustment methods

Apart from increasing the queue size which is limited by the available memory, the only parameter than can be adjusted at the server side, is the timeout that a connection is granted before expiring when the client does not respond. Ideally, we would like to adjust the timeout on a per-connection basis, by setting a very low, restrictive timeout to malicious connections and a very high, permissive timeout to legitimate connections. Unfortunately, as discussed earlier, the server is unable to distinguish between legitimate and malicious connections. With this in mind, the most obvious indicator that can be used for adjusting the timeout is the number of connections in the queue. Therefore, the timeout adjustment methods that we analyse are:

A. The traditional fixed timeout method, where the timeout T_{out} is constant, regardless of the queue occupation. This is the classical method that is widely used in network equipments and protocol implementations.

B. The threshold method, where the timeout alternates between two fixed values, T_0 and T_1, as the number of connections in the queue crosses a predefined threshold. This adjustment method is not new and is already implemented in the TCP stack of some OS, e.g. Microsoft Windows Server 2003 (Microsoft TechNet, 2003).

C. The linear method, a straightforward generalisation of the former that we introduced in (Boteanu *et al.*, 2007a), where the timeout value is determined according to a linear function depending on the number of connections in the queue, with two predefined empty-queue T_0 and full-queue T_1 timeout values.

Figure 4.5 illustrates how the timeout varies as a function of the queue occupation for the fixed and dynamic methods.

4.3.7 Timeout assignment policies

We established in the previous section that the server is going to adjust the tolerated timeout value according to the queue occupation. However, we still have to define how these timeout changes are going to affect connections. The question that we need to answer is what happens to a connection if the timeout tolerated by the server changes while the connection is in the queue. To address this issue, we introduce two policies for assigning the timeout values to connections:

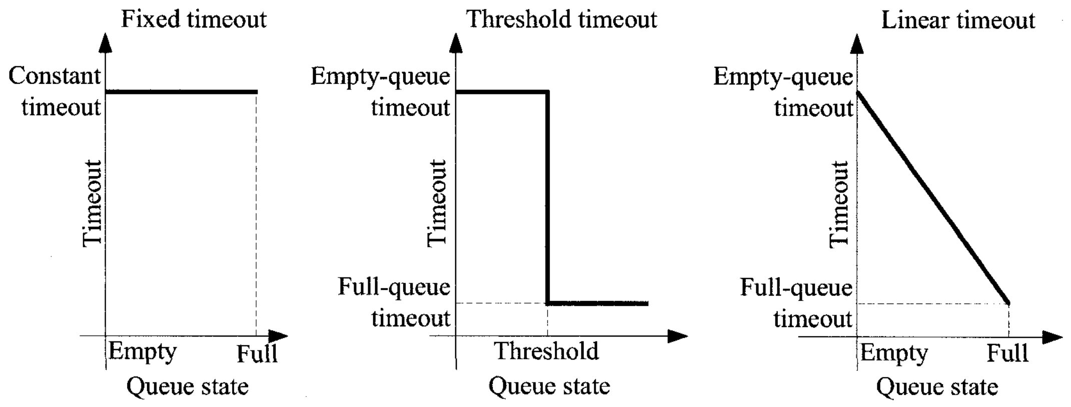

Figure 4.5 Fixed, threshold and linear timeout adjustment methods

I. The *deterministic policy*, where connections are assigned an expiration time at the moment where they enter the queue. If by the expiration time the connection has not left the queue, it is dropped. Because for each connection the past (queue occupation at arrival) is taken into account, this policy is not memoryless.

II. The *deferred policy*, where connections are dropped only if they have been in the queue longer than the current timeout. When the server transitions from a permissive state, with a high timeout, to a more restrictive state, with low timeout, it abruptly drops the tolerated time to complete to the low timeout value. This causes the oldest connection in the queue, that would expire under the new low timeout value, to be dropped instantly. We call this particular behaviour the *abrupt-tolerance-drop effect.* However, if the server transitions to the same state but from a state with the same low timeout or from a state with a lower timeout, no connection is dropped instantly after the transition because the server is at least as permissive as it was in the previous state. Clearly, the behaviour of the server depends on the previous state and thus does not behave as a memoryless process.

The deterministic and deferred policies take the timeout into account at the connection arrival and at the connection expiration, respectively. These policies could easily be implemented in an OS TCP/IP stack or on network hardware. However, they have the disadvantage that they are not memoryless, which makes them difficult to evaluate mathematically. For this reason, we introduce in this paper a third policy, who behaves in a memoryless fashion:

III. The *Poisson policy*, where connections are always dropped at the same rate, that depends on the queue occupation. This corresponds to a Poisson process because the history of the previous server states has no influence on the drop rate.

Intuitively, the Poisson policy would correspond to the server using "relativistic" clock speeds in each state for keeping track of the connection ages. For example, let us suppose that each connection is always allowed the same amount of time before expiring, say 1,000 seconds. In a particular state where the tolerated timeout value is T_{out}, the speed of the server clock would be such that these 1,000 seconds would pass in exactly T_{out} seconds of "queue time". In another state with $T'_{\text{out}} < T_{\text{out}}$ tolerated timeout, these same 1,000 seconds would pass "quicker", i.e. in T'_{out} "queue time" seconds. Although this policy satisfies the memoryless property, it does not have a straight-forward implementation, as opposed to the deterministic and the deferred policies.

When coupling a timeout *adjustment method* with a timeout *assignment policy* we obtain a *timeout strategy*. Note that for the fixed timeout method, all the assignment policies have the same effect, because the timeout never changes.

4.3.8 Performance metrics

We are now interested in defining the performance metrics that will allow us to evaluate the QoS perceived by the clients when using the dynamic timeout strategies. Depending on the communication type that is taking place between the client and the server, several measures can be considered, for example the delay, jitter and percentage of lost packets as well as the duration of transactions (Mirkovic *et al.*, 2006). However, in our case the connections are handled in parallel and in an independent fashion, the service time being due solely to the round-trip time (RTT). Hence, from a client perspective, the only significant measure for an individual connection is whether it completed of whether it failed, regardless if that was by expiration or rejection.

Ideally, we would like to measure the transient probability $\phi(t)$ that the connection arriving at time t at the queue fails, which is defined as the probability for the connection to be rejected $\phi_{\text{r}}(t)$ or to expire $\phi_{\text{e}}(t)$, i.e. $\phi(t) = \phi_{\text{r}}(t) + \phi_{\text{e}}(t)$, since these are disjoint events. Let us note with n_{ss} the number of connections that arrive before the queue is stable. Because the transient behaviour of the queue is difficult to predict when evaluating the dynamic timeout strategies analytically, we measure the average connection fail probability Φ for all connections i that arrive when the queue is stable, i.e. $i > n_{\text{ss}}$.

$$\Phi = \frac{\sum_{i=n_{\text{ss}}+1}^{n} \phi(\tau_i)}{n - (n_{\text{ss}} + 1)} \tag{4.1}$$

where τ_i is the arrival time of the i^{th} connection. In order for this value to be an accurate estimation of the probability of a connection failing, the queue must become stable rapidly, an assumption that we explore in more detail in in Section 4.5.3.

On the other hand, when evaluating the performance of the timeout strategies using stochastic simulations and network experiments, we measure the percentage φ of legitimate connections that fail during the attack and average this value over several runs.

Finally, although the average fail probability Φ and the percentage of failed connections φ are the natural measures to take because of their straight-forward decomposition into the expired and rejected components (i.e. $\Phi = \Phi_{\text{e}} + \Phi_{\text{r}}$; $\varphi = \varphi_{\text{e}} + \varphi_{\text{r}}$), when comparing the performance of different strategies, we will look at the complementary measures, $1 - \Phi$ and $1 - \varphi$, which represent the average success probability and the percentage of successful connections, respectively.

4.4 Mathematical model

In order to evaluate the previously defined performance metrics we model the queue of a server implementing the abstract protocol as a Birth-Death Markov chain with c states. Each state E_k represents that there are k connections in the queue. Let us analyse the timeout strategies in the light of the Markov chain model.

4.4.1 Fixed timeout method

The connection arrival rate is the same for all states E_k, i.e. $\lambda^{(k)} = \lambda$. As discussed earlier, we suppose that both legitimate and malicious arrivals are generated by independent Poisson processes. Hence, the overall arrival rate λ is equal to the sum of the legitimate and malicious arrival rates, λ_{l} and λ_{m}, respectively. Being the sum of two Poisson processes, the overall arrival process is a Poisson process as well, and does not depend on the server state:

$$\lambda^{(k)} = \lambda = \lambda_{\mathrm{l}} + \lambda_{\mathrm{m}}. \tag{4.2}$$

The speed with which a legitimate client would accept the service offered by the server and generate a *legitimate connection completion* event is μ_{c}. However, these type of responses that would arrive after the timeout elapses are ignored by the server with *legitimate connection expiration* events being generated instead, when the timeout elapses. Therefore, the probability distribution function (PDF) of the legitimate connection service time $G_{\mathrm{l}}(t)$ has the form of an exponential distributions for t smaller than the timeout T_{out} followed by a Dirac delta function $\delta(t)$ in T_{out}. The weight of the Dirac delta p_{le} is so that the CDF of the service time is 1 after T_{out} and represents the probability that a legitimate connection expires after entering the queue. The expiration has to arrive *after* the timeout, which causes the Dirac delta to be placed infinitesimally close to T_{out}, in T_{out}^{+}:

$$G_{\mathrm{l}}(t) = \begin{cases} \mu_{\mathrm{c}} e^{-t\mu_{\mathrm{c}}} & t < T_{\mathrm{out}} \\ 0 & \text{otherwise} \end{cases} + \delta(t - T_{\mathrm{out}}^{+}) p_{\mathrm{le}} \tag{4.3}$$

where

$$p_{\mathrm{le}} = \int_{T_{\mathrm{out}}}^{\infty} \mu_{\mathrm{c}} e^{-t\mu_{\mathrm{c}}} dt = e^{-T_{\mathrm{out}}\mu_{\mathrm{c}}} \tag{4.4}$$

The mean service time for legitimate connections is:

$$t_{\mathrm{l}} \triangleq \int_{0}^{\infty} t G_{\mathrm{l}}(t) dt = \frac{1 - e^{-T_{\mathrm{out}}\mu_{\mathrm{c}}}}{\mu_{\mathrm{c}}} \tag{4.5}$$

In order to exhaust the server resources, malicious clients never generate *connection completion* events. All the malicious connections that get accepted in the queue leave the queue at timeout by expiring, i.e. $t_{\mathrm{m}} = T_{\mathrm{out}}$.

Knowing the mean service time of both legitimate and malicious connections, we calculate the overall mean service time. During an interval of time Δt, out of the $\lambda \Delta t$ connections that arrive at the server, only a number a are accepted. These are

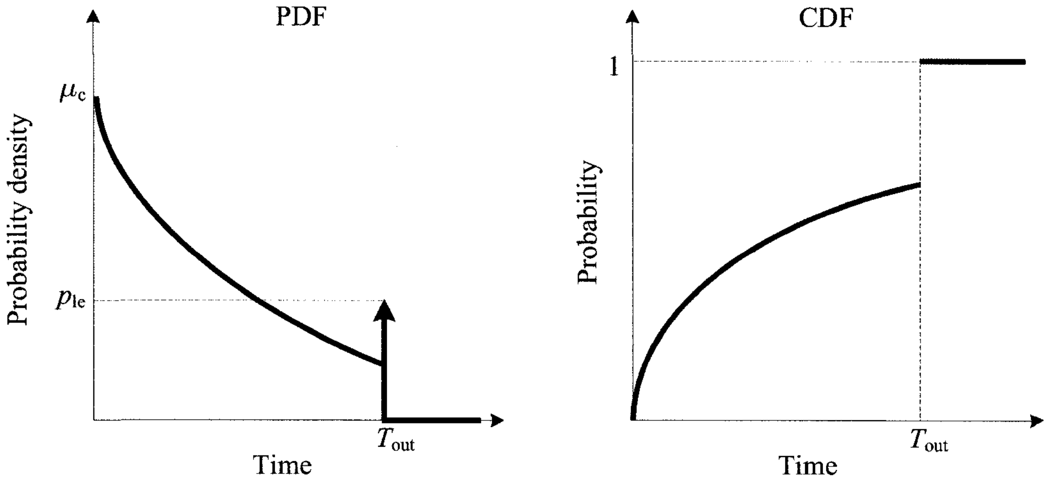

Figure 4.6 Legitimate service time PDF and CDF for the fixed timeout method

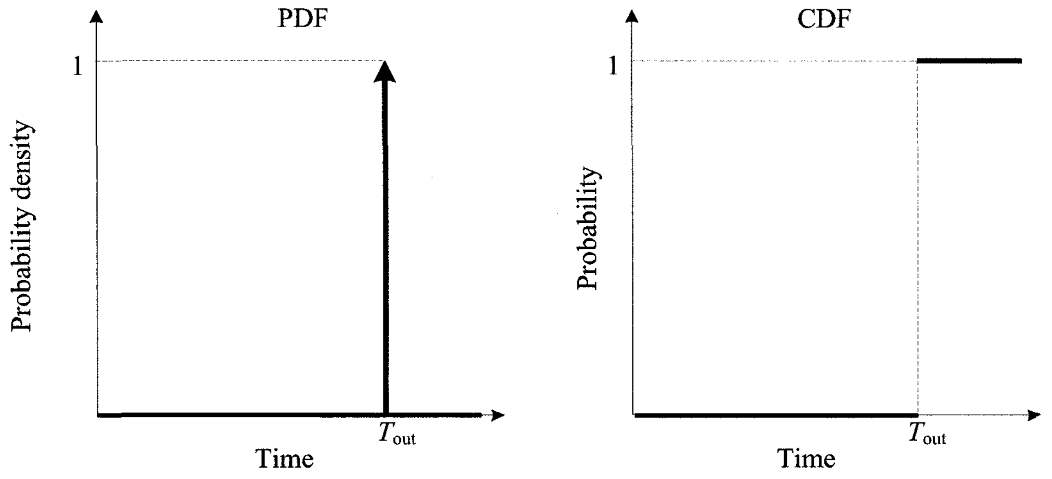

Figure 4.7 Malicious service time PDF and CDF for the fixed timeout method

the connections that will make a difference for the overall mean service time during Δt. Out of the a connections, a_{l} are legitimate and a_{m} are malicious. The proportion of legitimate and malicious connections that are accepted is equal to the proportion of legitimate and malicious connections that arrive at the queue, due to the arrivals being Poisson processes, i.e. $a_l = a\lambda_{\text{l}}/\lambda$; $a_m = a\lambda_{\text{m}}/\lambda$.

The mean service time $\tilde{t}$ during the interval Δt is equal to the average of the legitimate and malicious service times weighted by the number of legitimate and malicious connections that enter the queue during this interval.

$$\tilde{t} = t_{\text{l}}\frac{a_{\text{l}}}{a} + t_{\text{m}}\frac{a_{\text{m}}}{a} = \frac{\lambda_{\text{l}}t_{\text{l}} + \lambda_{\text{m}}t_{\text{m}}}{\lambda_{\text{l}} + \lambda_{\text{m}}} \tag{4.6}$$

The mean connection service speed ($\mu \triangleq 1/\tilde{t}$) is equivalent to the mean service rate for an individual slot:

$$\mu = \frac{\lambda_{\text{l}} + \lambda_{\text{m}}}{\lambda_{\text{l}}(1 - e^{-T_{\text{out}}\mu_{\text{c}}})/\mu_{\text{c}} + T_{\text{out}}\lambda_{\text{m}}} \tag{4.7}$$

The load of an individual slot ρ, defined as the arrival rate divided by the slot individual service rate, i.e. $\rho \triangleq \lambda/\mu$, becomes:

$$\rho = \lambda_{\text{l}}(1 - e^{-T_{\text{out}}\mu_{\text{c}}})/\mu_{\text{c}} + T_{\text{out}}\lambda_{\text{m}} \tag{4.8}$$

Because in the state E_k there are k slots that are occupied and all k connections in these slots are served independently in a parallel fashion, the overall service rate $\mu^{(k)}$ in this state is equal to the mean connection service speed μ multiplied by the number of connection in the queue k:

$$\mu^{(k)} = k\mu \tag{4.9}$$

The model we obtain is known as $M/G/c/c$ in Queue Theory and is illustrated in Figure 4.8. It is a Birth-Death Markov chain with Poisson process distribution arrivals (M), general distribution departures (G), capacity c and c slots that allow for connections to be served in parallel. This is a classical model for which the steady-state probability of the server to be in state E_k, meaning that k slots in the queue are in use $p^{(k)}$, is known:

$$p^{(k)} = p^{(0)} \prod_{i=0}^{k-1} \frac{\lambda^{(i)}}{\mu^{(i+1)}} \tag{4.10}$$

with

$$p^{(0)} = \frac{1}{\sum_{k=0}^{c} \prod_{i=0}^{k-1} \frac{\lambda^{(i)}}{\mu^{(i+1)}}} \tag{4.11}$$

When the server achieves steady state, it means that the probabilities that various server states are repeated will remain constant.

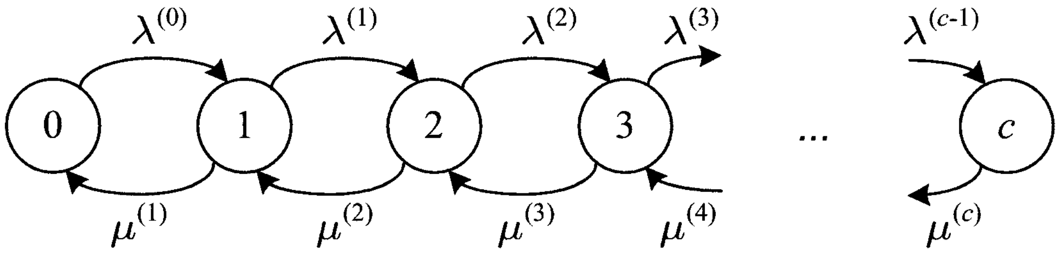

Figure 4.8 Markov chain queue representation for the fixed timeout method

The steady-state probability that a server queue with c slots is full is known as the *Erlang B loss function*:

$$B(c) = \frac{\prod_{i=0}^{c-1} \frac{\lambda^{(i)}}{\mu^{(i+1)}}}{\sum_{k=0}^{c} \prod_{i=0}^{k-1} \frac{\lambda^{(i)}}{\mu^{(i+1)}}} \tag{4.12}$$

Note that in our case, the probability of the server being in state E_c is equivalent to a server with c slots being full ($p^{(c)} = B(c)$) but the probability of the server being in state E_k is not equivalent to a server with k slots being full ($p^{(k)} \neq B(k)$). The previous equation can be expressed in a simpler recursive form (Jagerman *et al.*, 1997):

$$B(c)^{-1} = \frac{\mu^{(c)}}{\lambda^{(c-1)}} B(c-1)^{-1} + 1 \quad ; \quad B(0)^{-1} = 1 \tag{4.13}$$

We can now express (4.10) and (4.11), which are computational expensive to evaluate as a function of (4.12):

$$p^{(k)} = \frac{B(c)}{\prod_{i=k}^{c-1} \frac{\lambda^{(i)}}{\mu^{(i+1)}}} \tag{4.14}$$

Because the timeout is fixed, the arrival rate does not depend on the state k and the service rate depends linearly on the state k, as described in (4.9). We can thus solve the recurrence in (4.13):

$$B(c)^{-1} = \frac{e^{\rho}\Gamma(c+1,\rho)}{\rho^{c}} \tag{4.15}$$

where $\Gamma(a,z)$ is the upper incomplete gamma function, i.e. $\Gamma(a,z) = \int_z^{\infty} t^{a-1}e^{-t}dt$. We insert (4.15) in (4.14) and obtain:

$$p^{(k)} = \frac{\rho^{k}(c!/k!)}{e^{\rho}\Gamma(c+1,\rho)} \tag{4.16}$$

After steady state is achieved, the probability $p^{(k)}$ that the queue reaches the state E_k in the future remains constant and is described by (4.16). Because the arrivals follow a Poisson process and are thus independent, the average probability of connections that arrive at the queue after the steady state is achieved to find the queue in state E_k is equal to the steady-state probability of the queue being in state E_k. Finding the queue in state E_c is equivalent to the connection being rejected. Hence, the average probability of a connection that arrived after after steady state to be rejected Φ_{r} is equal to the probability of the queue being full at steady state:

$$\Phi_{\mathrm{r}} = p^{(c)} \tag{4.17}$$

Similarly, the probability of a connection that arrived after the steady state is achieved to expire Φ_{e} is equal to the steady state probability that the queue is not full and the connection is accepted $(1 - p^{(c)})$ multiplied by the probability that a legitimate connection that entered the queue would expire p_{le}:

$$\Phi_{\mathrm{e}} = (1 - p^{(c)})p_{\mathrm{le}} \tag{4.18}$$

As described earlier, the only significant performance measure is the probability that a legitimate connection that arrived after the steady state is achieved fails (Φ), which is equal to the probability that the connection that arrived after the steady state is rejected (Φ_{r}) plus the probability the connection that arrived after the steady state expires (Φ_{e}).

$$\Phi \quad = \quad \Phi_{\mathrm{r}} + \Phi_{\mathrm{e}} = \frac{\rho^c}{e^c\Gamma(c+1,\rho)}(1 - e^{-T_{\mathrm{out}}\mu_{\mathrm{c}}}) + e^{-T_{\mathrm{out}}\mu_{\mathrm{c}}} \tag{4.19}$$

where ρ, the load of an individual slot is given by (4.8).

4.4.2 Threshold timeout method

The threshold timeout method is an extension of the fixed timeout method. Instead of using a fixed timeout value, two timeout values are used. Initially, when the queue is empty a high timeout value T_0 is used. Whenever the queue occupation is greater than a certain threshold S, a lower timeout value T_1 is used.

$$T_{\mathrm{out}}^{(k)} = \begin{cases} T_0 & k \leq S \\ T_1 & \text{otherwise} \end{cases} \tag{4.20}$$

The way a connection will be assigned a timeout value varies with respect to the policy that is used.

Threshold Poisson timeout strategy

Although the Poisson policy does not have a straight-forward implementation, we choose to analyse it because of its modelling simplicity. The results obtained in this section will be a first step to analysing the more complex deterministic and deferred policies. When enforcing the Poisson policy, the service rates in the states E_1 to E_S are the same as those of a server with a fixed timeout method and with a timeout value of T_0. The service rates in the states E_{S+1} to E_c are the same as those of a server with a fixed timeout method but with a timeout value of T_1

$$\mu^{(k)} = \begin{cases} \mu_0^{(k)} & k \le S \\ \mu_1^{(k)} & \text{otherwise} \end{cases} = \begin{cases} k\mu_0 & k \le S \\ k\mu_1 & \text{otherwise} \end{cases} \tag{4.21}$$

where μ_0 and μ_1, the mean service speeds before and after the queue reaches the threshold are as described in (4.7) for timeout values of T_0 and T_1, respectively:

$$\mu_0 = \frac{\lambda_l + \lambda_m}{\lambda_l(1 - e^{-T_0\mu_c})/\mu_c + T_0\lambda_m} \tag{4.22}$$

$$\mu_1 = \frac{\lambda_l + \lambda_m}{\lambda_l(1 - e^{-T_1\mu_c})/\mu_c + T_1\lambda_m} \tag{4.23}$$

Arrival rates do not depend on the state E_k the server is in and are the same as in (4.2):

$$\lambda^{(k)} = \lambda = \lambda_l + \lambda_m \tag{4.24}$$

Figure 4.9 illustrates the model we obtained. This is a Birth-Death Markov chain with server state dependent service rates.

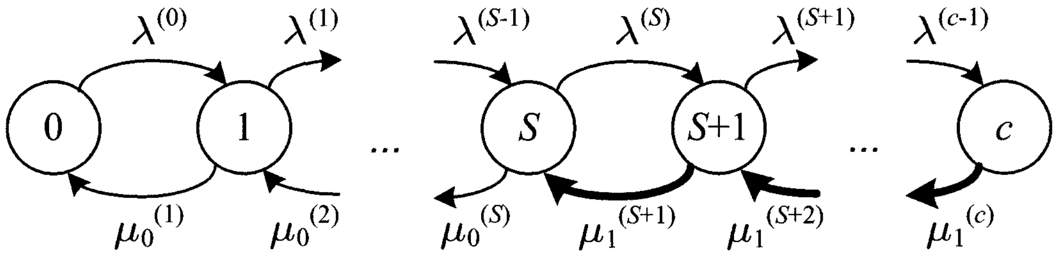

Figure 4.9 Markov chain queue representation for the threshold Poisson timeout strategy. Service transitions from states E_{S+1} to E_c are illustrated with thicker lines because in these states the service rates are higher due to the shorter timeout

We apply the same reasoning described in (4.13) and (4.14) for calculating the steady-state probabilities. As one would have expected, the average probability of a legitimate connection arriving after the steady state is achieved to be rejected Φ_r is equal the probability of the queue being full at steady-state, just like in the case of the fixed timeout method:

$$\Phi_{\mathrm{r}} = p^{(c)} \tag{4.25}$$

However, the average probability of a legitimate connection that arrives after the steady state is achieved to expire depends not only on the queue occupation distribution at steady-state but also on the transitions that the queue will make after the connection arrived. Because we are not able to evaluate these transitions, we will make the approximation that at steady-state the queue transitions that occur during the service time of connections place the queue is states with similar timeout values. Hence, according to this approximation which we refer to as the steady-state (SS) approximation, the average probability of a legitimate connection to expire depends only on the queue occupation distribution at steady-state:

$$\tilde{\Phi}_{\mathrm{e}} = \sum_{k=0}^{c-1} p^{(k)} p_{\mathrm{le}}^{(k)} \tag{4.26}$$

where $p^{(k)}$, the probability that the server is in state E_k at steady state is:

$$p^{(k)} = \frac{\prod_{i=0}^{k-1} \frac{\lambda^{(i)}}{\mu^{(i+1)}}}{\sum_{j=0}^{c} \prod_{i=0}^{j-1} \frac{\lambda^{(i)}}{\mu^{(i+1)}}} \tag{4.27}$$

and $p_{\mathrm{le}}^{(k)}$, the probability that a legitimate connection that enters the queue expires under the timeout $T_{\mathrm{out}}^{(k)}$ is evaluated similarly to (4.4):

$$p_{\mathrm{le}}^{(k)} = \begin{cases} e^{-T_0 \mu_c} & k \leq S \\ e^{-T_1 \mu_c} & \text{otherwise} \end{cases} \tag{4.28}$$

Note that we did not use the *Erlang B loss function* for expressing the steady-state probabilities as we previously did in in Section 4.4.1 in (4.14). Because the service rate described in (4.21) changes at the threshold state E_S, the recursive representation of the *Erlang B loss function* is no longer easy to solve. We did however use the recursive representation when computing the steady-state probabilities numerically.

Once again the only significant performance measure is the approximative probability that a legitimate connection arriving after the steady state fails ($\tilde{\Phi}$) which is equal to the probability that the connection is rejected (Φ_{r}) plus the approximative probability the connection expires ($\tilde{\Phi}_{\mathrm{e}}$), for connections arriving after the steady state.

$$\begin{aligned}\tilde{\Phi} &= \Phi_{\mathrm{r}} + \tilde{\Phi}_{\mathrm{e}} \\ &= p^{(0)} \frac{\lambda^c}{c!\mu_0^S \mu_1^{c-S}} + p^{(0)} e^{-T_0 \mu_c} \sum_{k=0}^{S} \frac{\lambda^k}{k!\mu_0^k} + p^{(0)} e^{-T_1 \mu_c} \sum_{k=S+1}^{c} \frac{\lambda^k}{k!\mu_0^S \mu_1^{k-S}} \end{aligned} \tag{4.29}$$

with

$$p^{(0)} = \frac{1}{\displaystyle\sum_{k=0}^{S} \frac{\lambda^k}{k!\mu_0^k} + \sum_{k=S+1}^{c} \frac{\lambda^k}{k!\mu_0^S \mu_1^{k-S}}} \tag{4.30}$$

and μ_0 and μ_1, the service speeds before and after the threshold as described in (4.22) and (4.23), respectively.

Although the performance of the Poisson policy is relatively easy to compute by simply extending the fixed timeout method, it is difficult to implement this policy in a queueing system. However, the results obtained in this section will prove to be useful when compared to those of the deferred policy, which we expect to be very similar.

Threshold deterministic timeout strategy

The deterministic policy consists in assigning a timeout right when the connection enters the queue. For example, if the server is in state E_k and a connection is accepted in the queue, the server passes in state E_{k+1}. The connection is thus assigned a timeout of $T_{\mathrm{out}}^{(k+1)}$. If the connection does not complete before the assigned timeout, it is dropped by the server. Because the deterministic policy consists in using for each connection a timeout based on the past state of the server, when the connection arrived, it breaks the memoryless property of the system. In this section we try however to make several approximations that allow us to use the same model we built earlier in order to get an insight into the performance of the deterministic policy for a threshold timeout method.

Consider the server queue having each slot numbered, from 0 to c. When a connection arrives it is placed on the slot with the lowest id number. Let us suppose there are n connections in the queue, making slots 0 to n to be occupied. When a connection being served on slot k leaves the queue, connections on slots $k+1$ to n translate on position to the left, thus leaving no empty slots in the middle of the queue. This is without loss of generality because the connection positions in the queue are merely labels, all connections being served in parallel and independently. With this in mind, the overall service rate when the server is in state E_k is the sum of the service speeds of each connection:

$$\mu^{(k)} = \mu_{|1|} + \mu_{|2|} + \ldots + \mu_{|k|} \tag{4.31}$$

where $\mu_{|i|}$, the service speed of the connection that occupies slot i in the queue depends on the past state of the server when this connection arrived $A_{|i|}$:

$$\mu_{|i|} = \begin{cases} \mu_0 & A_{|i|} < E_S \\ \mu_1 & \text{otherwise} \end{cases} \tag{4.32}$$

Evaluating (4.32) requires being able to tell what was the state of the server when each of the connections currently present in the queue arrived. Clearly this is not a memoryless model. To get past this lack of information we make the very coarse approximation that the connection on slot i arrived when the server was in state E_{i-1} and has thus been assigned the timeout value $T_{\text{out}}^{(i)}$.

$$\tilde{\mu}_{|i|} = \begin{cases} \mu_0 & i < S \\ \mu_1 & \text{otherwise} \end{cases} \tag{4.33}$$

Intuitively, the approximation would hold only if the queue acted like a *First In, Last Out* (FILO) queue, meaning that only the connection placed last in the queue would expire or complete. Hence, we refer to this approximation as the FILO queue approximation. This is obviously not the normal behaviour of a server and we will measure the error generated by this approximation in Section 4.6.3. By inserting

(4.33) into (4.31) we obtain the approximative service rate when server is in state E_k:

$$\tilde{\mu}^{(k)} = \tilde{\mu}_{|1|} + \tilde{\mu}_{|2|} + \ldots + \tilde{\mu}_{|k|} = \mu_0 \min(k, S) + \mu_1(k - \min(k, S)) = \begin{cases} k\mu_0 & k < S \\ S\mu_0 + (k - S)\mu_1 & \text{otherwise} \end{cases} \tag{4.34}$$

We can now calculate the approximative steady-state queue occupation distribution similarly to 4.27:

$$\tilde{p}^{(k)} = \frac{\displaystyle\prod_{i=0}^{k-1} \frac{\lambda^{(i)}}{\tilde{\mu}^{(i+1)}}}{\displaystyle\sum_{j=0}^{c} \prod_{i=0}^{j-1} \frac{\lambda^{(i)}}{\tilde{\mu}^{(i+1)}}} \tag{4.35}$$

Just as we used the steady-state queue occupation probabilities to evaluate the performance of the Poisson policy, so can we use the steady-state queue occupation approximative probabilities to evaluate the performance of the deterministic policy. The average probability for a legitimate connection arrived after the steady state is achieved to be rejected is the steady-state probability that the queue being full:

$$\tilde{\Phi}_{\mathrm{r}} = \tilde{p}^{(c)} \tag{4.36}$$

The average probability for a legitimate connection arrived after the steady state is achieved to expire is the steady-state probability of the queue being in a non blocking state ($E_k; k < c$) multiplied by the probability of a legitimate connection arrived in state E_k to expire:

$$\tilde{\Phi}_{\mathrm{e}} = \sum_{k=0}^{c-1} \tilde{p}^{(k)} p_{\mathrm{le}}^{(k)} \tag{4.37}$$

A legitimate connection fails either if it is rejected or if it expires:

$$\begin{aligned}\tilde{\Phi} &= \tilde{\Phi}_{\mathrm{r}} + \tilde{\Phi}_{\mathrm{e}} \\ &= p^{(0)} \frac{\lambda^c}{S!\mu_0^S} \prod_{i=S+1}^{c} \frac{1}{S\mu_0 + (i-S)\mu_1} + p^{(0)} e^{-T_0\mu_c} \sum_{k=0}^{S} \frac{\lambda^k}{k!\mu_0^k} + \\ &+ p^{(0)} e^{-T_1\mu_c} \sum_{k=S+1}^{c} \left(\frac{\lambda^k}{S!\mu_0^S} \prod_{i=S+1}^{k} \frac{1}{S\mu_0 + (i-S)\mu_1} \right) \end{aligned} \tag{4.38}$$

where

$$p^{(0)} = \frac{1}{\sum_{k=0}^{S} \frac{\lambda^k}{k!\mu_0^k} + \sum_{k=S+1}^{c} \left(\frac{\lambda^k}{S!\mu_0^S} \prod_{i=S+1}^{k} \frac{1}{S\mu_0 + (i-S)\mu_1} \right)} \tag{4.39}$$

Threshold deferred timeout strategy

When enforcing the deferred policy, the timeout is not assigned to a connection when it arrives at the server. Instead, the arrival time is recorded and if at any point the arrival time of a connection plus the current timeout is greater than the current time, the connection is dropped from the queue. If more than one connection satisfy this condition, only the oldest one will be dropped. Afterwards, the current timeout is re-evaluated based on the new server state and the process continues.

As opposed to the deterministic policy, the deferred policy does not have a strong dependence of the past. The connections expire regardless of the queue occupation when they arrived. When the server is in state $E_k, k \leq S$ the connections are served with mean service speed μ_0. Otherwise, the connections are served with mean service speed μ_1. However, there is an exception to this rule. When the server transitions from state E_S to state E_{S+1}, the timeout is suddenly lowered from T_0 to T_1 which may cause several connections to have an age greater than what the server tolerates under the new, shorter timeout T_1. We call this the abrupt-tolerance-drop effect. Thus, if a connection is *older* than the short timeout T_1 but is *younger* that the long timeout T_0 then this connection gets dropped instantly when the server transitions from E_S to E_{S+1}. We required the connection to be younger that the long timeout T_0 because otherwise the connection would have had to expire before the transition. Hence, the service rates in the states E_1 to E_S are the same as those of a server with a fixed timeout method and with a timeout value of T_0. The service rate at threshold (μ_S), when the server is in state E_{S+1}, is something that we need to investigate. The service rates in the states E_{S+2} to E_c are the same as those of a server with a fixed timeout method but with a timeout value of T_1. The model is illustrated in Figure 4.10.

$$\mu^{(k)} = \begin{cases} k\mu_0 & k \leq S \\ (S+1)\mu_S & k = S+1 \\ k\mu_1 & \text{otherwise} \end{cases} \tag{4.40}$$

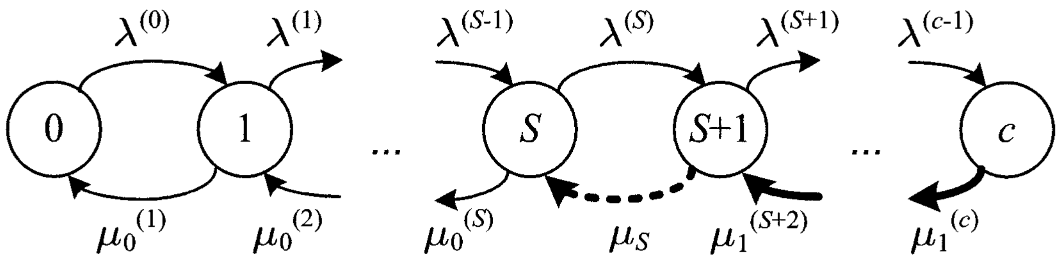

Figure 4.10 Birth-Death chain queue representation for the threshold deferred timeout strategy. Service transitions from states E_{S+2} to E_c are illustrated with thicker lines because in these cases the service rates are higher due to the shorter timeout. The service transition from the state E_{S+1} is illustrated with a dashed line because it is not memoryless

We now look at the connection service speed μ_S which applies when the server is in state E_{S+1}.

Legitimate connections. If previous to being in state E_{S+1} the server was in state E_S then the abrupt-tolerance-drop effect that we discussed earlier occurs. Therefore, the probability distribution function (PDF) of the legitimate connection service time has the form of a Dirac delta in 0^+ followed by an exponential distributions for t smaller than T_1 followed and another Dirac delta in T_1^+. The first Dirac delta has weight p_{li} which represents the probability that a connection is older than T_1 but younger than T_0, making the connection expire after the transition. The expiration arrives instantly *after* the transition, which causes the Dirac delta to be placed on the positive side of the axis, infinitesimally close to 0. The weight of the second Dirac delta (p_{le}) is so that the CDF of the service time is 1 after T_1 and represents the probability that a legitimate connection would not expire instantly but would however expire under timeout T_1. Once again, the Dirac delta is placed at value infinitesimally greater than T_1 signifying that the connection will expire *after* the timeout. Figure 4.11 illustrates the service time PDF and CDF.

$$G_1^{(S\rightarrow S+1)}(t) \; = \; \delta(t-0^+)p_{\text{li}} + \delta(t-T_1^+)p_{\text{le}} + \begin{cases} \mu_c e^{-t\mu_c} & t < T_1 \\ 0 & \text{otherwise} \end{cases} \tag{4.41}$$

Because the service rate follows a Poisson process distribution and is thus memoryless, the conditional probability that a connection is older than the timeout T_1 knowing that the connection is still in the queue, i.e. is younger than T_0, is evaluated as:

$$p_{\text{li}} = \frac{T_0 - T_1}{T_0} \tag{4.42}$$

and p_{le} satisfies the condition that the service time CDF is 1 in T_1:

$$p_{\text{le}} = 1 - \int_0^{T_1} G_1^{(S\rightarrow S+1)}(t)dt = e^{-T_1\mu_c} - \frac{T_0 - T_1}{T_0} \tag{4.43}$$

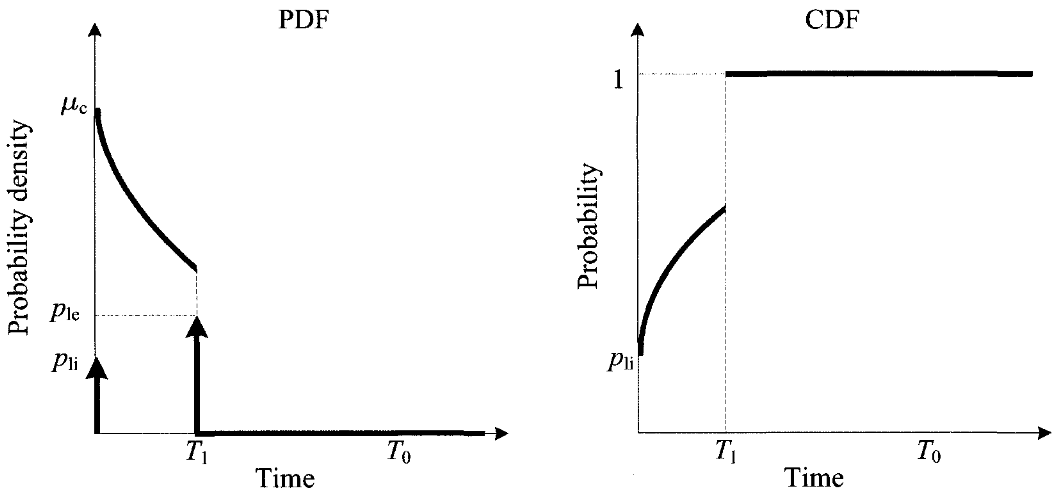

Figure 4.11 Legitimate service time PDF and CDF after transition $E_S \rightarrow E_{S+1}$ for the threshold deferred timeout strategy

The mean service time for legitimate connections is :

$$t_1^{(S\to S+1)} \triangleq \int_0^{\infty} tG_1^{(S\to S+1)}(t)dt = \frac{1-e^{-T_1\mu_c}}{\mu_c} + \frac{T_1(T_1-T_0)}{T_0} \tag{4.44}$$

If previously to being in state E_{S+1} the server was in state E_{S+2} then the abrupt-tolerance-drop effect does not occur. The service time PDF and CDF are illustrated in Figure 4.12. Similar to the fixed timeout method, the mean service time for legitimate connections is:

$$t_1^{(S+2\to S+1)} = t_1^{(S+2)} = \frac{1-e^{-T_1\mu_c}}{\mu_c} \tag{4.45}$$

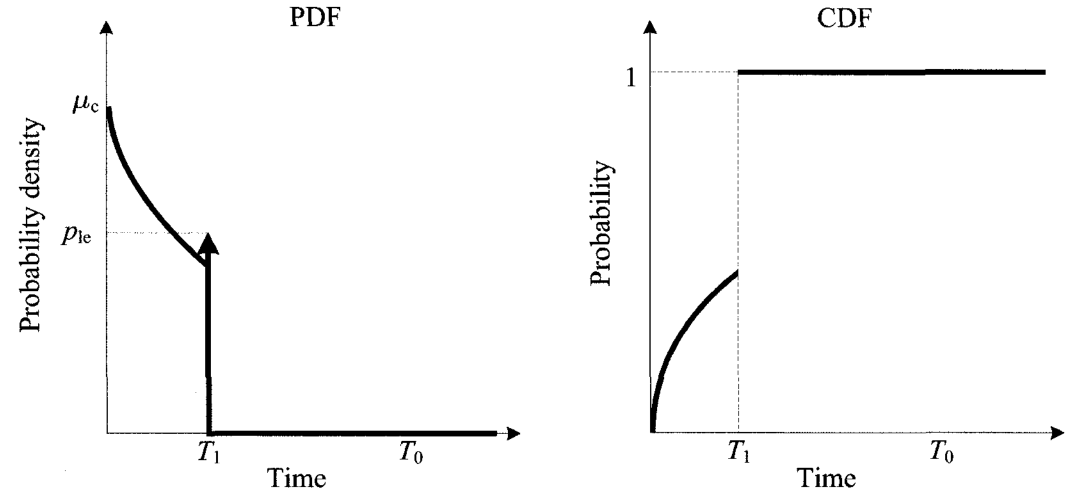

Figure 4.12 Legitimate service time PDF and CDF after transition $E_{S+2} \to E_{S+1}$ for the threshold deferred timeout strategy

Finally, the legitimate connection service speed in state E_{S+1} is equal to the average between the legitimate connection service speed when the previous state was E_S and that when the previous state was E_{S+2} weighted by the probabilities of the server previously being in states E_S and E_{S+2}, $p^{(S\to S+1)}$ and $p^{(S+2\to S+1)}$, respectively.

$$\mu_{1S} = \frac{p^{(S\to S+1)}/t_1^{(S\to S+1)} + p^{(S+2\to S+1)}/t_1^{(S+2\to S+1)}}{p^{(S\to S+1)} + p^{(S+2\to S+1)}} \tag{4.46}$$

The difficulty of solving (4.46) comes from evaluating the probabilities $p^{(S\to S+1)}$

and $p^{(S+2 \to S+1)}$. We leave this problem aside for now as we will later solve it by introducing an approximation.

Malicious connections. Let us apply the same reasoning we used for the legitimate connections for evaluating the malicious connection service speed in state E_{S+1}. Figure 4.13 illustrates the service time PDF and CDF when the previous was E_S, the PDF being defined by:

$$G_{\mathrm{m}}^{(S \to S+1)}(t) = \delta(t - 0^{+})p_{\mathrm{mi}} + \delta(t - T_1^{+})p_{\mathrm{me}} \tag{4.47}$$

with

$$p_{\mathrm{mi}} = \frac{T_0 - T_1}{T_0} \tag{4.48}$$

and with p_{me} so that the CDF of the service time is 1 in t_f^{+}:

$$p_{\mathrm{me}} = 1 - \int_0^{T_1} G_{\mathrm{m}}^{(S \to S+1)}(t)dt = \frac{T_1}{T_0} \tag{4.49}$$

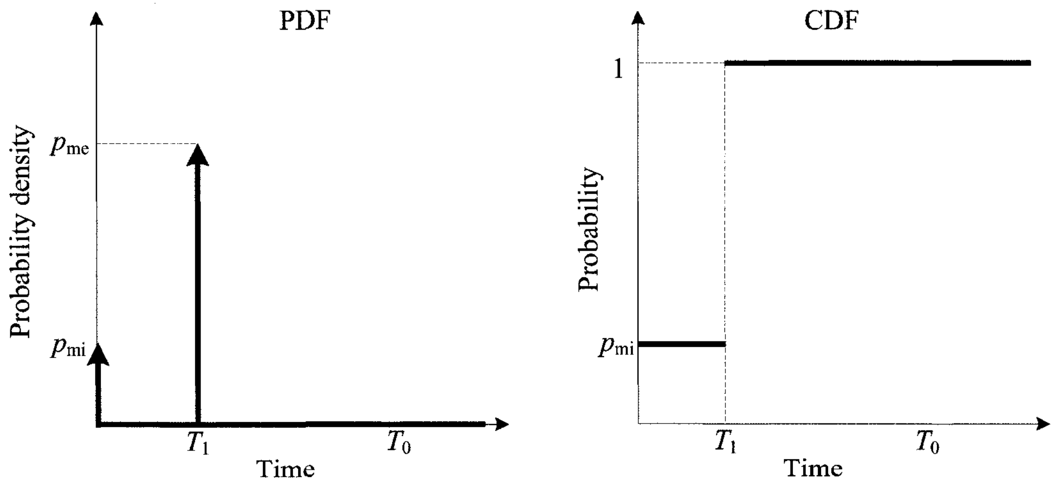

Figure 4.13 Malicious service time PDF and CDF after transition $E_S \to E_{S+1}$ for the threshold deferred timeout strategy

The mean service time for malicious connections becomes:

$$t_{\mathrm{m}}^{(S \to S+1)} \triangleq \int_0^\infty t G_{\mathrm{m}}^{(S \to S+1)}(t) dt = \frac{(T_1)^2}{T_0} \tag{4.50}$$

Just as in the case of the legitimate connections, if previously to being in state E_{S+1} the server was in state E_{S+2} then the abrupt-tolerance-drop effect does not occur. We illustrate the service time PDF and CDF in Figure 4.14. The malicious service time in this case is simply T_1. Overall, the malicious connection service speed in state E_{S+1} is :

$$\mu_{\mathrm{m}S} = \frac{p^{(S \to S+1)}/t_{\mathrm{m}}^{(S \to S+1)} + p^{(S+2 \to S+1)}/T_1}{p^{(S \to S+1)} + p^{(S+2 \to S+1)}} \tag{4.51}$$

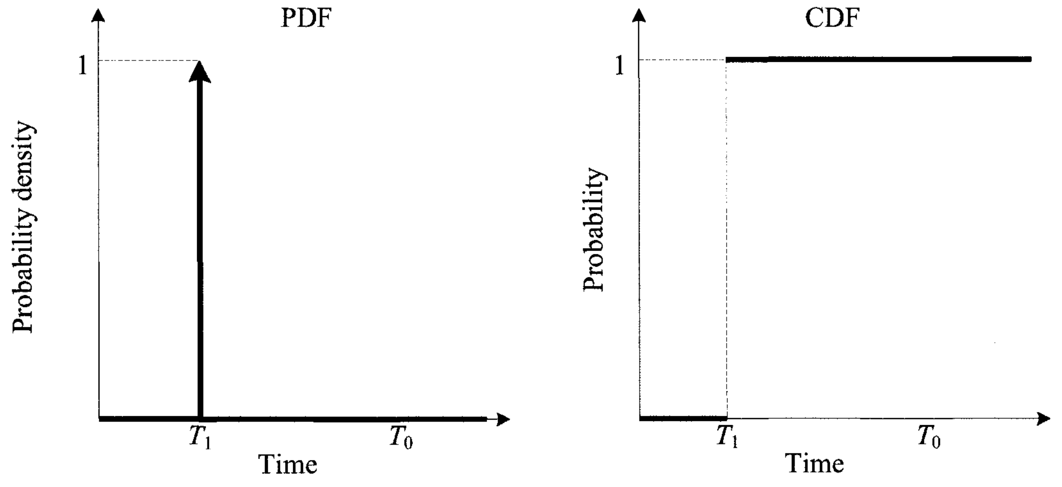

Figure 4.14 Malicious service time PDF and CDF after transition $E_{S+2} \to E_{S+1}$ for the threshold deferred timeout strategy

We can now combine the legitimate and malicious mean service times in state E_{S+1} in order to express the overall mean service time in state E_{S+1}, as we did in (4.6):

$$\tilde{t}_S = \frac{\lambda_1/\mu_{1S} + \lambda_{\mathrm{m}}/\mu_{\mathrm{m}S}}{\lambda_1 + \lambda_{\mathrm{m}}} \tag{4.52}$$

The service rate depends on the previous server state. In order to evaluate (4.52) we require the probabilities that the server was previously is in states E_S and E_{S+2}, respectively. We conclude that when in state E_{S+1}, the server is not memoryless. The recursion between the transient state of the chain and the service rate in state E_{S+1} is not trivial to express due to the complex nature of our model, let alone to solve. We are left with no choice but to make the following approximation: when the server transitions between states E_{S+1} and E_S, the previous state of the server was E_S. The approximation seems natural in the sense that if a transition $E_S \rightarrow E_{S+1}$ occurs, it is probable that a transition $E_{S+1} \rightarrow E_S$ occurs right after due to a connection expiring because of the abrupt-tolerance-drop effect. However, if after a transition $E_S \rightarrow E_{S+1}$ no connections expire and a new connection arrives placing the server in state E_{S+2}, the server has escaped the attraction point at E_{S+1} and the queue will probably fill up some more, making transitions $E_{S+2} \rightarrow E_{S+1}$ less probable. We will refer to this approximation in the future as the Attraction Point (AP) approximation. With this in mind, the estimated legitimate and malicious service speeds at threshold become:

$$\tilde{\mu}_{lS} = \frac{1}{t_l^{(S \rightarrow S+1)}} = \frac{T_0 \mu_c}{T_0(1 - e^{-T_0 \mu_c}) + T_1 \mu_c (T_1 - T_0)} \tag{4.53}$$

$$\tilde{\mu}_{mS} = \frac{1}{t_m^{(S \rightarrow S+1)}} = \frac{T_0}{(T_1)^2} \tag{4.54}$$

Having approximative values for the service speed at threshold μ_S we can now calculate the approximative steady-state queue occupation distribution similarly to the Poisson policy, as described in (4.27):

$$\tilde{p}^{(k)} = \frac{\prod_{i=0}^{k-1} \frac{\lambda^{(i)}}{\tilde{\mu}^{(i+1)}}}{\sum_{j=0}^{c} \prod_{i=0}^{j-1} \frac{\lambda^{(i)}}{\tilde{\mu}^{(i+1)}}} = \begin{cases} \dfrac{\tilde{p}^{(0)} \lambda^k}{k! \mu_0^k} & k \leq S \\[2ex] \dfrac{\tilde{p}^{(0)} \lambda^{S+1}}{(S+1)! \mu_0^S \mu_S} & k = S+1 \\[2ex] \dfrac{\tilde{p}^{(0)} \lambda^k}{k! \mu_0^S \mu_S \mu_1^{k-S-1}} & \text{otherwise} \end{cases} \tag{4.55}$$

where

$$\tilde{p}^{(0)} = \frac{1}{\sum_{k=0}^{S} \frac{\lambda^k}{k!\mu_0^k} + \frac{\lambda^{S+1}}{(S+1)!\mu_0^S \mu_S} + \sum_{k=S+2}^{c} \frac{\lambda^k}{k!\mu_0^S \mu_S \mu_1^{k-S-1}}} \tag{4.56}$$

and

$$\mu_S = \frac{T_0 \mu_c \lambda}{\lambda_l T_0 (1 - e^{-T_0 \mu_c}) + \lambda_l T_0 \mu_c (T_1 - T_0) + \lambda_m \mu_c T_1^2} \tag{4.57}$$

Although by having the approximative steady-state queue occupation probabilities we could apply the same SS approximation that we used for the Poisson policy in (4.26), we choose not to follow this path. Trying to evaluate the performance of the deferred policy in this way means using both the AP and the SS approximation, which would only amplify the errors. However, (4.55) is useful because it allows us measure the resemblance of the steady-state queue occupation probabilities obtained when using the deferred policy with those obtained when using the Poisson policy, which we suspect to be very similar.

4.4.3 Linear timeout method

The linear timeout method is a straight-forward extension of the threshold timeout method, where instead of lowering the timeout when the queue occupation is greater than a certain threshold, the timeout is lowered with a small increment every time a connection enters the queue. When a connection departs the queue, the timeout is increased with the same small increment. When the queue is empty, an initial timeout value T_0 is used. When the queue is full, the timeout becomes T_1. The dynamic timeout in (4.20) becomes:

$$T_{\text{out}}^{(k)} = T_0 + \frac{k(T_1 - T_0)}{c} \tag{4.58}$$

Let us now analyse the policies for assigning the dynamic timeout to connections.

Linear Poisson timeout strategy

The Poisson policy for assigning timeouts is very similar to that of the threshold method, except that for the linear method the timeout in different for every state. The same reasoning used in Section 4.4.2 applies for evaluating the performance of the linear Poisson strategy. However, because the timeout is different in every state, we cannot group the terms of the performance equation together as we did in (4.29):

$$\begin{aligned}\tilde{\Phi} &= \Phi_{\mathrm{r}} + \tilde{\Phi}_{\mathrm{e}} \\ &= p^{(0)} \prod_{i=1}^{c} \left(\frac{\lambda_{\mathrm{l}}(1 - e^{-T_{\mathrm{out}}^{(k)} \mu_{\mathrm{c}}})}{\mu_{\mathrm{c}}} + T_{\mathrm{out}}^{(k)} \lambda_{\mathrm{m}} \right) + \\ &\quad + p^{(0)} \sum_{k=0}^{c-1} e^{-T_{\mathrm{out}}^{(k)} \mu_{\mathrm{c}}} \prod_{i=1}^{k} \left(\frac{\lambda_{\mathrm{l}}(1 - e^{-T_{\mathrm{out}}^{(k)} \mu_{\mathrm{c}}})}{\mu_{\mathrm{c}}} + T_{\mathrm{out}}^{(k)} \lambda_{\mathrm{m}} \right) \end{aligned} \tag{4.59}$$

with

$$p^{(0)} = \frac{1}{\sum_{k=0}^{c-1} \prod_{i=1}^{k} \left(\frac{\lambda_{\mathrm{l}}(1 - e^{-T_{\mathrm{out}}^{(k)} \mu_{\mathrm{c}}})}{\mu_{\mathrm{c}}} + T_{\mathrm{out}}^{(k)} \lambda_{\mathrm{m}} \right)} \tag{4.60}$$

Linear deterministic timeout strategy

When analysing the threshold deterministic strategy we established that the service rate is history dependent. The same is the case with the linear deterministic strategy. The overall service rate when the server is in state E_k is:

$$\mu^{(k)} = \mu_{|1|} + \mu_{|2|} + \dots + \mu_{|k|} \tag{4.61}$$

The service speed of the connection that occupies slot i in the queue ($\mu_{|i|}$) depends on the server state when the connection arrived at the queue ($A_{|i|}$). Without loss of generality, if $A_{|i|} = E_{j-1}$, the service speed of the connection on slot i is described by:

$$\mu_{|i|} = \frac{\lambda_{\mathrm{l}} + \lambda_{\mathrm{m}}}{\frac{\lambda_{\mathrm{l}}(1 - e^{-T_0 - (T_1 - T_0)\frac{j}{c}\mu_{\mathrm{c}}})}{\mu_{\mathrm{c}}} + \lambda_{\mathrm{m}} \left(T_0 + (T_1 - T_0)\frac{j}{c} \right)} \tag{4.62}$$

The approximation that we made in Section 4.4.2 , supposing that the queue acts like a FILO queue ($A_{|i|} = E_{i-1}$), meaning that the connection on slot i arrived when the server was in state E_{i-1} makes more sense in the case of the linear method. Because the timeout gets gradually decreased when the queue is filled, the last connection that entered the queue is presented with the lowest timeout and it is probable that it leaves the queue first, by expiring. According to this approximation, the service rate of the connection on slot i becomes:

$$\tilde{\mu}_{|i|} = \frac{\lambda_\mathrm{l} + \lambda_\mathrm{m}}{\dfrac{\lambda_\mathrm{l}(1 - e^{-T_0 - (T_1 - T_0)\frac{i}{c}\mu_\mathrm{c}})}{\mu_\mathrm{c}} + \lambda_\mathrm{m}\left(T_0 + (T_1 - T_0)\dfrac{i}{c}\right)} \tag{4.63}$$

We obtain the approximative service rate when server is in state E_k:

$$\tilde{\mu}^{(k)} = \sum_{i=1}^{k} \tilde{\mu}_{|i|} \tag{4.64}$$

The approximative steady-state queue occupation distribution is evaluated similarly to 4.27:

$$\tilde{p}^{(k)} = \frac{\displaystyle\prod_{i=0}^{k-1} \frac{\lambda^{(i)}}{\tilde{\mu}^{(i+1)}}}{\displaystyle\sum_{j=0}^{c} \prod_{i=0}^{j-1} \frac{\lambda^{(i)}}{\tilde{\mu}^{(i+1)}}} \tag{4.65}$$

We use the estimation in (4.65) to calculate the performance of the server by evaluating the steady-state reject and expire probabilities:

$$\tilde{\Phi}_\mathrm{r} = \tilde{p}^{(k)} \tag{4.66}$$

$$\tilde{\Phi}_\mathrm{e} = \sum_{k=0}^{c-1} \tilde{p}^{(k)} p_\mathrm{le}^{(k)} \tag{4.67}$$

The average fail probability for legitimate connections arriving at the queue after the steady state is achieved is:

$$\tilde{\Phi} \;=\; \tilde{\Phi}_{\mathrm{r}} + \tilde{\Phi}_{\mathrm{e}} = \tilde{p}^{(0)} \frac{\lambda^{c}}{\prod_{i=1}^{c}\sum_{j=1}^{i}\tilde{\mu}_{|j|}} + \tilde{p}^{(0)} \sum_{k=0}^{c} \frac{e^{-T_{\mathrm{out}}^{(k)}\mu_c}\lambda^{k}}{\prod_{i=1}^{k}\sum_{j=1}^{i}\tilde{\mu}_{|j|}} \tag{4.68}$$

with

$$\tilde{p}^{(0)} = 1\Big/\sum_{k=0}^{c}\frac{\lambda^{k}}{\prod_{i=1}^{k}\sum_{j=1}^{i}\tilde{\mu}_{|j|}} \tag{4.69}$$

and $\tilde{\mu}_{|j|}$, the service speed of the connection on slot j as described in (4.63).

Linear deferred timeout strategy

When enforcing the deferred policy, the abrupt-tolerance-drop effect occurs in all states E_1 to E_c, the service rates in these states depending on the previous state of the server. In state E_c however, the only possible previous state is E_{c-1}. The model is illustrated in Figure 4.15.

$$\mu^{(k)} = \begin{cases} \dfrac{p^{(k-1\to k)}\mu^{(k-1\to k)} + p^{(k+1\to k)}\mu^{(k+1\to k)}}{p^{(k-1\to k)} + p^{(k+1\to k)}} & k < c \\[2ex] \mu^{(k-1\to k)} & k = c \end{cases} \tag{4.70}$$

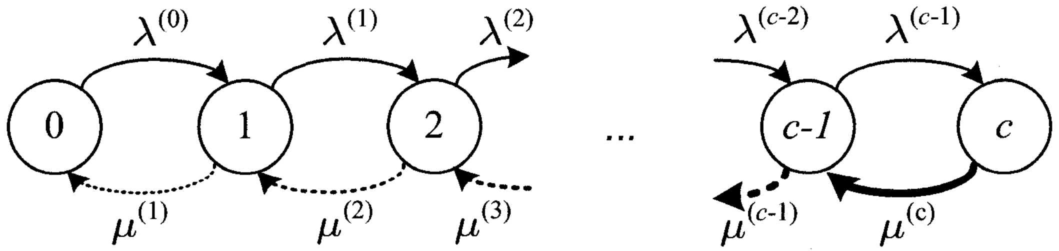

Figure 4.15 Birth-Death chain queue representation for the linear deferred timeout strategy. The dashed lines represent transition whose rates are past dependent

We extend the use of the AP approximation that we made for the threshold deferred timeout strategy by considering that whenever a transition $E_{k+1} \rightarrow E_k$ occurs, meaning that a connection left the queue, if the connection expired than this happened because of the abrupt-tolerance-drop effect due to the server previously being in state E_k. We employ the same reasoning for coupling the legitimate and malicious service rates as we did for the fixed timeout in Section 4.4.1 and we obtain the approximative individual service rate in state E_{k+1}:

$$\tilde{\mu}^{(k+1)} = \mu^{(k \rightarrow k+1)} \triangleq \frac{1}{\tilde{t}^{(k \rightarrow k+1)}} = \frac{\lambda_{\mathrm{l}} + \lambda_{\mathrm{m}}}{\lambda_{\mathrm{l}}/\mu_{\mathrm{l}}^{(k \rightarrow k+1)} + \lambda_{\mathrm{m}}/\mu_{\mathrm{m}}^{(k \rightarrow k+1)}} \tag{4.71}$$

where

$$\mu_{\mathrm{l}}^{(k \rightarrow k+1)} = \frac{T_{\mathrm{out}}^{(k)} \mu_{\mathrm{c}}}{T_{\mathrm{out}}^{(k)} \left(1 - p_{\mathrm{le}}^{(k)}\right) + T_{\mathrm{out}}^{(k+1)} \mu_{\mathrm{c}} \left(T_{\mathrm{out}}^{(k+1)} - T_{\mathrm{out}}^{(k)}\right)} \tag{4.72}$$

and

$$\mu_{\mathrm{m}}^{(k \rightarrow k+1)} = \frac{T_{\mathrm{out}}^{(k)}}{\left(T_{\mathrm{out}}^{(k+1)}\right)^2} \tag{4.73}$$

and

$$p_{\mathrm{le}}^{(k)} = e^{-T_{\mathrm{out}}^{(k)} \mu_{\mathrm{c}}} \tag{4.74}$$

We use the approximative values for the service rates to calculate the approximative steady-state queue occupation distribution similarly to the Poisson policy, as described in (4.27):

$$\tilde{p}^{(k)} = \frac{\prod_{i=0}^{k-1} \frac{\lambda^{(i)}}{\tilde{\mu}^{(i+1)}}}{\sum_{j=0}^{c} \prod_{i=0}^{j-1} \frac{\lambda^{(i)}}{\tilde{\mu}^{(i+1)}}} \tag{4.75}$$

where

$$\begin{aligned}\tilde{\mu}^{(k)} &= kT_{\text{out}}^{(k-1)}\mu_{\text{c}}\lambda \Big/ \Big(\lambda_{\text{l}} T_{\text{out}}^{(k-1)} \left(1 - e^{-T_{\text{out}}^{(k-1)}\mu_{\text{c}}}\right) + \\ &\quad + \lambda_{\text{l}} T_{\text{out}}^{(k-1)}\mu_{\text{c}} \left(T_{\text{out}}^{(k)} - T_{\text{out}}^{(k-1)}\right) + \lambda_{\text{m}}\mu_{\text{c}} \left(T_{\text{out}}^{(k)}\right)^2 \Big) \end{aligned} \tag{4.76}$$

Once again, we choose not to use the SS approximation to estimate the performance of the deferred policy but plan to make use of (4.75) to show the resemblance of the linear Poisson and linear deferred strategies.

4.4.4 Convergence study

In order for the steady-state based results to be valid, the convergence of the system toward the steady-state should not be very long when compared to the timeout values. We analyse theoretically the convergence speed of the fixed timeout method using the Modified-Offered-Load (MOL) approximation introduced by Jagerman (1975).

The MOL approximation was initially designed in order to evaluate the transient behaviour of a $M_t/G/c/c$ queue, where the arrivals follow a Poisson process, but with mean rate varying in time. The approximation consists in coupling results from an $M_t/G/\infty$ queue with an $M_t/G/c/c$ one. The $M_t/G/\infty$ queue differs from the $M_t/G/c/c$ queue in the fact that it has an infinite number of slots, that are served independently by the server in a parallel fashion. The mean number of occupied slots in an $M_t/G/\infty$ queue at time t is:

$$m_\infty(t) = \int_{-\infty}^{t} [1 - C(t-u)]\lambda(u)du \tag{4.77}$$

where $C(t)$ is the service time CDF and $\lambda(t)$ is the arrival rate at time t.

We adapt the MOL approximation for our purposes by considering that there is no activity before time $t = 0$. Furthermore, we need to consider both the legitimate and malicious arrival and service processes. Hence, we consider that after time $t = 0$, both the legitimate and the malicious traffic start, with the constant rates λ_{l} and λ_{m}, respectively.

$$\lambda_{\text{l}}(t) \triangleq \begin{cases} \lambda_{\text{l}} & t > 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.78}$$

$$\lambda_{\mathrm{m}}(t) \triangleq \begin{cases} \lambda_{\mathrm{m}} & t > 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.79}$$

When considering the fixed timeout method, the legitimate and malicious service time CDF are described by:

$$C_{\mathrm{l}}(t) \triangleq \int_0^t G_{\mathrm{l}}(u)du = \begin{cases} 0 & t < 0 \\ 1 - e^{-t\mu_{\mathrm{c}}} & 0 \leq t < T_{\mathrm{out}} \\ 1 & T_{\mathrm{out}} \leq t \end{cases} \tag{4.80}$$

$$C_{\mathrm{m}}(t) \triangleq \int_0^t G_{\mathrm{m}}(u)du = \begin{cases} 0 & t < T_{\mathrm{out}} \\ 1 & T_{\mathrm{out}} \leq t \end{cases} \tag{4.81}$$

The mean number of occupied slots by legitimate connections in an infinite queue $m_{\mathrm{l}\infty}(t)$ and the mean number of occupied slots by malicious connections in an infinite queue $m_{\mathrm{m}\infty}(t)$ for time t greater than 0 are defined as:

$$m_{\mathrm{l}\infty}(t) = \int_{-\infty}^t [1 - C_{\mathrm{l}}(t-u)]\lambda_{\mathrm{l}}(u)du = \begin{cases} \lambda_{\mathrm{l}}(1 - e^{-t\mu_{\mathrm{c}}})/\mu_{\mathrm{c}} & t < T_{\mathrm{out}} \\ \lambda_{\mathrm{l}}(1 - e^{-T_{\mathrm{out}}\mu_{\mathrm{c}}})/\mu_{\mathrm{c}} & T_{\mathrm{out}} \leq t \end{cases} \tag{4.82}$$

$$m_{\mathrm{m}\infty}(t) = \int_{-\infty}^t [1 - C_{\mathrm{m}}(t-u)]\lambda_{\mathrm{m}}(u)du = \begin{cases} t\lambda_{\mathrm{m}}/\mu_{\mathrm{c}} & t < T_{\mathrm{out}} \\ T_{\mathrm{out}}\lambda_{\mathrm{m}}/\mu_{\mathrm{c}} & T_{\mathrm{out}} \leq t \end{cases} \tag{4.83}$$

The MOL approximation consists in substituting the load λ/μ by the mean number of occupied slots $m_\infty(t)$ in the infinite queue (Massey et Whitt, 1994). Because we are dealing with two types of arrival and service rates, legitimate and malicious, we substitute the time dependent service rates $\mu_{\mathrm{l}}(t)$ and $\mu_{\mathrm{m}}(t)$ with the normalized rates $\lambda_{\mathrm{l}}/m_{\mathrm{l}\infty}(t)$ and $\lambda_{\mathrm{m}}/m_{\mathrm{m}\infty}(t)$, respectively. Then, we apply the same reasoning for coupling the legitimate and malicious service rates as we did in (4.7) and obtain the time dependant connection service speed according to the MOL approximation:

$$\mu_{\mathrm{MOL}}(t) = \begin{cases} \dfrac{\lambda_{\mathrm{l}} + \lambda_{\mathrm{m}}}{\lambda_{\mathrm{l}}(1 - e^{-t\mu_{\mathrm{c}}})/\mu_{\mathrm{c}} + t\lambda_{\mathrm{m}}} & t < T_{\mathrm{out}} \\ \dfrac{\lambda_{\mathrm{l}} + \lambda_{\mathrm{m}}}{\lambda_{\mathrm{l}}(1 - e^{-T_{\mathrm{out}}\mu_{\mathrm{c}}})/\mu_{\mathrm{c}} + T_{\mathrm{out}}\lambda_{\mathrm{m}}} & T_{\mathrm{out}} \leq t \end{cases} \tag{4.84}$$

In state E_k, the overall service rate is thus:

$$\mu^{(k)}_{\mathrm{MOL}}(t) = k\mu_{\mathrm{MOL}}(t) \tag{4.85}$$

When applying the approximation to the fixed timeout method, (4.13) and (4.14) become time dependent:

$$B(c,t)^{-1} = \frac{\mu^{(c)}_{\mathrm{MOL}}(t)}{\lambda^{(c-1)}} B(c-1,t)^{-1} + 1 \quad ; \quad B(0,t) = 1 \tag{4.86}$$

$$p^{(k)}(t) = \frac{B(c,t)}{\prod_{i=k}^{c-1} \dfrac{\lambda^{(i)}}{\mu^{(i+1)}_{\mathrm{MOL}}(t)}} \tag{4.87}$$

We analyse (4.85), (4.86) and (4.87) and notice that the only time-dependent parameter is the service speed $\mu_{\mathrm{MOL}}(t)$. However, according to (4.84) the service speed only varies before the first timeout elapses, i.e. $t < T_{\mathrm{out}}$. Hence, according to the MOL approximation, when using the fixed timeout method, the queue achieves steady state after an interval of time equal to the timeout T_{out}. Although we do not show the mathematical proof, the same conclusion can be drawn for the dynamic timeout strategies: the queue achieves steady state at most after an interval of time equal to the longest timeout, T_0. In order to be effective, attacks usually last much longer than the timeout. Hence, we conclude that the steady-state performance measures we established previously are relevant and can be used as accurate approximations of performance. We will confirm these theoretical results by stochastic simulations in Section 4.5.3.

In Section 4.4 we studied several timeout strategies. Due to the non-memoryless property of most of these strategies, we introduced various approximations that allowed us to estimate the steady-state distribution of the queue and the performance of the server when employing each of these strategies. We present a summary of the theoretical results obtained for the various timeout adjustment methods and assignment policies studied, as well as the approximations used in Table 4.3.

4.5 Model validation

In order to validate the correctness of the mathematical models established earlier, and to evaluate the impact of the approximations used to cope with the non-memoryless behaviour of the different strategies, we ran stochastic simulations for various traffic rates and queue sizes.

4.5.1 Simulation setup

For simulation purposes, we implemented the dynamic timeout strategies in a home-made stochastic simulator. Before running each simulation, the legitimate arrivals times and times to complete are generated according to Poisson processes. The malicious arrivals times are generated according to either a Poisson process or a burst attack model, which we describe in more detail in Section 4.6.4. These occurrences are saved to a file so that simulation of the various timeout strategies in identical conditions is possible. At time $t = 0$ of each simulation, the server queue is empty. This is when both the legitimate and malicious traffic start. The simulation is run during a period of time ten times longer than the timeout, in the case of the fixed timeout method, and ten times longer than the empty-queue T_0 timeout, in the case of the dynamic timeout methods. After the simulation is complete, the connections that

Table 4.3 Summary of the theoretical results on timeout adjustment methods and assignment policies with their corresponding mathematical equation indexes. MOL is the Modified Offered Load approximation, FILO is the First-In-Last-Out queue approximation, AP is the Attraction Point approximation and SS is the Steady-State approximation

Method type	Fixed	Dynamic					
Method	Fixed	Threshold			Linear		
Policy	n/a	Poisson	Deterministic	Deferred	Poisson	Deterministic	Deferred
Transient state	MOL (4.87)						
Steady-state	exact (4.16)	exact (4.27)	FILO (4.35)	AP (4.55)	exact (4.27)	FILO (4.65)	AP (4.75)
Performance	exact (4.19)	SS (4.29)	FILO (4.38)		SS (4.59)	FILO (4.68)	

are still in the queue and have not yet expired or completed are discarded. Finally, the transient state behaviour as well as the overall performance is gathered from the simulation logs.

4.5.2 Steady-state queue occupation

In the theoretical model, the legitimate connection success rate is calculated based on the steady-state server occupation probabilities. However, in Section 4.4 we made several approximations in order to adapt the Markov chain model to the deterministic and deferred timeout assignment policies. Before comparing the performance of the strategies, we need to assess the errors generated by these approximations. Furthermore, we have no simulation or experimental equivalent of the Poisson assignment policy and our mathematical model does not allow us to compute the performance of the deferred assignment policy. We hope that by comparing the steady-state queue occupation, not only will we validate the approximations made earlier but we will also establish similarities between some of the assignment policies.

In Figure 4.16, we illustrate the theoretical and simulation queue occupation probability distributions at the steady state; for the simulations, this is computed as the average of queue occupations over the whole simulation. We draw several conclusions from these results. First, the fixed method queue occupation simulation results match very closely the theoretical results. Second, the theoretical threshold Poisson, theoretical threshold deferred and simulation threshold deferred distributions show a similar spike around the threshold at state E_{32}. However, the value of the spike is around 0.9 for the theoretical threshold deferred strategy, 0.5 for the theoretical threshold Poisson strategy and 0.4 for the simulation threshold deferred strategy. Third, the theoretical linear Poisson, theoretical linear deferred and simulation linear deferred distributions show a normal distribution-like shape, with height of around 0.2 and centered in state E_{58}, E_{60} and E_{60}, respectively. Finally, the theoretical and simulation distributions of the deterministic policy show the same shape, that of a flattened normal distribution for the threshold deterministic strategies and similar to an exponential with high values at full-queue for the linear deterministic strategies. However, in spite of having similar shapes, the distributions centers are misplaced by 7 slots in the case of the linear deterministic strategies and do not have equal standard deviations in the case of the threshold deterministic strategies.

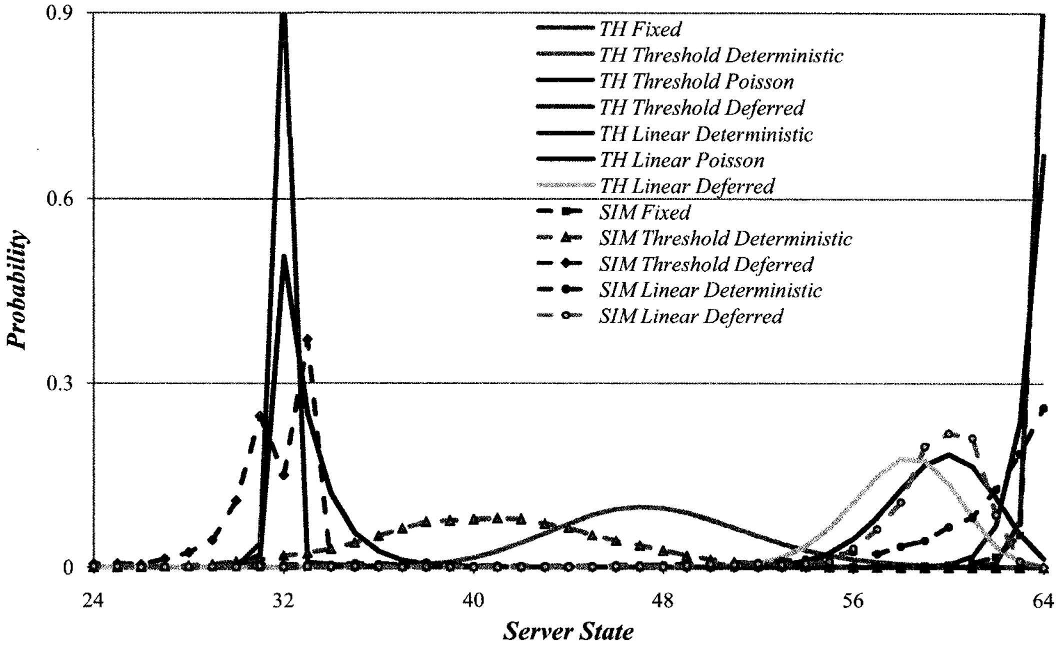

Figure 4.16 Theoretical (solid lines) and simulation (dashed lines) steady-state queue occupation probabilities, for timeout 75 s, empty-queue timeout 75 s, full-queue timeout 1 s, legitimate arrival rate 10 cnx/s, malicious arrival rate 10 cnx/s, legitimate complete rate 1 cnx/s, capacity 64 cnx, threshold at 32 and various timeout strategies. The probability of the server being is states $E_k; k < 24$ is practically null and is not represented in the figure. The occupation probabilities are defined only for integer values of the server states; however in the figure lines used in order to help the reader observe the tendency of each strategy. The queue occupation for the simulation fixed method matches perfectly the queue occupation for the theoretical fixed method. The theoretical threshold Poisson, theoretical threshold deferred and simulation threshold deferred strategy occupations as well as their linear counterparts are very similar. The theoretical and simulation deterministic queue occupation functions are similarly shaped but are translated with up to 7 slots

We conclude that the Poisson assignment policy occupation results are similar to the deferred assignment policy occupation results, which should translate to similar performances for the theoretical Poisson strategies and the simulation and experimental deferred strategies. Also, the theoretical deterministic queue occupation results match the simulation queue occupation results only roughly. This will translate to noticeable performance differences between the theoretical and simulation / experimental results for these strategies.

4.5.3 Transient behaviour

We compare the evolution in time of the legitimate connection success rate observed in five simulations with theoretical results computed using the MOL approximation described in Section 4.4.4. According to the MOL approximation, the legitimate connection success rate should vary from time 0 s during a period equal to the timeout. After the timeout, the steady-state should be reached. The MOL approximation results and simulation results are illustrated in Figure 4.17. Although simulations show an erratic behaviour when looked at individually, the average of the five simulation runs resembles the MOL approximation prediction. Moreover, in average the steady-state is reached sooner than expected by around 33%. Because the convergence occurs so quickly both in theory and simulations, we conclude that steady-state performance measures are relevant to the overall evaluation of the timeout strategies.

4.6 Performance evaluation

Having validated the correctness of the mathematical model, we now proceed to evaluating the performance of the various timeout strategies. To this end, we will compare theoretical and simulation results of attacks against the abstract protocol considered in Section 4.3.1, together with laboratory experiments of SYN-flood attacks as described in (Boteanu *et al.*, 2007b). However, because measuring the performance of the timeout strategies in different environments (legitimate and malicious traffic rate and queue size), we will establish a trade-off between the malicious arrival rate and server capacity.

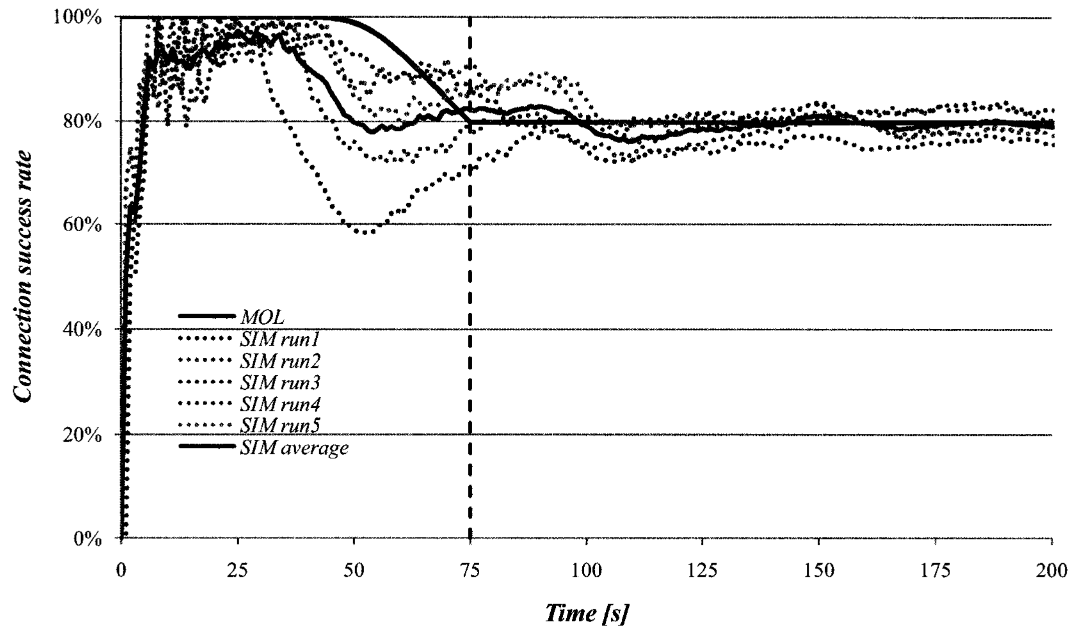

Figure 4.17 Theoretical MOL approximation (solid line) and simulation (dotted lines) transient legitimate connection success rate for the fixed timeout method, for timeout 75 s, legitimate arrival rate 1 cnx/s, malicious arrival rate 1 cnx/s, legitimate complete rate 1 cnx/s and capacity 64 cnx . The dashed line at time 75 s represents that time at which the queue should reach the steady-state, according to the MOL approximation. The red line illustrates the average success rate over the five runs. In simulations, the average success rate converges to the steady-state value quicker than expected, at time 50 s instead of 75 s as predicted by the MOL approximation

4.6.1 Capacity - Attack rate trade-off

We are interested in how the trade-off between the attack rate and server capacity varies for the same legitimate connection success probability, or equivalently for the same connection fail probability. Even though the fully expanded expressions of (4.19), (4.29), (4.38), (4.59) and (4.68) are quite complex, what lies beneath it is a trade-off between these quantities that is essentially linear for the same connection complete probability, as we have verified with several numerical calculations.

Let us define the virulence v of an attack as the attack rate λ_{m} divided by the queue size c. Intuitively, this value corresponds to the frequency (times per second) with which an attack could fill up the queue, or equivalently the number of full queues per second that the attack could saturate. We are interested in knowing whether the performance varies or remains constant for a particular attack virulence. To answer this question, we illustrate in Figure 4.18 the performance of the fixed timeout method for various attack rates and queue capacities. The figure can be compared to a spiral staircase where the red solid lines represent the steps. The steps go down from the capacity y-axis to the attack rate x-axis while turning with 90 degrees. By definition, on each of these steps, the virulence is constant and is represented by the angle of the step projection line in the xy-plane with the capacity y-axis. The question now becomes whether the performance remains constant or not on each of these steps. We observe that on each step the performance is indeed constant as the capacity and attack rate increase, except for maybe very low capacity and attack rate values.

To get a better picture of the linear trade-off, we pick one of the vertical cut planes, at virulence $v = 0.25\ \mathrm{s}^{-1}$, and observe the performance of all strategies as the capacity and attack rate increase, in Figure 4.19. At the leftmost point of the Figure 4.19 the capacity is 0 connections which explains the null performance of all the strategies. Aside from this point, the performance of all strategies remains essentially constant as the attack rate and the queue size increase. The horizontal characteristic of these performance lines, for all virulence values, illustrates the linear trade-off between the capacity and attack rate of all the strategies.

4.6.2 Experimental setup

For the experimental performance evaluation, we choose to implement the scenario of a SYN-flood attack, where the TCP stack of the server is flooded with malicious

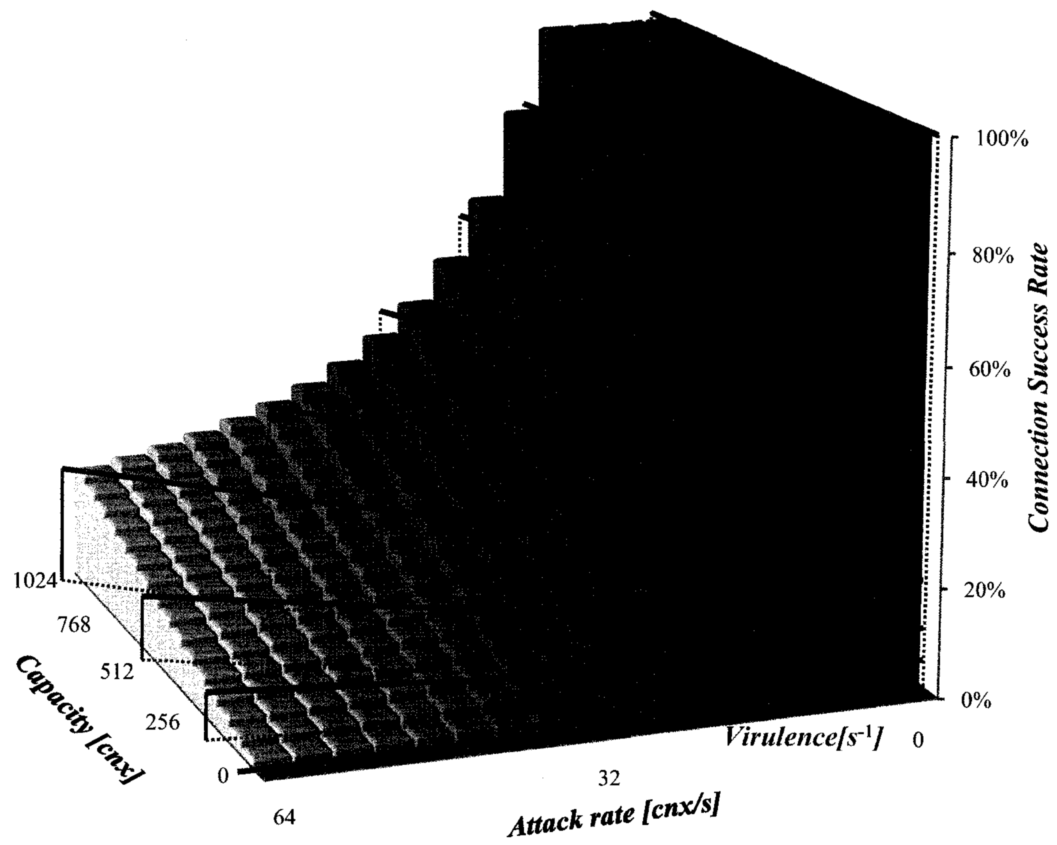

Figure 4.18 Capacity-attack rate trade-off for the fixed timeout method, for timeout 75 s, legitimate arrival rate 0.1 cnx/s and legitimate complete rate 1 cnx/s. On the x-axis, the attack rate varies from 0 cnx/s to 64 cnx/s while on the y-axis, the capacity varies from 0 cnx to 1024 cnx. The z-axis illustrates the legitimate connection success rate. The angular axis in the xy-plane, illustrated by a green arc in the figure, represents the virulence (attack rate divided by capacity). The red planes represent vertical cuts through the solid at constant virulence

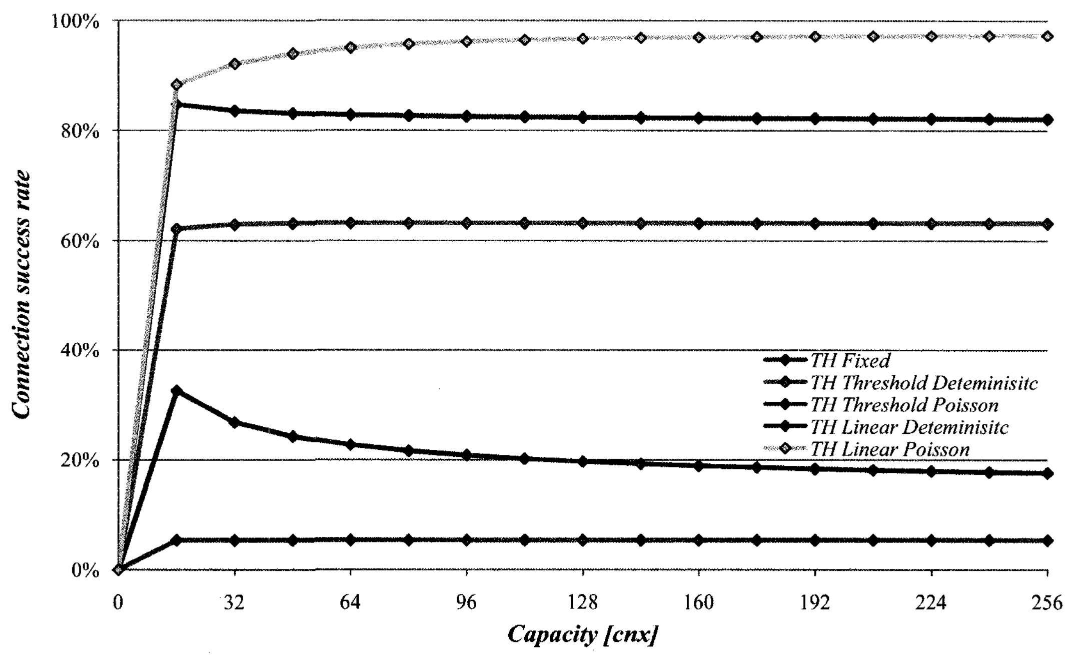

Figure 4.19 Capacity vs. attack rate trade-off at constant virulence 0.25 s^{-1}, timeout 10 s, empty-queue timeout 75 s, full-queue timeout 1 s, legitimate arrival rate 0.1 cnx/s, legitimate complete rate 1 cnx/s, capacity varying from 0 to 256 cnx and attack rate and varying from 0 cnx/s to 64 cnx/s.

SYN messages. The five components of our experimental setup are the following:

1. The attack traffic generator, generating illegitimate SYN packets on the network.

2. The legitimate traffic generator, attempting to establish fully fledged TCP connections.

3. The server, whose TCP stack half-open connection queue is being flooded.

4. The Queue Guardian (QG), a separate application whose role is to protect the server queue.

5. The network, on which both kinds of traffic travel.

Attack Traffic Generator

For this component, we used the IXIA 400T, a special purpose traffic generator chassis, built for performance and conformance testing of network applications. The model we used has four separate Ethernet ports, capable of generating traffic up to 1 Gbps each.

In order to generate the malicious traffic we used the IxExplorer application that runs on the IXIA hardware. Since neither the hardware nor the software can natively generate Poisson traffic, this type of attack was synthesised by cyclically sequencing 255 different *modes*, each mode consisting in sending one single SYN packet. For each attack rate, pauses between modes were statically set to random values following an exponential distribution. We performed a Kolmogorov-Smirnov test on the inter-arrival times of the IxExplorer-generated traffic measured on the server. The maximum difference between the theoretical exponential and the observed CDF was as low as 0.12 for an attack of 1000 packets/s, which confirms that the traffic follows the Poisson process model closely.

Legitimate Traffic Generator

We used a home-made C++ application to generate the legitimate traffic necessary for successful TCP handshake. Both the SYN and ACK messages were sent with exponentially distributed inter-arrival times. Contrary to TCP stack implementations in standard OS, this test application will *not* send a SYN retry message if there is no response from the server. This was a deliberate choice meant to keep the connection attempt rate constant and independent of the connection complete rate. For performance measuring purposes, all the legitimate SYN messages came from the same IP address. This address is discriminated only when counting the total number of legitimate connection attempts. After a TCP handshake is completed, the application will send a RST message in order to free the connection on the server side. We deployed the legitimate traffic generator on a dedicated machine running Gentoo Linux, with 2 GB of memory.

The server whose TCP stack is flooded is also a Gentoo Linux, with 2 GB of memory, which allowed us to experiment with queue sizes up to 16384.

Queue Guardian (QG)

Rather than modifying the TCP stack kernel code, which is neither easy nor practical in real-life deployments, we chose to implement the dynamic timeout strategies on a separate application, in a manner transparent to the server and the legitimate clients. The QG has four different roles:

1. It maintains an up-to-date mirror of the server queue. This is achieved by sniffing the network connection and interpreting packets being send and received by the server. We used the `libpcap` library to sniff all IP packets on the network.

2. It drops connections from the mirror queue, according to the chosen dynamic timeout method and timeout assignation policy.

3. It forces the server queue to drop the same connections that were dropped from the mirror queue. This is achieved by sending RST packets to the server. The IP and TCP headers are spoofed so that the message appears to come from the original client. In order to send the spoofed RST packets at high speeds, this role was implemented using raw sockets.

4. It regularly logs the state of the queue as well as the number of different types of packets sniffed on the network. This log is used later for evaluating the performance of the timeout strategy under test.

For the deterministic policy, we used a priority queue implemented as a red-black tree to store the connections, ordered by their expiration time. When all legitimate connections get served, the complexity of the algorithm is $O(\log cN_m + cN_l)$, where c is the size of the server queue and N_m and N_l are the number of SYN-ACK responses sent to malicious and legitimate SYN packets, respectively. For the deferred policy, only the oldest connection in the queue needs to be analysed: if it is present in the queue for longer than the current timeout, it will be dropped from the queue. Hence, a single FIFO ring-buffer can be used to implement this policy. When all legitimate connections get served, the complexity of the algorithm is $O(N_m + cN_l)$. In practice, however, the legitimate connections are almost always at the end of the queue so only $N_m + N_l$ atomic operations need to be performed. Finally, for performance reasons, we chose to implement each of these four roles in separate threads in the QG application. The QG is run on a separate machine, based on a Intel Core 2 Duo processor at 2.16 GHz.

Network Setup

A 16-port gigabit switch (Linksys SRV-2016) was used to connected all these components together. The legitimate traffic generator machine, the server and the IXIA traffic generator were each connected to a separate port on the switch. For sniffing purposes, the QG machine was connected on a switch port setup to mirror the server port. For sending RST packets, a separate card on the QG machine was connected to another network port on the switch. Other deployment schemes are possible as well and are discussed in (Boteanu *et al.*, 2007b). Figure 4.20 illustrates the network connections between the components we have used.

Testing Methodology

In all the experiments we ran, the following steps were followed in sequence:

1. The server queue size was configured with the value required for testing.
2. The server timeout was configured to be at least as long as the longest timeout on the QG. This way, all the connections drops are triggered by the QG.
3. The legitimate connection traffic generator was started with the connection arrival and connection completion rates required for testing.
4. The QG was configured with the required parameters and started.

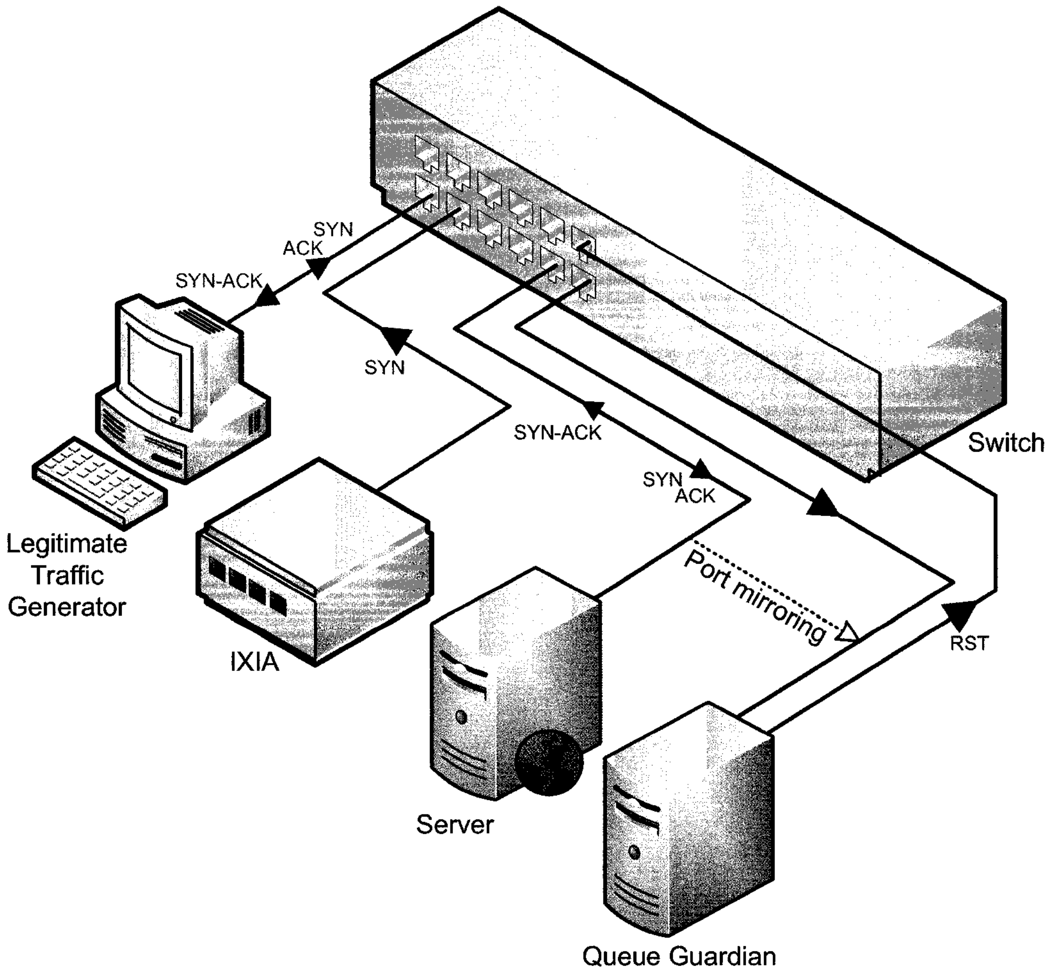

Figure 4.20 Experimental lab network setup

5. The attack traffic parameters were configured in IxExplorer.
6. The attack was started and the experiment was run during a period of time ten times longer than the longest timeout on the QG.
7. The connection success rate was computed based on the QG's log.

Connection completion events correspond to ACK messages being sent to the server. *Legitimate connection arrival* events correspond to SYN messages being sent from the legitimate IP address. The connection success rate was computed as the ratio between number of connections that completed and the number of legitimate connection attempts during the attack.

4.6.3 Comparative results

We measure the performance of the two dynamic timeout methods, threshold and linear, along with the fixed timeout method for comparison purposes using the theoretical model, simulations and experiments. For the dynamic methods, we theoretically evaluated the Poisson and deterministic assignment policies and measured deterministic and deferred assignment policies by simulations and experiments. The attack traffic was generated having the exponentially distributed malicious connections inter-arrival times. We tested the attacks against a small queue size of 128 cnx and a more reasonable queue size of 1024 cnx and explored virulences from 0.015 s^{-1} to 8 s^{-1}. The corresponding attack speeds varied from 2 cnx/s to 1024 cnx/s when testing against a queue size of 128 cnx, and from 32 cnx/s to 8192 cnx/s when testing against a queue size of 1024 cnx. The legitimate connection attempt rate was 10 cnx/s and the mean RTT time for the legitimate traffic was 200 ms (as observed experimentally in (Shakkottai *et al.*, 2004)). The fixed timeout strategy used a timeout value of 10 s and the dynamic timeout strategies used empty- and full-queue timeout values of 10 s and 200 ms, respectively. Results for the tests against a queue size of 1024 are shown in Figure 4.21.

Overall, the experimental results are very similar to the simulation results, and this for both queue sized considered. The average difference between the simulation and experimental results is 2%. The greatest discrepancy (17%) was measured for the linear deterministic strategy faced with an attack of virulence 8 s^{-1} against a queue size of 1024. The standard deviation for both simulation and experimental results

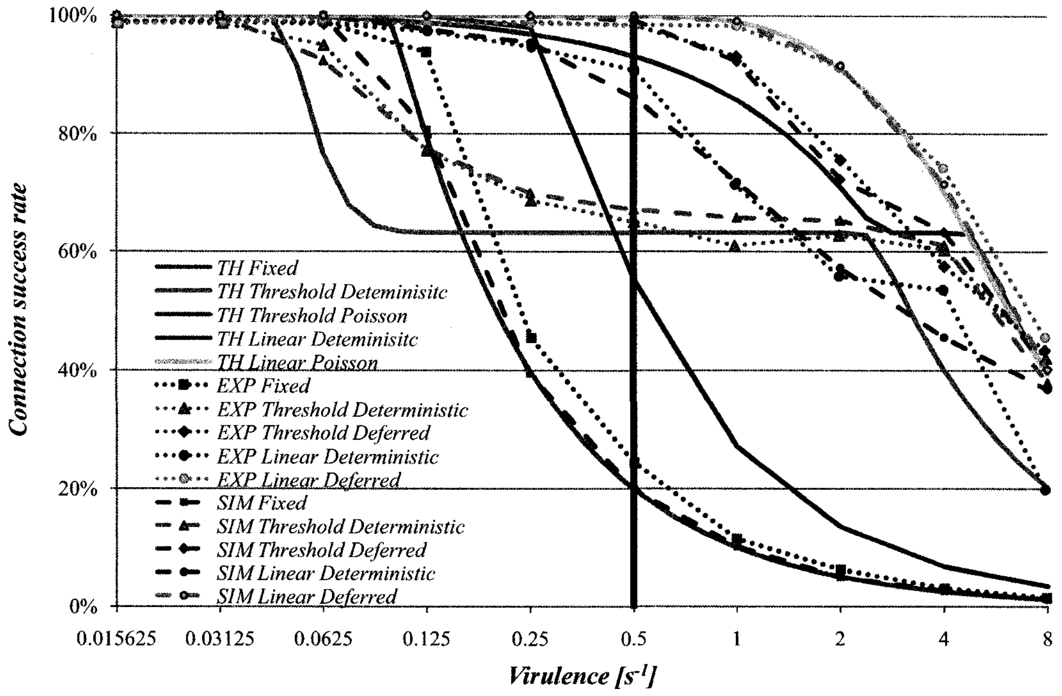

Figure 4.21 Theoretical (TH), simulation (SIM) and experimental (EXP) performance comparison at steady-state, for timeout 10 s, empty-queue timeout 10 s, full-queue timeout 0.2 s, legitimate arrival rate 10 cnx/s, malicious complete rate 5 cnx/s, capacity 1024 cnx, threshold at 512 and malicious attack rate varying from 16 cnx/s to 8192 cnx/s. The theoretical values are illustrated by solid lines, the simulation values by dashed lines and the experimental values by dotted lines. Theoretical, simulation and experimental values for a strategy are represented with the same colour, except for the theoretical Poisson assignment policies which are equivalent to the simulation and experimental deferred assignment policies

was always lower than 3%. Theoretical results are also very similar to experimental and simulation results except for the deterministic assignment policy. The fixed and linear Poisson strategy theoretical results match closely the fixed and linear deferred simulation and experimental results, as expected from the similar steady-state server occupation distributions. The threshold Poisson theoretical results are also close to the threshold deferred simulation experiments results, the greatest discrepancy (7%) begin measured for virulences between 0.5 s^{-1} and 1 s^{-1}. The theoretical results for the threshold deterministic strategy follow the same behaviour as its simulation and experimental counterparts, with differences of up to 22% being measured for virulences of 0.1 s^{-1} and 8 s^{-1}. Finally, the theoretical linear deterministic does not match its simulation and experimental counterparts for other than low virulences. We consider the theoretical results obtained for the deterministic policy unreliable and do not use them when evaluating this strategy. The error is caused by the FILO approximation which has a greater impact on the linear method than on the threshold method. The threshold method is less sensitive to the steady-state server occupation probability distribution than the linear method, because the threshold method only depends on the CDF value at the threshold state E_S. What specific slots are occupied at the left and at the right of the threshold state do not influence by any means the value of the timeout, which makes the threshold method easier to approximate.

As anticipated from previous work, results for low and high virulences are not interesting. For low virulence values (< 0.05 s^{-1}) the attack is not strong enough to degrade QoS at the the server, even when using the fixed timeout strategy. For very high virulence values (> 8 s^{-1}) the attack is so strong that none of the dynamic timeout strategies can maintain a connection success rate greater than 50%. In between these values, in what we call the *window of interest*, several interesting conclusions can be drawn about the relative performance of the various strategies.

First, the dynamic timeout strategies perform better (or no worse) than the fixed timeout strategy. We measured differences of up to 85% between the linear deferred strategy and the fixed timeout strategy, and up to 50% between the threshold deterministic and the fixed timeout strategy around virulences of 1 s^{-1}. Second, the deferred policy always performs better than the deterministic policy. Differences up to 30% can be observed between the deferred and the deterministic policies around virulences of 2 s^{-1}. This is due to the fact that the deferred policy is more reactive, deciding whether a connection should expire or not based on the current status of the queue, as opposed to the status of the queue at the time of the connection arrival in the case of the deterministic policy. Third, the linear method performs better than the threshold method except when it is used with the deterministic policy and for virulence values greater than 1 s^{-1}. The threshold method generally displays an over-protective behaviour, which has the effect of correcting some of the delayed reactivity of the deterministic policy for medium and high virulences.

4.6.4 Attack model variation

In order to study the generality of the previous results with respect to different attack types we also used a deterministic process to generate bursts attacks. In the *burst* attack model, the illegitimate connection requests arrive in (almost) instantaneous bursts of a fixed number of attempts, with burst spaced at a fixed burst interarrival time (BIT). A virulence of 0.5 s^{-1} was chosen, which corresponds to attack rates of 64 cnx/s and 512 cnx/s when testing against queue sizes of 128 cnx and 1024 cnx, respectively. The average connection success rates over 9 experimental and simulation runs for the queue of size 1024 cnx are illustrated in Figure 4.22. The vertical black line at BIT = 0.015625 s in Figure 4.22 represents that the packet inter-arrival time is the same as the mean packet inter-arrival time in the Poisson experiments at virulence 0.5 s^{-1}, marked by the vertical black line in Figure 4.21. The violet vertical line at BIT = 2 s in Figure 4.22 marks the point where one single burst would fill up an empty queue entirely. Figure 4.23 offers a three-dimensional illustration of the correspondence between the Poisson attack and the burst attack figures.

Two "phases" can be observed when analysing the burst attack results. The "liquid phase", at the leftmost part of the figures, with BIT $<$ 2 s, corresponds to attack traffic bursts smaller than the queue size. The "solid", rightmost phase, for BIT $>$ 2 s, corresponds to attack traffic bursts greater than the queue size. The resonance effect is created at BIT = 2 s, corresponding to bursts of the same size as the server queue. In simulations, the fixed timeout strategy performance is practically null at this value. This is due to the fact that the simulated attack and legitimate traffic start at the same time and the attack burst instantly fills up the entire queue. During a period of 10 s, equal to the timeout value, the queue is full and no legitimate connection attempts can be processed. After this period, exactly after the malicious connections are dropped from the queue, the following burst arrives and fills up all the

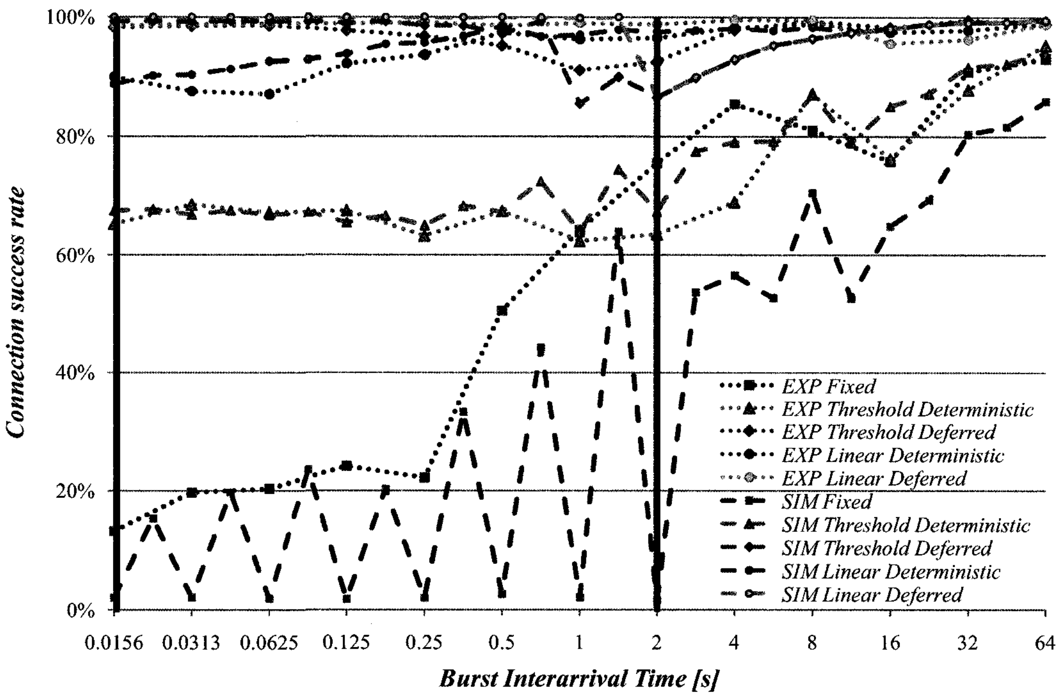

Figure 4.22 Legitimate connection complete rate for various strategies against burst attacks, with fixed queue size of 128 cnx, legitimate arrival rate 100 cnx/s, legitimate complete rate 5 cnx/s, empty- and full-queue timeout values 10 s and 0.2 s, respectively, virulence 0.5 s^{-1} for various burst inter-arrival times (x-axis), over 9 experimental and simulation runs

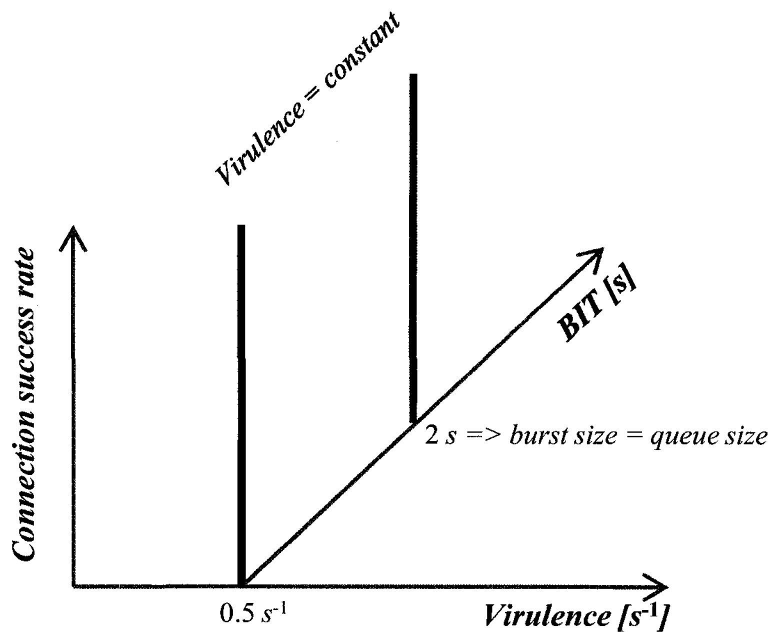

Figure 4.23 Relationship between Poisson attack parameters of Figure 4.21 represented here on the xz-plane, and burst attack parameters from Figure 4.22, yz-plane

queue once again. This happens when the burst traffic is perfectly synchronized with the queue timeout, as is the case with the simulator. In experiments, however, we do not observe the same behaviour.First of all, the legitimate traffic and the malicious traffic are not synchronised. By the time the first attack burst arrives, around three slots in the queue are already used by legitimate connection, so three of the attack packets are discarded by the server. During a period of 10 s, only the number of slots used by legitimate connection at the time the first burst arrived will be available. However, because there are only 10 legitimate connection attempts per second, and because the legitimate connections complete rather quickly (5 every second), the few free slots in the server queue are enough for a large percentage of legitimate connection to complete. Furthermore, in experiments, the burst are never instantaneous due to packet transmission times and eventual collisions in the Ethernet network. This allows for legitimate connection to infiltrate the burst and thus reduce the burst efficiency for the attacker. Due to the above mentioned factors, we can say that the network acts as a "low-pass filter" thus greatly diminishing the resonance effect. In simulations, the fixed timeout strategy is influenced by the resonance effect with "harmonics" at BIT $= 2^{-k}$ s, for $k = \{0..5\}$. In experiments, however, the resonance effect is absorbed by the network. The only two strategies that seem to be slightly affected by the resonance effect in experiments, are the linear deterministic and the threshold deferred timeout strategies, and this only for the harmonic at BIT $= 1$ s.

Is it important to note that the deferred policy, which performs better than the deterministic one, is also more robust and consistent, having lower standard deviation values. The fixed timeout strategy, on the other hand, is the most unstable, both in simulation and in experiments, with maximum standard deviation values of over 10%.

4.6.5 Parameter optimisation

No matter what timeout method is used, the choice of the timeout parameters can influence the performance. For this reason, it is logical that we try to understand how these values should be chosen and how much can be gained from a careful choice of these values. Therefore, in this section we analyse the two dynamic timeout methods when coupled with the deferred policy as well as the fixed timeout method, for comparison sake. We explore various configuration values using stochastic simulations to get an insight on the optimal timeout parameters.

When employing the fixed timeout method, the only parameter to configure at the server-side is the timeout (T_{out}). For very low T_{out} values, the legitimate connection reject rate is practically null. The server is throwing the connections out of the queue so quickly that the queue never gets the chance to fill. On the other hand, the legitimate connection expire rate is very high, because connections have very little time to complete before the server declares them as expired. For very high T_{out} values, the effect is inverted, where the legitimate connection expire rate is null and the legitimate connection reject rate is high. This happens because connections have a very long time to complete and thus almost never expire. However, because connections are allowed to stay a long time in the queue, the queue easily fills and most of the connections arriving at the queue are rejected. The optimal T_{out} value is somewhere in-between, where the sum of the expire and reject rates is minimum, and depends on the legitimate and malicious traffic rates as well as the server queue size. Although we offer no analytical expression for the optimal T_{out} value, it can be computed numerically from (4.19). The behaviour of the fixed timeout method when varying the timeout is illustrated in Figure 4.24.

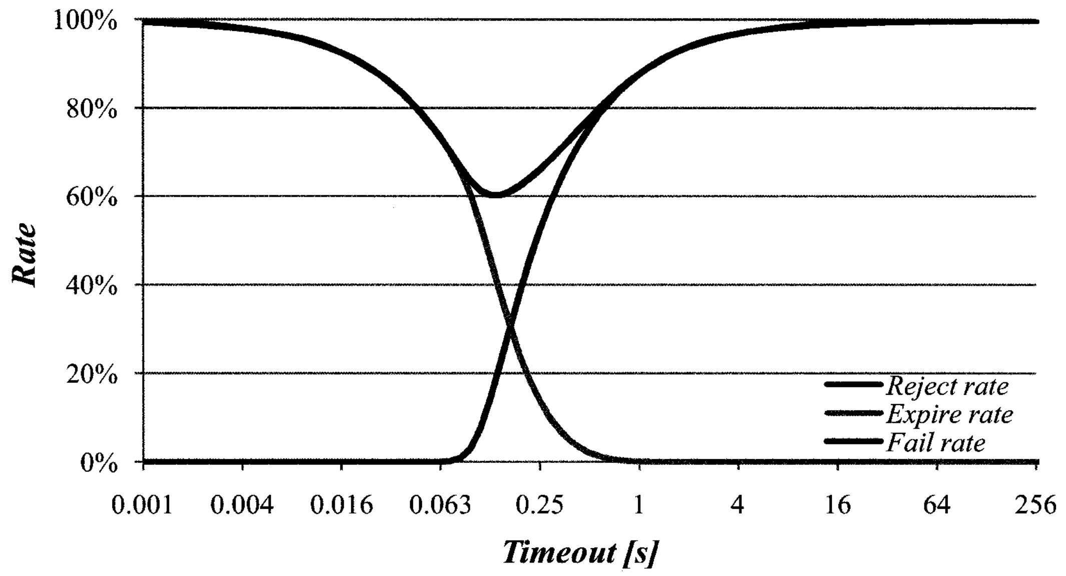

Figure 4.24 Legitimate connection reject (φ_{r}), expire (φ_{e}) and fail (φ) rates for the fixed timeout method, for capacity 32 cnx, legitimate arrival rate 10 cnx/s, malicious arrival rate 256 cnx/s, legitimate complete rate 5 cnx/s and timeout varying from 2^{-10} s to 2^{10} s. For very low timeout values, the expire probability is high, whereas for very high timeout values the reject probability is high. The optimal timeout value $T_{\mathrm{out}}^{(\mathrm{opt})}$ is 0.13 s and the success rate $1 - \varphi$ for this timeout is 40%

When using the threshold and linear timeout adjustment methods, the parameters that can be configured at the server-side are the initial, empty-queue, timeout T_0, the full-queue timeout T_1 and, only for the threshold method, the threshold S. For the sake of comparison, in our case, we set the threshold at half the size of the queue ($S = c/2$) and compare the two dynamic timeout strategies when varying the two timeout values, T_0 and T_1. The behaviour of the linear method, illustrated in Figure 4.25, presents similar characteristics to that of the fixed method. When low values are used for both T_0 and T_1, the reject rate is low but the expire rate is high. When high values are used for both T_0 and T_1, the expire rate is low but the reject rate is high. The special case where T_0 and T_1 are equal is equivalent to the fixed timeout method and can be observed in the plane described by the z-axis and the first diagonal of xy-plane. The case where the T_1 is higher than the T_0 is not valid as we required the timeout the be lowered as the queue fills. This corresponds to the blank portion in Figure 4.25. However, the case where T_0 is high and T_1 is low does not have any equivalent when using the fixed method. Interestingly, in these conditions, the reject rate is very low and the expire rate is lower than its maximum. This can be explained as follows: As the queue fills, the timeout is decreased, which makes it more unlikely for the queue to get filled entirely. Hence, new connections are almost always accepted in the queue which makes the reject probability low. On the other hand, the timeout is not low all the time, only when the queue holds a lot of connections. The rest of the time, the timeout is high enough so that some legitimate connections that entered the queue complete, hence the expire rate is relatively low as well. The legitimate connection success probability for the linear deferred strategy is illustrated in Figure 4.26. Although the maximum performance that the linear deferred strategy can offer for this particular traffic rates and queue size is not much higher than that of the fixed strategy, the former is much more robust to parameter configuration. The greatest advantage of the linear deferred strategy over the fixed strategy is that not only do high T_0 and low T_1 values provide the best performance under attack, these same values are optimal in non-attack scenarios This is something that we verified empirically for various traffic rates and queue sizes.

The behaviour of the threshold deferred strategy illustrated in Figures 4.27 is somewhat similar to that of the linear deferred strategy in the sense that low T_0 and T_1 values as well as high T_0 and T_1 values provide bad performance but high T_0 and low T_1 values are a good compromise between the reject and expire rates. However,

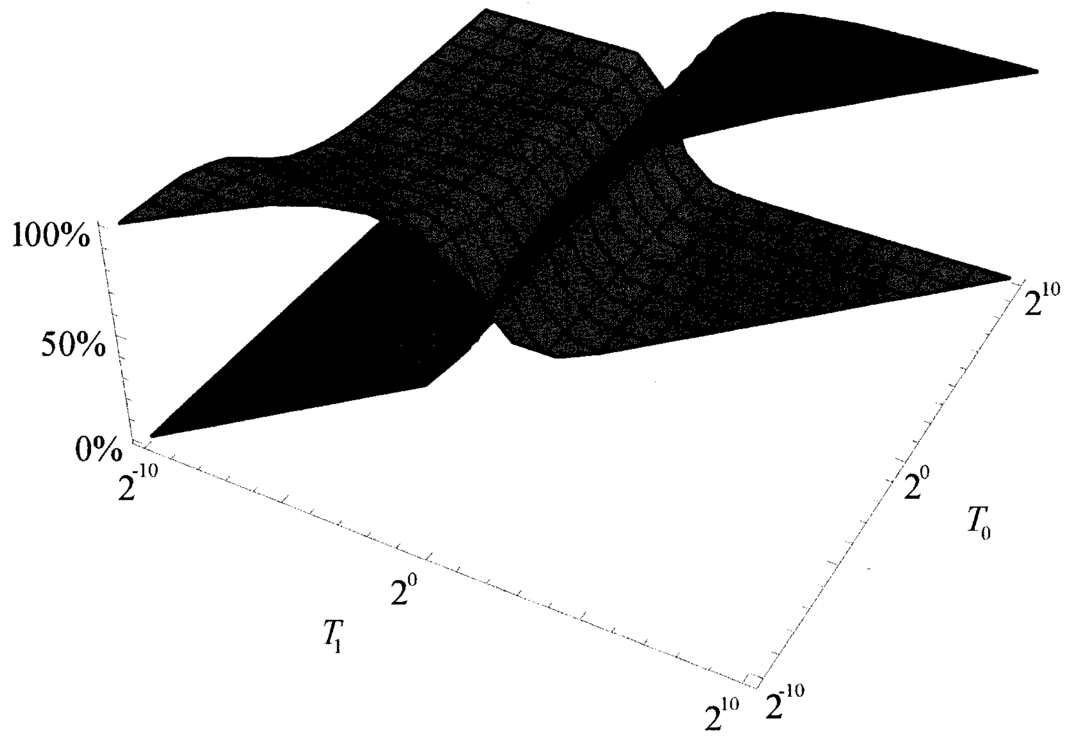

Figure 4.25 Legitimate connection reject (blue surface) and expire (green surface) rates for the linear deferred timeout strategy, for capacity 32 cnx, legitimate arrival rate 10 cnx/s, malicious arrival rate 256 cnx/s, legitimate complete rate 5 cnx/s and empty- and full-queue timeout varying from 2^{-10} s to 2^{10} s. The reject and expire rates of the fixed timeout method illustrated in Figure 4.24 are a specific case of this figure, observed in the plane defined by the z-axis and the diagonal of the xy-plane

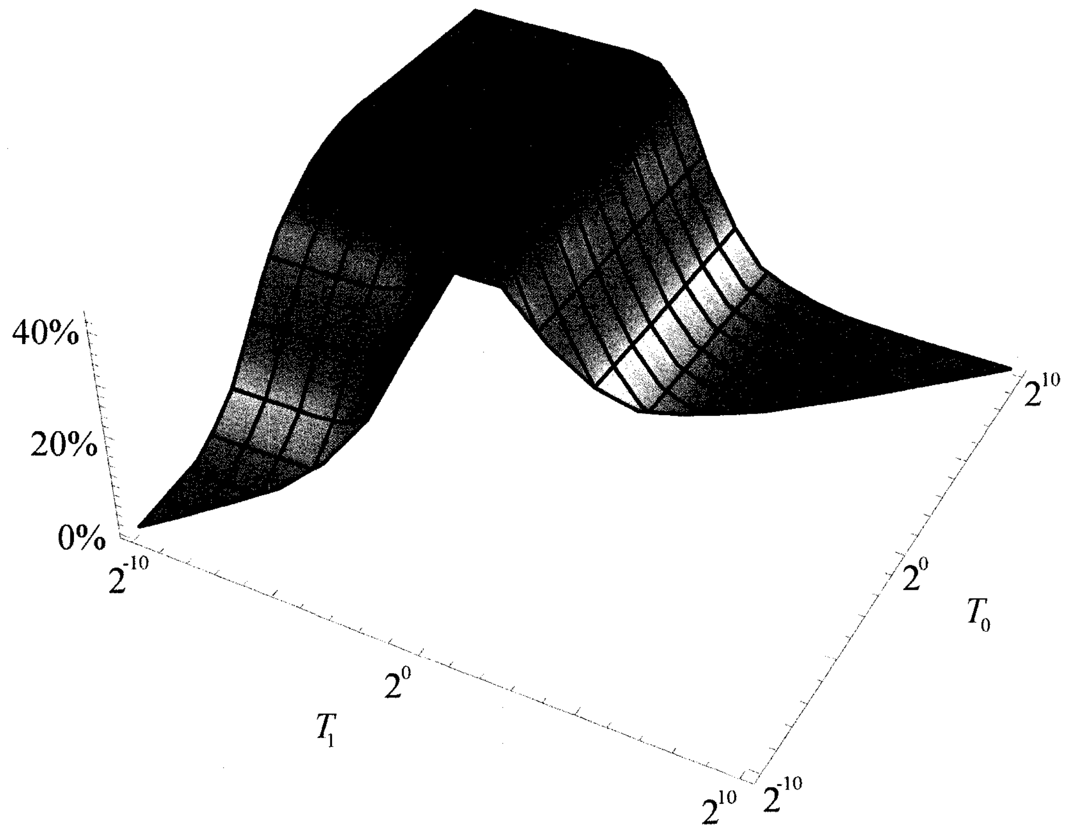

Figure 4.26 Legitimate connection success rate for the linear deferred timeout strategy, for capacity 32 cnx, legitimate arrival rate 10 cnx/s, malicious arrival rate 256 cnx/s, legitimate complete rate 5 cnx/s and T_0 and T_1 varying from 2^{-10} s to 2^{10} s. Low T_0 and T_1 values and high T_0 and T_1 values offer bad performance, painted with red on the figure. The best performance painted with blue on the figure, with success rates of up to 43%, is obtain for high T_0 values and low T_1 values

when looking and the overall performance of this strategy in Figure 4.28 we observe that high T_0 and low T_1 values do not offer optimal performance, as it was the case with the linear deferred strategy. On the contrary, the maximum performance is equal to that of the fixed strategy and is obtained regardless of T_0, for a T_1 value equal to the optimal T_{out} values for the fixed strategy. This can be explained by the fact that when the server is under attack and the threshold is crossed, the timeout T_1 is used. Therefore, the optimal T_1 in this specific conditions is the same as the optimal T_{out} for the fixed strategy in the exact same conditions. Although this sounds somewhat disappointing, there are two advantages that the threshold method has over the fixed method. First, after the attack stops, the threshold method uses T_0 as the timeout value which performs good in these conditions. On the other hand, the fixed method would keep using the same unnecessarily restrictive timeout, something that is not optimal any more in non-attack conditions. Second, a plateau is observed for high T_0 and low T_1 values, similar to that of the of the linear deferred strategy, with the exception that the performance is not optimal for these values. In most of the times, it is impossible to compute the optimal timeout value for the fixed method, so reaching the plateau could be, although not optimal, good enough performance.

4.6.6 Traffic rate variation

Having analysed what the performance impact of varying the timeout parameters is, we now proceed to measuring the sensitivity of the timeout counter-measures when varying the legitimate and malicious traffic rates, using stochastic simulations.

First, for the fixed timeout method, we observe that the success rate is 100% when both the legitimate and malicious traffic rates are very low. As the legitimate traffic rate increases, the performance continues to be high, up to the point where the queue is saturated, as illustrated in Figure 4.29. In our case, this happens for a legitimate arrival rate around 128 cnx/s. However, when the malicious traffic rate reaches as low as 8 cnx/s, the performance drops under 50% regardless of the legitimate traffic rate. Note that the traffic rates exhausting all the resources are low because, for this analysis and without loss of generality (due to the results of Section 4.6.1), we chose a relatively small queue size, of only 32 cnx.

We analyse the threshold deferred strategy in the same traffic conditions used previously for the fixed method. In this case, however, we choose to use very high T_0

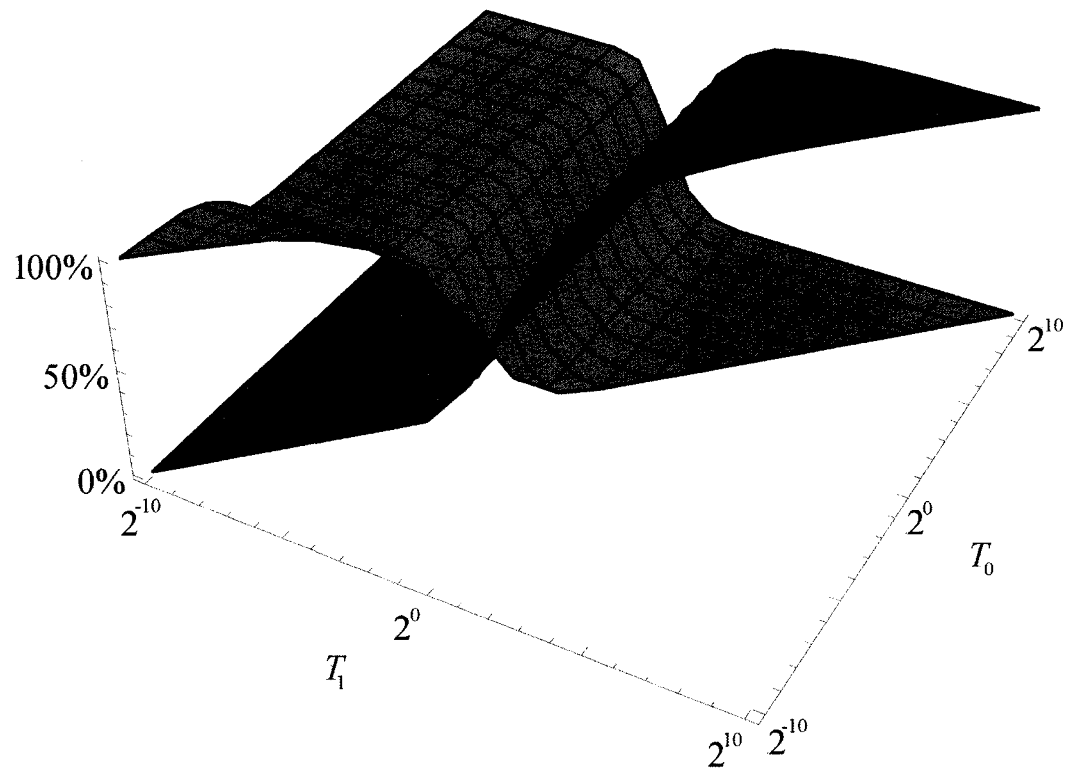

Figure 4.27 Legitimate connection reject (blue surface) and expire (green surface) rates for the threshold deferred timeout strategy, for capacity 32 cnx, legitimate arrival rate 10 cnx/s, malicious arrival rate 256 cnx/s, legitimate complete rate 5 cnx/s and T_0 and T_1 varying from 2^{-10} s to 2^{10} s.

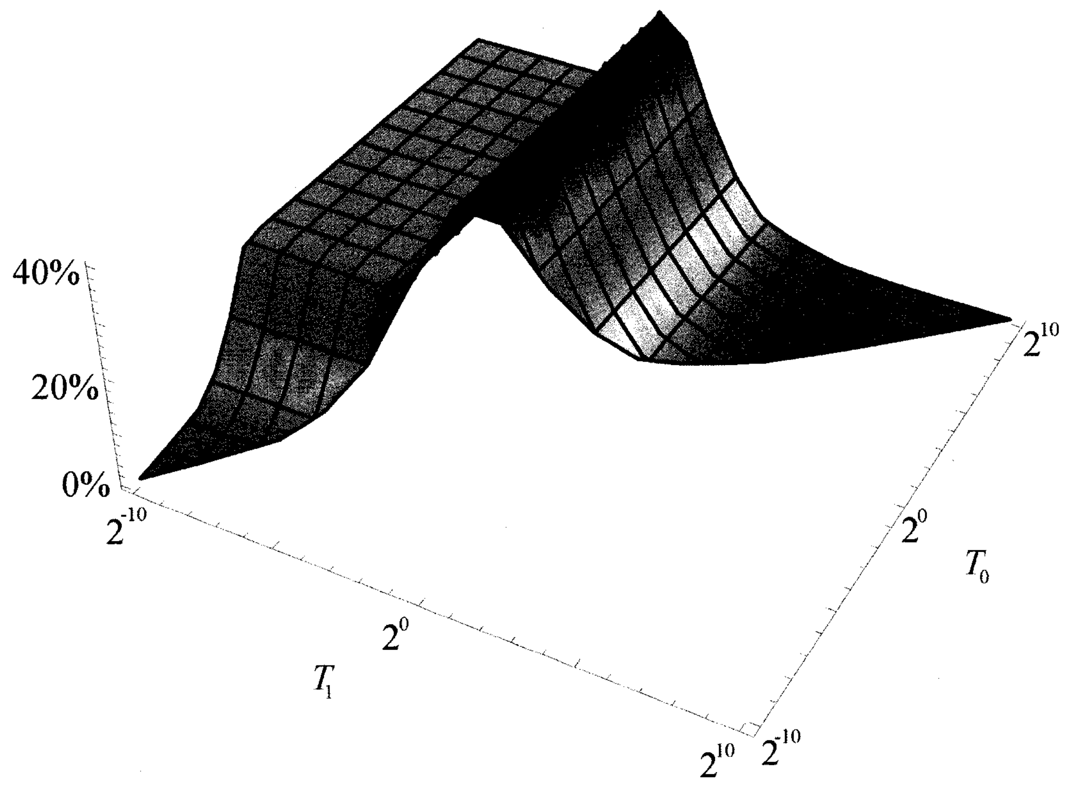

Figure 4.28 Legitimate connection success rate for the threshold deferred timeout strategy, for capacity 32 cnx, legitimate arrival rate 10 cnx/s, malicious arrival rate 256 cnx/s, legitimate complete rate 5 cnx/s and T_0 and T_1 varying from 2^{-10} s to 2^{10} s. Similar to the linear method, low T_0 and T_1 values and high T_0 and T_1 values offer bad performance, painted with red on the figure. The best performance painted with blue on the figure, with success rate of 40%, is obtain for T_1=0.13 s. The plateau for high T_0 and low T_1 has success rate of 25%

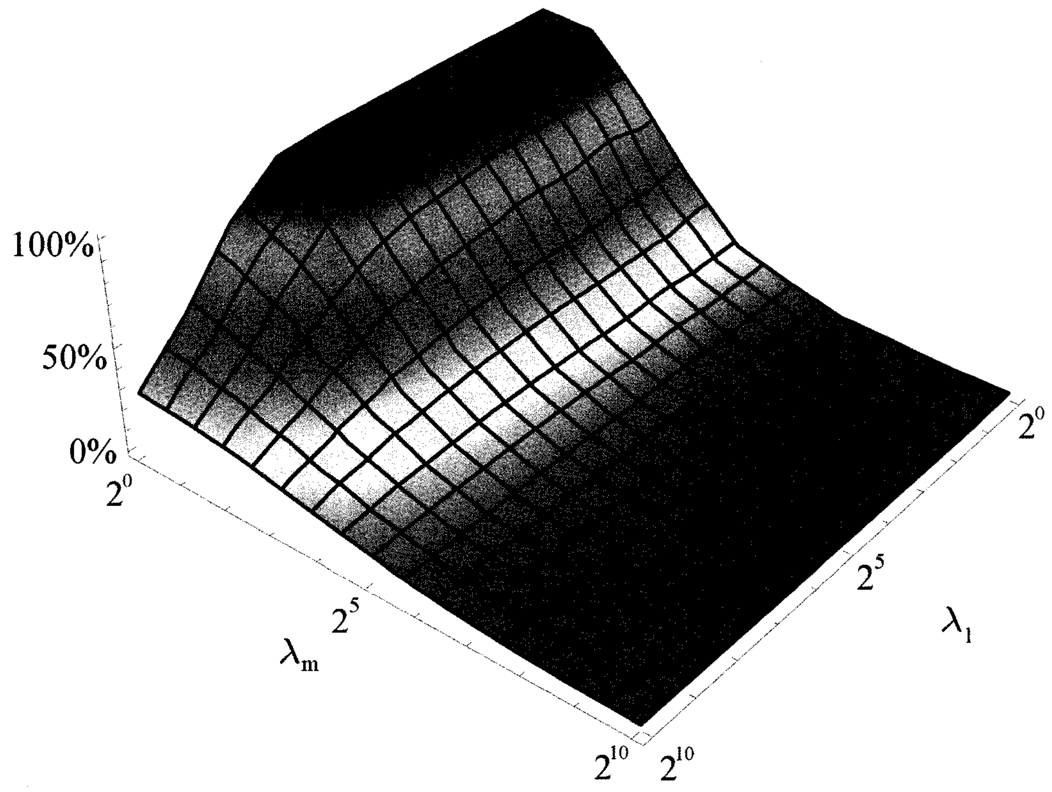

Figure 4.29 Legitimate connection success rate for the fixed timeout strategy, for capacity 32 cnx, legitimate complete rate 5 cnx/s, timeout 10 s and legitimate and malicious arrival rates varying from 2^0 cnx/s to 2^{10} cnx/s

and very low T_1 timeout values, which we previously established to be non-optimal but good overall configuration parameters. The results illustrated in Figure 4.30 show that the threshold deferred strategy maintains a performance level above 50% for attacks up to 64 cnx/s. However, when there is very little attack traffic but the legitimate traffic rate is very high, in our case 1024 cnx/s, the threshold deferred strategy performs worst than the fixed method. This is due to the fact that very high T_0 and very low T_1 are not optimal timeout values for the threshold deferred strategy. In this particular case, the legitimate traffic rate is so high that the server reaches the threshold state and uses the very low timeout on legitimate connections, which causes more damage than a higher timeout would.

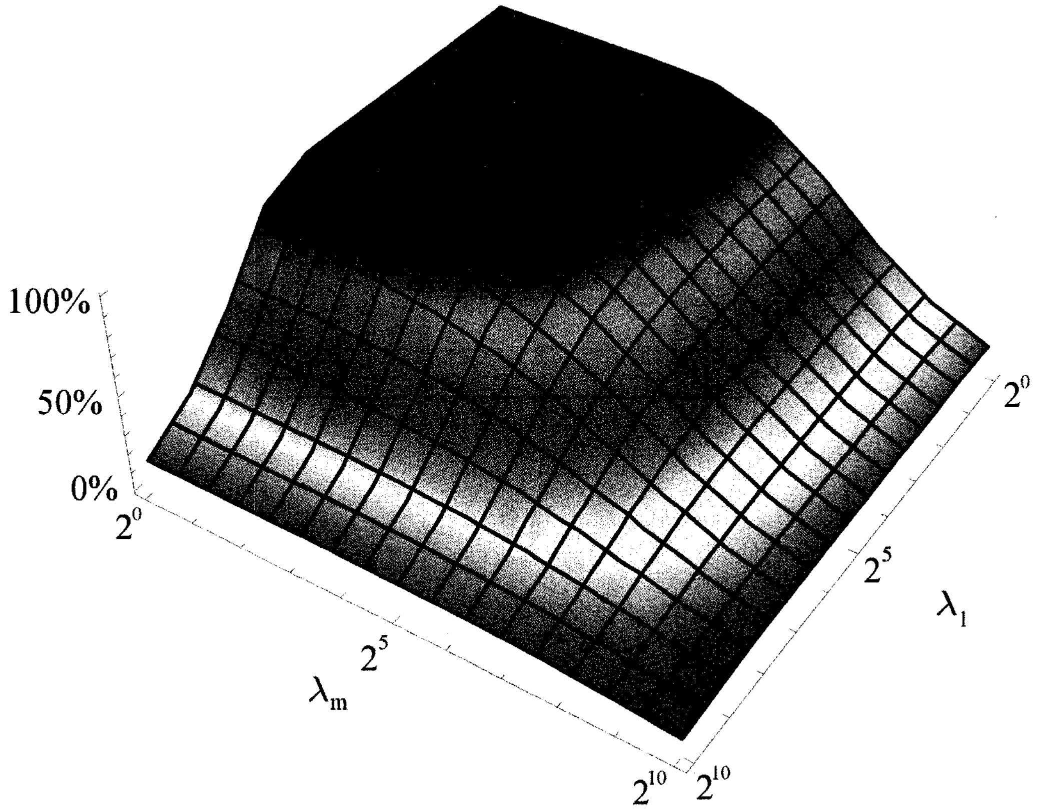

Figure 4.30 Legitimate connection success rate for the threshold deferred timeout strategy, for capacity 32 cnx, legitimate complete rate 5 cnx/s, $T_0 = 2^{-10}$ s, $T_1 = 2^{10}$ s and legitimate and malicious arrival rates varying from 2^0 cnx/s to 2^{10} cnx/s

Finally, we analyse the linear deferred strategy with the same very high T_0 and very low T_1 timeout values. As opposed to the threshold deferred strategy, the T_0 and T_1

values chosen are now optimal. The performance of the linear deferred strategy under varying traffic rates is illustrated in Figure 4.31. The linear deferred strategy does not exhibit the same low performance effect for high legitimate and low malicious traffic rates. On the contrary, the linear deferred strategy outperforms the fixed method and the threshold deferred strategy for all traffic rates. Moreover, a performance increase of up to 14% when compared to the fixed method is measured, when the legitimate traffic is high (128 cnx/s) and the malicious traffic is low (1 cnx/s), precisely those rates for which the threshold deferred strategy displayed significantly decreased performance.

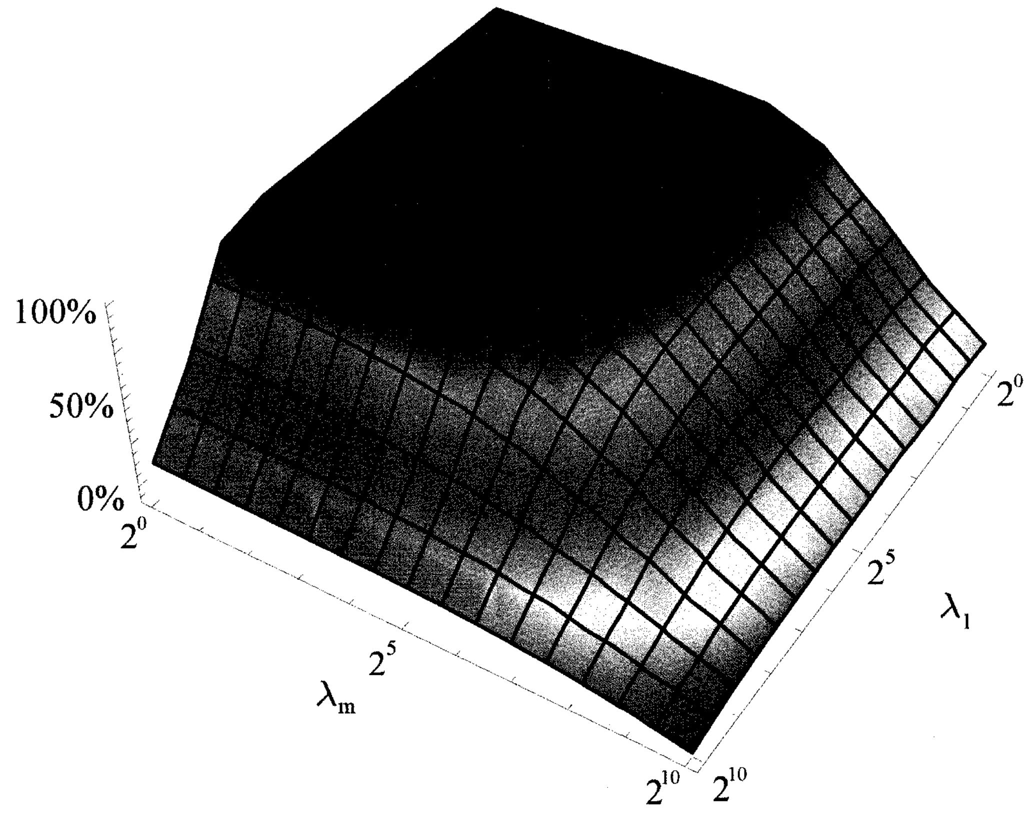

Figure 4.31 Legitimate connection success rate for the linear deferred timeout strategy, for capacity 32 cnx, legitimate complete rate 5 cnx/s, $T_0 = 2^{-10}$ s, $T_1 = 2^{10}$ s and legitimate and malicious arrival rates varying from 2^0 cnx/s to 2^{10} cnx/s

4.7 Conclusions and future work

In this paper, we have studied the performance of different queue management strategies against DoS attack that try to exhaust the server connection tracking resources. We modelled the server queue as a Birth-Death Markov chain and analysed in this model three methods of adjusting the timeout value based on the queue occupation: the fixed, the threshold and the linear methods. The last two methods are dynamic timeout methods and the different policies that can be enforced for deciding whether a connection expired or not are the deterministic and the deferred policies. For modelling convenience, we also introduced the Poisson policy which we use as an intermediate step for obtaining theoretical results about the deterministic and deferred policies. The deterministic and deferred policies perform non-memoryless operations, which we approximated to memoryless so that we can get an insight on the performance of these policies within our model. Furthermore, we tested the performance of all three timeout adjustment methods with the deterministic and deferred policies both in stochastic simulations and in laboratory SYN-flood experiments.

Under the simplifying hypothesis that the traffic seen by the server arrives with exponentially distributed inter-arrival times, we confirmed the consistency of the theoretical results when compared to simulation and experimental results. The Poisson policy generated very similar results to the deferred policy both when comparing the theoretical and simulation steady-state queue occupation probabilities and when comparing the theoretical, simulation and experimental overall connection success rate. This is interesting because the Poisson policy is very intuitive to model where as the deferred policy is easily and efficiently implementable in real-life queue management mechanisms.

Overall, the theoretical results matched the simulation and experimental results very closely except for the deterministic policy. On one hand, the threshold deterministic theoretical results followed the trend of the simulation and experimental results but with relatively high errors. On the other hand, the linear deterministic theoretical results did not match the simulation and experimental results for other than low virulence values. This is clearly due to the FILO approximation which has a lower impact on the threshold method than on the linear. The threshold method only changes the timeout between two values which makes it less dependent on the queue occupation and thus easier to approximate.

In order to validate the robustness of our mathematical model in the light of attacks other than the simplistic Poisson model, we evaluated the performance of the different timeout strategies when the malicious connections arrive in deterministic bursts of various sizes and at various intervals of time, by using stochastic simulations and in-laboratory SYN-flood attacks. Whether with respect to the Poisson or the burst attack models, we can draw several conclusions related to the performance of the different timeout strategies which are consistent for simulations, experiments and theoretical results:

I. **Capacity vs. attack rate trade-off**. There exists a linear trade-off between the server resources (number of slots available in the queue) and the attacker resources (malicious connection arrival rate). For this reason, we defined the virulence of an attack, as the attack rate divided by the server capacity. At constant virulence, as the attack rate and queue size increase, the performance of all strategies remains constant (at different values for each strategy).

II. **Fixed vs. dynamic timeout methods**. Using a dynamic timeout strategy is always a good idea. The only exception is the threshold deterministic strategy that is overprotective when faced with low virulence attacks.

III. **Threshold vs. linear methods**. The linear timeout adjustment method performs better than the threshold timeout adjustment method. The sole exception occurs when coupling the methods with the less efficient deterministic policy for medium to high virulences. In this case, the overprotective behaviour of the threshold method compensates the low reactivity of the deterministic policy.

IV. **Deterministic vs. deferred policies**. The deferred policy always performs better than the deterministic policy when protecting against Poisson process attacks. Moreover, the deferred policy has a lower computational overhead than its deterministic counterpart.

V. **Resonance effect**. In simulations, we observed a resonance effect for burst attacks with burst interarrival times (BIT) and virulences such that a single burst fills the queue entirely. At these values, the fixed timeout strategy and the dynamic timeout strategies with the deferred policy show a significant performance decrease. However, this effect was not visible in experiments due to the low-pass filter behaviour of the network that absorbed the resonance effect.

VI. **Timeout optimisation**. High empty-queue and low full-queue timeout values are optimal values for the linear deferred strategy, regardless of the legitimate traffic rate, queue size and malicious traffic type and rate. For all other strategies, there exist no universally good values to use in all conditions, except maybe for the threshold deferred strategy where high empty-queue timeout and low full-queue timeout offer non-optimal but reasonable performance when compared with the fixed timeout method.

In essence, all of the relevant findings can be summarised into one single recommendation for protocol designers and system administrators managing the configuration of connection table management in servers and network hardware.

Overall Recommendation: We suggest always using the linear deferred timeout strategy with a high empty-queue timeout value ($T_0 \approx 2^{10}$ s) and a low full-queue timeout value ($T_1 \approx 2^{-10}$ s). These settings will provide the best overall rate of successful connections in all conditions.

It is important to note that the difference in performance between this optimal strategy and optimal parameter choices, and other possible combinations is only significant within the so-called *window of interest*, which ranges between virulence values of 0.05 and 8 s^{-1}. Outside of this approximately 2 orders of magnitude range, the recommendation above is still valid, but is not very useful because all strategies will perform equally well or equally badly. Nonetheless, since the implementation overhead of such strategies with respect to the others is only linear in the queue size, it pays to always use them, even if it is unsure whether the residual traffic rate generated by unfiltered or undectetable connections, will be within this window of interest. And even in the case where the attack will be more virulent than 8 s^{-1}, then the linear tradeoff we have described above indicates that the same quality of service can be maintained by matching the attack rate with a like increase in the size of the queue, which in most protocols and applications will normally be a cheap thing to do, and hence a good defensive choice.

We hope to further confirm and enhance our findings in future work by comparing the linear deferred strategy with high empty-queue and low full-queue timeout values with a similar, yet simpler approach, where the queue would act as a ring buffer. This is based on the intution that the linear deferred strategy with an infinite empty-queue timeout T_0 and very small, tending to zero, full-queue timeout T_1, will essentially behave as such a ring buffer. On the other hand, an implemenation of a ring buffer-based queue strategy will be more lightweight and result in a lower overhead. This hypothesis needs to be confirmed in simulation and tested experimentally.

On the mathematical modelling side, we would like to improve the approximations we made in order to more precisely evaluate the deterministic and deferred timeout assignment policies. On the experimental side, it would important to explore the effect on the performance of dynamic timeout strategies of using different network topologies; for example, this might (or might not) affect the presence and strength of such phenomena as the resonance effect.

Finally, while we measured experimentally the performance of the dynamic timeout strategies only for the SYN-flood attack, in sections 4.3.3 and 4.3.4 we showed how our abstract protocol can be instantiated to reflect the queue behaviour at higher protocol levels. Hence, this work can be easily generalised to devise dynamic timeout implementations for common connection oriented protocols. However, one of the immediate difficulties of doing so is that the standards for most relevant protocols (e.g. HTTP v1.1 (Fielding *et al.*, 1999), TLS v1.1 (Dierks et Rescorla, 2006) and FTP (Postel et Reynolds, 1985)) do not define connection timeout mechanisms. Nonetheless, the applications that implement these protocols do include such timeout mechanisms, and as such the results obtained here can be applied to make them more resilient to the corresponding version of connection depletion attacks. Verifying this intuition for such protocol implementations would greatly increase the applicability and relevance of not only the results described here, but also of the modelling and experimental techniques introduced. It is for that reason, the object of ongoing work by our research group.

Acknowledgements: We would like to thank Fabian Monrose for providing some useful feedback on potential applications of our work to other types of protocols. We would also like to thank and acknowledge the co-authors of the previous versions of this work, John Mullins, John McHugh and Édouard Reich, who have accompanied us in the initial phases of this endeavour.

CHAPITRE 5

Discussion génerale et conclusion

Dans ce mémoire nous nous sommes intéressés à la performance d'un serveur dont un service orienté connexion est sous attaque de déni de service d'épuisement de ressources. Contrairement aux autres approches qui ont été explorées dans le passé, notre solution ne se base pas sur la discrimination du trafic légitime et malicieux. De plus, la solution que nous proposons ne consiste pas à modifier la logique du protocole mais simplement de gérer de façon dynamique l'expiration des connexions dans la file du serveur.

Nous avons modélisé la file des connexions établies ou en cours d'établissement du serveur à l'aide des chaines de Markov. Ce type de chaine a été utilisé dans le passé pour mesurer la performance des équipements réseau. Nous avons adapté le modèle mathématique afin de modéliser, en plus du trafic légitime, le trafic malicieux. Suite à l'analyse du modèle mathématique, nous avons confirmé qu'augmenter la taille de la file des connexions permet au serveur d'être plus résistant aux attaques. De plus, le paramètre évident qui peut être optimisé du coté serveur pour rendre le serveur plus résistant aux attaques de déni de service est le délai d'inactivité.

En plus de la stratégie par défaut qui consiste à garder le délai d'inactivité fixe, nous avons modélisé deux méthodes réactives qui diminuent la valeur du délai d'inactivité au fur et à mesure que la file de connexions se remplit : la méthode *threshold* et la méthode *linear*. Si la méthode *threshold* est déjà implémentée par certains solutions de protection, la méthode *linear* a été introduite par nous. Face à un délai d'inactivité dynamique il existe plusieurs politiques que le serveur peut adopter pour assigner le délai d'inactivité aux connexions dans la file. À notre connaissance, nous sommes les premiers à avoir décrit formellement et analysé les politiques *deterministic* et *deferred* qui sont envisageables d'être intégrées dans des implémentations des mécanismes de gestion de la file de connexions. Pour des fins de simplicité, nous avons aussi introduit et modélisé la politique *Poisson*, qui par contre n'est pas facilement utilisable en pratique.

Pour confirmer la validité du modèle théorique, nous avons développé un simulateur stochastique à base d'événements qui nous est propre et nous avons implémenté les méthodes du délai d'inactivité dynamique ainsi que la stratégie du délai d'inactivité fixe, pour des fins de comparaison. Nous avons testé les politiques d'assignation du délai d'inactivité *deterministic* et *deferred.* Afin de montrer la généralité du modèle et des stratégies du délai d'inactivité dynamique, nous avons reproduit expérimentalement une attaque de type *SYN-Flood.* Pour toutes les expériences et simulations, nous avons effectué plusieurs essais afin d'obtenir la valeur moyenne de la performance des stratégies mais aussi l'écart type de ces valeurs. Les stratégies de protection ont été implémentées dans un composant réseau proche du serveur.

Suite à l'analyse des résultats obtenus à l'aide du modèle théorique, par simulation stochastique et expérimentalement, nous confirmons la validité du modèle mathématique. De plus, nous observons que la politique théorique *Poisson* a un effet très similaire à la politique pratique *deferred.* Cependant, les approximations que nous avons considérées pour calculer théoriquement la performance de la politique *deterministic* introduisent des erreurs trop importantes par rapport aux résultats de simulations et experimentaux, qui rendent les résultats théoriques de la politique *deterministic* inutilisables.

En ce qui concerne la performance des différentes stratégies du délai d'inactivité dynamique, les conclusions que nous tirons sont les suivantes :

Compromis capacité - taux d'attaque. L'intuition du compromis linéaire entre la taille de la file des connexions du serveur et le taux d'arrivée des connexions malicieuses est confirmée. Ceci se traduit par le fait que pour offrir la même qualité de service face à une attaque avec un taux d'arrivée deux fois plus rapide, le serveur doit être dimensionné pour avoir une file de taille deux fois plus grande. Pour cette raison, nous avons introduit la *virulence* d'une attaque comme étant le taux d'arrivée des connexions malicieuses divisé par la taille de la file.

Les méthodes dynamiques versus la méthode *fixed*. Les méthodes dynamiques de gestion du délai d'inactivité offrent toujours une meilleure performance que la méthode *fixed.* La seule exception est la méthode *threshold* pour des attaques peu virulentes ou dans des cas où le taux du trafic légitime est très élevé, dû à un phénomène de type *Flash Crowd,* par exemple. En plus de la meilleure performance, les résultats des méthodes dynamiques sont plus stables statistiquement, dans le sens que la performance est moins variable (écart type plus petit) que celle de la méthode *fixed*, dans les mêmes conditions.

La méthode *linear* versus la méthode *threshold*. La méthode d'ajustement du délai d'inactivité *linear* est plus performante que la méthode *threshold*. La seule exception arrive quand les deux méthodes sont couplées avec la politique d'assignation du délai d'inactivité moins performante, *deterministic*, pour des attaques très virulentes. Ceci s'explique par le fait que le caractère surprotecteur de la méthode *threshold* compense la réactivité réduite de la politique *deterministic*.

La politique *deferred* versus la politique *deterministic*. La politique d'assignation du délai d'inactivité *deferred* est plus réactive que la politique *deterministic* parce qu'elle décide si une connexion doit expirer ou pas en fonction de l'état actuel de la file et non pas de l'état de la file au moment de l'arrivée de la connexion. Pour cette raison, la politique *deferred* offre une meilleure performance que la politique *deterministic*, et ce quelque soit la méthode utilisée.

L'effet de résonance. Dans les simulations stochastiques, nous avons observé un effet de résonance pour les attaques en rafale, quand une seule rafale remplit entièrement la file. La performance des stratégies à base de la politique *deferred* ainsi que la méthode *fixed* est diminuée dans ce cas spécifique. Cependant, l'effet de résonance n'est pas observé dans les expériences à cause de l'effet de filtre passe-bas du réseau.

Optimisation des paramètres. Seule la stratégie *linear deferred* présente des valeurs du délai d'inactivité à file vide (T_0) et à file pleine (T_1) qui sont optimales pour tous les scénarios. Ces valeurs, T_0 très grand ($\approx 2^{10}$) et T_1 très petit ($\approx 2^{-10}$), montrent la robustesse de cette stratégie par rapport aux variations du taux de trafic légitime et malicieux. Pour ces mêmes valeurs, la stratégie *threshold deferred* offre une performance non-optimale mais qui pourrait être un bon compromis par rapport à la méthode *fixed*.

Nous identifions dans la suite des limitations de notre approche et des directions de recherche pour des travaux futurs. Notre modèle théorique est construit sur la supposition que le trafic d'attaque suit la distribution d'un processus de Poisson. Un attaquant a intérêt à générer ce type de trafic pour ne pas se faire détecter par des mesures d'analyse statistique. De plus, l'architecture et le mode de fonctionnement de l'Internet transforment la distribution dans le temps du trafic IP reçu par une victime potentielle vers une distribution de processus de Poisson. Cependant, si l'attaque est réalisée sur d'autres protocoles de plus haut niveau et avec un taux d'arrivée plus petit, il peut être intéressant pour l'attaquant de générer du trafic dont la distribution est différente de celle d'un processus de Poisson. Pour cette raison nous considérons comme travail futur de recherche la modélisation mathématique d'autres distributions d'attaque.

L'approximation que nous avons faite pour calculer théoriquement la performance de la politique d'assignation du délai d'inactivité *deterministic* introduit des erreurs trop importantes et rend inutilisables les résultats théoriques. Nous visons à proposer des meilleures approximations pour cette politique dans des travaux futurs.

La politique d'assignation du délai d'inactivité *Poisson* que nous avons modélisée théoriquement offre des résultats théoriques très bons, similaires aux résultats obtenus par simulation stochastique est expérimentalement avec la politique *deferred*. Nous avons modélisé mathématiquement cette politique à cause de sa simplicité, mais il n'est pas évident quel serait l'algorithme correspondant à cette politique qui pourrait être implémenté dans une stratégie du délai d'inactivité dynamique dans la pratique. Comme travail futur, il serait intéressant de trouver un algorithme décrivant la politique *Poisson*, de l'implémenter et de tester l'impact de l'effet de résonance face à cette politique. Nous soupçonnons que cette politique serais plus robuste que la politique *deferred*, qui offre des performances similaires, à cause du fait que les connexions ne sont pas jetées instantanément après transitions de la file, comme c'est le cas avec la politique *deferred*. De plus, nous soupçonnons que la stratégie *linear deferred* configurée avec un délai d'inactivité à file vide très grand et avec un délai d'inactivité à file pleine très grand serait équivalente à une technique de gestion de la file comme une structure de données de type tampon circulaire (*ring buffer*). La confirmation de cette intuition permettrait de bâtir un modèle mathématique plus simple pour cette stratégie qui se traduirait par une estimation numérique de performance plus rapide et plus précise.

L'architecture que nous avons choisie pour la réalisation des mesures expérimentales des stratégies du délai d'inactivité dynamique contre une attaque de type *SYN-Flood* nous a limité à des vitesses d'attaque de l'ordre de 8 Mbps. Cette limitation est due à l'utilisation du même canal de communication pour le contrôle de la file du serveur, via l'envoie des paquets RST, que pour le transport des messages des clients.

Les solutions que nous proposons pour combler ce problème sont soit d'utiliser une connexion réseau dédiée au contrôle de la file des connexions du serveur, soit de déployer l'application de protection sur le serveur à protéger.

Finalement, nous avons considéré le cas où l'attaquant dépense toutes ses ressources pour attaquer le serveur à un seul niveau du modèle OSI. Si dans les expériences nous avons choisi de illustrer les stratégies de protections face à l'attaque *SYN-Flood* qui vise le protocole de communication au niveau transport, TCP, dans le modèle mathématique et les simulations stochastiques nous avons fait des abstractions qui permettent d'appliquer les résultats et conclusions à d'autres protocoles orientées connexion. De plus, nous montrons comment capturer le comportement d'une attaque d'inondation avec des requêtes HTTP ou encore d'une attaque de réservation des places pour un événement, en identifiant les événements qui génèrent des transitions dans le modèle abstrait. Notre intuition est que les attaques qui visent les protocoles de plus haut niveau sont plus avantageuses pour l'attaquant non seulement parce que cette approche est peu étudiée et que la plupart des méthodes de protection de plus bas niveau ne sont pas applicables mais aussi parce que le compromis entre les ressources de l'attaquant et du défenseur est plus favorable pour l'attaquant dans ce cas. Des travaux de recherche futurs pourront confirmer cette intuition et montrer le niveau d'attaque optimal pour l'attaquant ce qui permettrait à un défenseur de savoir comment et où concentrer les efforts.

Pour conclure, nous rappelons que les attaques de déni de service par épuisement des ressources sont un outil très puissant qui permet de bloquer complètement ou partiellement l'accès des clients légitimes à des ressources spécifiques sur Internet. Ce travail présente une modélisation des attaques d'épuisement des ressources, et plus important, des méthodes pratiques pour combattre ces types d'attaques. Pour les administrateurs des systèmes, ce travail permet de savoir comment configurer les mécanismes de protection déjà en place pour avoir une meilleure protection et comment dimensionner les serveurs pour garantir une certaine qualité de service. Pour les concepteurs des systèmes d'exploitation, des équipements réseau et des solutions de protection matérielles et logicielles, ce travail présente des stratégies de gestion du délai d'inactivité qui offrent une très bonne protection contre les attaques d'épuisement des ressources mais qui s'implémentent facilement, qui n'induisent pas beaucoup de temps de calcul supplémentaire et qui sont complémentaires avec d'autres techniques de défense.

Références

ADAIR, S. (2008). Gambling websites under attack. `http://www.shadowserver.org/wiki/pmwiki.php?n=Calendar.20080218`.

AL-DUWAIRI, B. et MANIMARAN, G. (2006). Intentional dropping : a novel scheme for SYN flooding mitigation. *INFOCOM 2006 : Proceedings of the 25th Annual Joint Conference of the IEEE Computer and Communications Societies.* IEEE Computer Society, 1–5.

AYRES, P. E., SUN, H., CHAO, H. J. et LAU, W. C. (2006). ALPi : A DDoS defense system for high-speed networks. *IEEE Journal on Selected Areas in Communications*, 24, 1864–1876.

BEAUMONT-GAY, M. (2007). A comparison of SYN flood detection algorithms. *ICIMP 2007 : Proceedings of the Second International Conference on Internet Measurement and Protection.* IEEE Computer Society.

BELLAÏCHE, M. et GRÉGOIRE, J.-C. (2007). SYN flooding attack detection by TCP handshake behaviour observation. *MonAM 2007 : Proceedings of the IEEE Workshop on Monitoring, Attack Detection and Mitigation.*

BERNSTEIN, D. J. (2003). SYN cookies. `http://cr.yp.to/syncookies.html`.

BOTEANU, D., FERNANDEZ, J. M., MCHUGH, J. et MULLINS, J. (2007a). Queue management as a DoS counter-measure ? *ISC 2007 : Proceedings of the Information Security Conference.* 263–280.

BOTEANU, D., FERNANDEZ, J. M. et MULLINS, J. (2006). On the efficiency of timeout-based DoS attack protections. Rapport technique, École Polytechnique de Montréal.

BOTEANU, D., REICH, E., FERNANDEZ, J. M. et MCHUGH, J. (2007b). Implementing and testing dynamic timeout adjustment as a DoS counter-measure. *QoP '07 : Proceedings of the ACM Workshop on Quality of Protection [held in conjunction with ACM Communications and Computer Security Conference (CCS)].* ACM, 34–39.

CAO, J., CLEVELAND, W. S., LIN, D. et SUN, D. X. (2001). On the nonstationarity of Internet traffic. *ACM SIGMETRICS Performance Evaluation Review*, 29, 102–112.

CHEN, S. et SONG, Q. (2005). Perimeter-based defense against high bandwidth DDoS attacks. *IEEE Transactions on Parallel and Distributed Systems*, 16, 526–537.

CHENG, C.-M., KUNG, H. et TAN, K.-S. (2002). Use of spectral analysis in defense against DoS attacks. *GLOBECOM '02 : Proceedinds of the Global Telecommunications Conference.* IEEE Computer Society, vol. 3, 2143–2148.

CHOUMAN, M., SAFA, H. et ARTAIL, H. (2005). Novel defense mechanism against SYN flooding attacks in IP networks. *Proceedings of the Canadian Conference on Electrical and Computer Engineering.*

DIERKS, T. et RESCORLA, E. (2006). RFC4346 : The transport layer security (TLS) protocol. Version 1.1. `http://www.ietf.org/rfc/rfc4346.txt`.

DIVAKARAN, D. M., MURTHY, H. A. et GONSALVES, T. A. (2006). Detection of SYN flooding attacks using linear prediction analysis. *ICON '06 : Proceedings of the 14th IEEE International Conference on Networks.* IEEE Computer Society, vol. 1.

DONG, K., YANG, S. et WANG, S. (2006). Analysis of low-rate TCP DoS attack against FAST TCP. *ISDA '06 : Proceedings of the Sixth International Conference on Intelligent Systems Design and Applications.* IEEE Computer Society.

DOULIGERIS, C. et MITROKOTSA, A. (2004). DDoS attacks and defense mechanisms : classification and state-of-the-art. *Computer Networks*, 44, 643–666.

FEINSTEIN, L., SCHNACKENBERG, D., BALUPARI, R. et KINDRED, D. (2003). Statistical approaches to DDoS attack detection and response. *DISCEX-III : Proceedings of the 3rd DARPA Information Survivability Conference and Exposition.* IEEE Computer Society, 303–314.

FENG, W. C., KAISER, E. et LUU, A. (2005). Design and implementation of network puzzles. *INFOCOM 2005 : Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies.* IEEE Computer Society.

FERGUSON, P. et SENIE, D. (1998). RFC2267 - Network Ingress Filtering : Defeating Denial of Service attacks which employ IP source address spoofing. `http://www.ietf.org/rfc/rfc2267.txt`.

FIELDING, R., GETTYS, J., MOGUL, J., FRYSTYK, H., MASINTER, L., LEACH, P. et BERNERS-LEE, T. (1999). RFC2616 : Hypertext Transfer Protocol – HTTP/1.1. `http://www.ietf.org/rfc/rfc2616.txt`.

GHAVIDEL, A. Z. et ISSAC, B. (2007). Secure transport protocols for DDoS attack resistant communication. *SCOReD 2007 : Proceedings of the 5th Student Conference on Research and Development.*

HANDLEY, M. et RESCORLA, E. (2006). RFC4732 - Internet Denial-of-Service considerations. `http://www.ietf.org/rfc/rfc4732.txt`.

IOANNIDIS, J. et BELLOVIN, S. M. (2002). Implementing Pushback : Router-based defense against DDoS attacks. *NDSS '02 : Proceedings of Network and Distributed System Security Symposium.* The Internet Society.

JAGERMAN, D. L. (1975). Nonstationary blocking in telephone traffic. *Bell System Technical Journal*, 54, 625–661.

JAGERMAN, D. L., MELAMED, B. et WILLINGER, W. (1997). Stochastic modeling of traffic processes. *Frontiers in queueing : models and applications in science and engineering.*

JIN, C., WANG, H. et SHIN, K. G. (2003). Hop-count filtering : an effective defense against spoofed DDoS traffic. *CCS '03 : Proceedings of the 10th ACM conference on Computer and Communications Security.* ACM, 30–41.

JUELS, A. et BRAINARD, J. (1999). Client puzzles : A cryptographic defense against connection depletion attacks. *NDSS '99 : Proceedings of the Network and Distributed System Security Symposium.* 151–165.

KEROMYTIS, A. D., MISRA, V. et RUBENSTEIN, D. (2004). SOS : an architecture for mitigating DDoS attacks. *IEEE Journal on Selected Areas in Communications*, 22, 176–188.

KHAN, S. et TRAORÉ, I. (2005). Queue-based analysis of DoS attacks. *Proceedings of the IEEE Workshop on Information Assurance and Security.* IEEE Computer Society, 266–273.

KIM, Y., LAU, W. C., CHUAH, M. C. et CHAO, H. J. (2006). PacketScore : A statistics-based packet filtering scheme against distributed Denial-of-Service attacks. *IEEE Transactions on Dependable and Secure Computing*, 03, 141–155.

KUZMANOVIC, A. et KNIGHTLY, E. W. (2003). Low-rate TCP-targeted Denial of Service attacks : the shrew vs. the mice and elephants. *SIGCOMM '03 : Proceedings of the Conference on Applications, technologies, architectures, and protocols for computer communications.*

LEMON, J. (2002). Resisting SYN flood DoS attacks with a SYN cache. *BSDC'02 : Proceedings of the BSD Conference.* USENIX Association.

LEYDEN, J. (2007). Consumer revenge site returns after DDoS attack. `http://www.theregister.co.uk/2007/10/29/moneysavingexpert_ddos/`.

LIM, B. et UDDIN, M. S. (2005). Statistical-based SYN-flooding detection using programmable network processor. *ICITA '05 : Proceedings of the Third International Conference on Information Technology and Applications.* IEEE Computer Society, 465–470.

LUI, J. C. S., MISRA, V. et RUBENSTEIN, D. (2004). On the robustness of soft state protocols. *ICNP '04 : Proceedings of the 12th IEEE International Conference on Network Protocols.* IEEE Computer Society, 50–60.

MACIÁ-FERNÁNDEZ, G., DíAZ-VERDEJO, J. E. et GARCíA-TEODORO, P. (2007). Evaluation of a low-rate DoS attack against iterative servers. *Computer Networks*, 51, 1013–1030.

MADAN, B. B., GOSEVA-POPSTOJANOVA, K., VAIDYANATHAN, K. et TRIVEDI, K. S. (2002). Modeling and quantification of security attributes of software systems. *DNS 2002 : Proceedings of the International Conference on Dependable Systems and Networks.* IEEE Computer Society, 505–514.

MASSEY, W. A. et WHITT, W. (1994). An analysis of the Modified Offered-Load approximation for the nonstationary Erlang loss mode. *The Annals of Applied Probability*, 4, 1145–1160.

MEADOWS, C. (1999). A formal framework and evaluation method for network Denial of Service. *CSFW '99 : Proceedings of the 12th IEEE Workshop on Computer Security Foundations.* IEEE Computer Society, 4.

MEADOWS, C. (2001). A cost-based framework for analysis of Denial of Service in networks. *Journal of Computer Security*, 9, 143–164.

MICROSOFT TECHNET (2003). Security considerations for network attacks. `http://www.microsoft.com/technet/security/topics/networksecurity/secdeny.mspx`.

MIRKOVIC, J., DIETRICH, S., DITTRICH, D. et REIHER, P. (2004). *Internet Denial of Service : Attack and defense mechanisms.* Prentice Hall PTR.

MIRKOVIC, J., HUSSAIN, A., WILSON, B., FAHMY, S., REIHER, P., THOMAS, R., YAO, W.-M. et SCHWAB, S. (2007a). Towards user-centric metrics for Denial-of-Service measurement. *ExpCS '07 : Proceedings of the Workshop on Experimental Computer Science.* ACM.

MIRKOVIC, J., PRIER, G. et REIHER, P. L. (2002). Attacking DDoS at the source. *ICNP '02 : Proceedings of the 10th IEEE International Conference on Network Protocols.* IEEE Computer Society, 312–321.

MIRKOVIC, J. et REIHER, P. (2004). A taxonomy of DDoS attack and DDoS defense mechanisms. *SIGCOMM Computer Communication Review*, 34, 39–53.

MIRKOVIC, J., REIHER, P., FAHMY, S., THOMAS, R., HUSSAIN, A., SCHWAB, S. et KO, C. (2006). Measuring Denial of Service. *QoP '06 : Proceedings of the 2nd ACM Workshop on Quality of Protection.* ACM, 53–58.

MIRKOVIC, J., ROBINSON, M. et REIHER, P. (2003). Alliance formation for DDoS defense. *NSPW '03 : Proceedings of the Workshop on New Security Paradigms.* ACM, 11–18.

MIRKOVIC, J., WEI, S., HUSSAIN, A., WILSON, B., THOMAS, R., SCHWAB, S., FAHMY, S., CHERTOV, R. et REIHER, P. (2007b). DDoS benchmarks and experimenter's workbench for the DETER testbed. *TRIDENTCOM 2007 : Proceedings of the 3rd International Conference on Testbeds and Research Infrastructure for the Development of Networks and Communities.* IEEE Computer Society.

MURPHY, K. (2007). Godaddy whacked by DDoS attack. `http://www.cbronline.com/article_news.asp?guid=D137B95B-B05A-4838-A584-CD1DD71DDE26`.

NAKASHIMA, T. et OSHIMA, S. (2006). A detective method for SYN flood attacks. *ICICIC '06 : Proceedings of the First International Conference on Innovative Computing, Information and Control.* IEEE Computer Society, 48–51.

NAKASHIMA, T. et SUEYOSHI, T. (2007). Performance estimation of TCP under SYN flood attacks. *CISIS 2007 : Proceedings of the First International Conference on Complex, Intelligent and Software Intensive Systems.* IEEE Computer Society, 92–99.

NATU, M. et MIRKOVIC, J. (2007). Fine-grained capabilities for flooding DDoS defense using client reputations. *LSAD 2007 : Proceedings of the ACM SIGCOMM Workshop on Large-Scale Attack and Defense.*

NAZARIO, J. (2007). Estonian DDoS attacks - a summary to date. `http://asert.arbornetworks.com/2007/05/estonian-ddos-attacks-a-summary-to-date`.

NOURELDIEN, N. A. et OSMAN, I. M. (2000). A stateful inspection module architecture. *Proceedings of IEEE TENCON 2000 Conference.* IEEE Computer Society, vol. 2, 259–265.

OHSITA, Y., ATA, S. et MURATA, M. (2005). Deployable overlay network for defense against distributed SYN flood attacks. *ICCCN 2005 : Proceedings of the 14th International Conference on Computer Communications and Networks.* IEEE Computer Society, 407–412.

OIKONOMOU, G., MIRKOVIC, J., REIHER, P. et ROBINSON, M. (2006). A framework for a collaborative DDoS defense. *ACSAC '06 : Proceedings of the 22nd Annual Computer Security Applications Conference on Annual Computer Security Applications Conference.* IEEE Computer Society, 33–42.

POSTEL, J. et REYNOLDS, J. (1985). RFC959 : File transfer protocol (FTP). `http://tools.ietf.org/html/rfc959`.

ROBINSON, M., MIRKOVIC, J., MICHEL, S., SCHNAIDER, M. et REIHER, P. (2003). DefCOM : defensive cooperative overlay mesh. *DISCEX-III : Proceedings of the 3rd DARPA Information Survivability Conference and Exposition.* IEEE Computer Society, vol. 2, 101–102.

SCHUBA, C. L., KRSUL, I. V., KUHN, M. G., SPAFFORD, E. H., SUNDARAM, A. et ZAMBONI, D. (1997). Analysis of a Denial of Service attack on TCP. *SP '97 : Proceedings of the 1997 IEEE Symposium on Security and Privacy.* IEEE Computer Society, 208.

SHAKKOTTAI, S., SRIKANT, R., BROWNLEE, N., BROIDO, A. et CLAFFY, K. (2004). The RTT distribution of TCP flows in the Internet and its impact on TCP-based flow control. Rapport technique, Cooperative Association for Internet Data Analysis (CAIDA).

SHEVTEKAR, A., ANANTHARAM, K. et ANSARI, N. (2005). Low rate TCP Denial-of-Service attack detection at edge routers. *IEEE Communications Letters*, 9, 363–365.

SHIN, S., KIM, K. et JANG, J. (2005). D-SAT : detecting SYN flooding attack by two-stage statistical approach. *Proceedings of the Symposium on Applications and the Internet.*

SIRIS, V. A. et PAPAGALOU, F. (2004). Application of anomaly detection algorithms for detecting SYN flooding attacks. *GLOBECOMM '04 : Proceedings of the Global Telecommunications Conference.* IEEE Computer Society.

SMITH, B. (2008). A storm (worm) is brewing. *Computer*, 41, 20–22.

SPAMNATION (2007). 419eater DDoS'd? `http://www.spamnation.info/blog/archives/2007/09/419eater_ddosd.html`.

TARTAKOVSKY, A. G., ROZOVSKII, B. L., BLAZZEK, R. B. et KIM, H. (2006). A novel approach to detection of intrusions in computer networks via adaptive sequential and batch-sequential change-point detection methods. *IEEE Transactions on Signal Processing*, 54, 3372–3382.

VARANASI, R., PHOHA, V. V. et JOSHI, S. (2004). IP-traceback based attacker tracking : A probabilistic technique for detecting Internet attacks using the concept of Hidden Markov Models. *Proceedings of the 5th IEEE Information Assurance Workshop, US Military Academy of West Point.* IEEE Computer Society.

WANG, H., ZHANG, D. et SHIN, K. G. (2002). Detecting SYN flooding attack. *NFOCOM 2002 : Proceedings of the Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies.* IEEE Computer Society, 1530–1539.

XIAO, B., CHEN, W., HE, Y. et SHA, E. H. M. (2005). An active detecting method against SYN flooding attack. *ICPADS '05 : Proceedings of the 11th International Conference on Parallel and Distributed Systems.* IEEE Computer Society, 709–715.

XU, J. et LEE, W. (2003). Sustaining availability of web services under Distributed Denial of Service attacks. *IEEE Transactions on Computers*, 52, 195–208.

YANG, G., GERLA, M. et SANADIDI, M. Y. (2004). Defense against low-rate TCP-targeted Denial-of-Service attacks. *ISCC '04 : Proceedings of the Ninth International Symposium on Computers and Communications.* IEEE Computer Society, 345–350.

YANG, X., WETHERALL, D. et ANDERSON, T. (2005). A DoS-limiting network architecture. *ACM SIGCOMM Computer Communication Review*, 35, 241–252.

YAU, D. K. Y., LUI, J. C. S., LIANG, F. et YAM, Y. (2005). Defending against distributed Denial-of-Service attacks with max-min fair server-centric router throttles. *IEEE/ACM Transactions on Networking*, 13, 29–42.

ZHANG, S. et DASGUPTA, P. (2003). Denying Denial-of-Service attacks : A router based solution. *ICDCS 2003 : Proceedings of the 23 IEEE International Conference on Distributed Computing Systems.* IEEE Computer Society.

ZOU, C. C., DUFFIELD, N., TOWSLEY, D. et GONG, W. (2006). Adaptive defense against various network attacks. *IEEE Journal on Selected Areas in Communications*, 24, 1877–1888.

ZUQUETE, A. (2002). Improving the functionality of SYN cookies. *Proceedings of the IFIP TC6/TC11 Sixth Joint Working Conference on Communications and Multimedia Security.* Kluwer Academic Publishers, 57–77.

ANNEXE A
Queue Management as a DoS counter-measure ?

[PAGE BLANCHE INTENTIONNELLE]

Queue Management as a DoS counter-measure?

Daniel Boteanu[1], José M. Fernandez[1], John McHugh[2], John Mullins[1]

[1] École Polytechnique de Montréal
[2] Dalhousie University

Abstract. In this paper, we study the performance of timeout-based queue management practices in the context of flood denial-of-service (DoS) attacks on connection-oriented protocols, where server resources are depleted by uncompleted illegitimate requests generated by the attacker. This includes both crippling DoS attacks where services become unavailable and Quality of Service (QoS) degradation attacks. While these queue management strategies were not initially designed for DoS attack protection purposes, they do have the desirable side-effect or providing some protection against them, since illegitimate requests time out more often than legitimate ones. While this fact is intuitive and well-known, very few quantitative results have been published on the potential impact on DoS-attack resilience of various queue management strategies and the associated configuration parameters. We report on the relative performance of various queue strategies under a varying range of attack rates and parameter configurations. We hope that such results will provide usable configuration guidelines for end-server or network appliance queue hardening. The use of such optimisation techniques is complementary to the upstream deployment of other types of DoS-protection countermeasures, and will probably prove most useful in scenarios where some residual attack traffic still bypasses them.
Keywords: Denial of service attack, degradation of service attack, queue management, timeout, dynamic timeout.

1 Introduction

A denial-of-service (DoS) network attack occurs when the victim receives a malicious stream of packets that prevent the legitimate communication from taking place. DoS flood attacks consist in sending the victim (typically a server) a higher volume of traffic than it can handle. This can be achieved either by saturating the server's network connection or by using weaknesses in the communication protocols that typically allow the attacker to generate high server resource usage for a limited attacker effort. Distributed denial-of-service (DDoS) attacks are simply DoS attacks performed by multiple agents, most frequently simultaneously. In this paper we direct our attention towards the resource exhaustion attacks on connection oriented protocols. Although the studied-to-death SYN-flood attack fits well into this category, we use it merely as an example to explain our approach. As we will discuss, it is our hope that our approach could potentially be applied to other TCP-based attacks (e.g. ACK flood) or higher-level attacks against Web servers (straight HTTP or via SSL), FTP servers, VPN gateways, mail servers, or even DoS protection mechanisms in upstream network appliances.

The impact of a DoS attack on a particular system will vary depending on the protocols and applications involved. Furthermore, an attack can have measurable impact on the Quality of Service (QoS) of a system even when the server resources are not completely exhausted [21], such as in the case of *Degradation of Service* attacks [19]. While degrading QoS or even rendering a service unavailable might be possible, this always comes at a cost for the attacker. For a given service level or attack impact, there is a direct relationship between the resources expended by the attacker and the target. These tradeoffs have been discussed for crippling DoS attacks [16, 17], but these formalisms cannot be easily applied to QoS degradation attacks. While some experimental testbeds have been proposed to try to measure these tradeoffs [2], there are in fact very few quantitative results (modelling or experimental) concerning degradation of service attacks.

Various methods and appliances for protecting against DoS attacks have been suggested, for example Cisco Guard XT, Captus IPS, COSSACK, DefCOM, D-WARD, MANAnet Shield, Mazu Enforcer, NetBouncer, Peakflow, Proof of Work, Pushback, Secure Overlay Services, Traceback and others (see [19, 20] for complete surveys on the topic). These can be viewed as first- and second-line defences, where first-line defences use traffic profiling or anomaly detection mechanisms and filter it accordingly [8, 25, 22], and second-line defences consist in modifying TCP/IP protocols to positively affect the resource tradeoff in favour of the defender [3, 28, 10, 12, 7].

Nonetheless, it is possible for sophisticated attacks to evade both types of defences. Thus, a considerable amount of residual attack traffic could still evade both network- and host-based defences and reach the end server OS and application connection queues. When all traffic-discriminating counter-measures have been bypassed, legitimate and residual attack traffic is indistinguishable. Fortunately, certain features of the end server can mitigate the impact of residual traffic, even in those conditions. These features therefore constitute a last line of defence. Queue management algorithms that were initially designed to minimise the impact of network traffic loss or high latency fall into that category. One of the most common such features is the attribution of timeout periods to all incoming connections. Protocols that implement this feature and that do not necessarily require explicit messages to close a connection are called *soft protocols* and they perform better than their counterpart *hard protocols* in unexpected network conditions like DoS attacks [14]. In practice, the timeouts can be adjusted dynamically according to administrator-configurable thresholds on resource usage levels, as has been suggested and implemented in network appliances [10] and OS [18] for the specific purpose of improving resiliency against DoS attacks. When these thresholds are reached, the server is placed in a "protective" state, which in principle has the effect of favouring fast legitimate connections over attack connections.

Unfortunately, it is not clear at all what the "optimal" threshold values are, as there is no quantitative method for estimating the parameters that minimise the effects of DoS attacks while maintaining equivalent levels of QoS. In the rest of this paper, we try to address this gap. One of the reasons we are interested in this is because such features are very general and already in place at the various network and application layers. Note also that maximising their effectiveness as DoS protections is complementary and in principle compatible with the deployment and use of other upstream defences.

In the next section we propose a stochastic modelling tool for DoS attacks, based on Markov chains. Using this model, we analyse three different timeout-based protection strategies in Sect. 3. We provide in Sect. 4 the experimental results obtained by simulating these strategies according to two different implementations. Finally, we conclude and give directions for future work in Sect. 5.

2 Modelling Servers under DoS Attack with Markov Chains

Markov chains are a stochastic modelling tool that describe the states and dynamics of a system at successive times. They are said to be *memoryless* if the probability of transition between any two states is independent of the previous states. The stochastic process generating state transition events is thus said to be *markovian*, which is equivalent to saying that they are distributed in time according to a Poisson distribution. Markov chains can also be used to model systems in which this is not the case, i.e. those where state transitions probabilities will depend on past history. In instances where the key parameters such as rate of arrivals and departures are known, the model can be "solved". First, this means that given state probabilities at given time, predictions can be made about state probabilities at a later time. We can also compute *steady-state probabilities*, which correspond to the likelihood of the various states at the equilibrium of the system.

Markov chains are suitable for modelling network performance and have been used in that purpose for many years. In particular, Markov Chains have also been used as modelling tool in network security. Baras [1] suggests detecting route falsification attacks in mobile ad-hoc networks (MANET) using a Hidden Markov Model (HMM). More recent studies [27] show that using edge sampling techniques along with HMM can be used to reconstruct a network attack path. HMMs can also be used in Intrusion Detection Systems [11, 15], the transitions between each state in the Markov model being generated by intrusion, detection and recovery events. Finally, Khan *et al.*[13] have successfully used Markov Chain modelling of queues to design DoS traffic detection strategies. In our case, we will use Markov chains to model the performance of servers under DoS attacks.

2.1 Description of the model

Typical Markov chain models used in network performance have each state characterised by the number of connections in the system. A maximum number of connections c can be served at the same time. Connections arrive with rate λ and are served with rate μ. In our case, each state in the chain is characterised by two values: N_l and N_m, the number of connections used by legitimate users and malicious users, respectively. A maximum number of both legitimate and malicious connection c can be served in the same time. All connection requests that arrive when the server is in a saturated state ($N_l + N_m = c$) will be rejected. Transitions between states occur with different rates for the legitimate and malicious connection requests: λ_l and μ_l for the arrivals and servings of legitimate connections and λ_m and μ_m for the arrivals and servings of malicious connections. The chain has a triangular form where states on the upper line represent that no malicious connections are present in the system and states on the diagonal represent that only malicious connections are present in the system (see Fig. 6 in the appendix). The following events generate transitions between states:

- *connection arrived*: the server received a connection request from a client. It occurs at a rate λ;
- *connection completed*: the connection was either elevated to a higher-level protocol, or the client was served with the required information and the connection was closed successfully. It occurs at a rate μ_l;
- *connection rejected*: the server was not able to serve the connection because no more connections channels were available (queue full). It occurs at a rate ϕ_r;
- *connection expired*: the server tried to serve the connection but the communication timed out and the connection was dropped. It occurs at a rate ϕ_e; and
- *connection failed*: the connection was either rejected or it expired. It occurs at a rate $\phi = \phi_r + \phi_e$.

In the particular case of a SYN-flood attack, N_l will actually represent the number of legitimate connections (and N_m the number of malicious ones) that are half-open. The *connection arrived* and *connection completed* events represent a SYN message and the corresponding ACK message being received by the server, respectively. In the case of an SSL connection depletion attack, N_l and N_m represent the number of legitimate and malicious completed TCP connections, respectively, that have not yet established a secure channel and for which the negotiation phase is still in progress. The *connection arrived* events represent the *Client hello* message being received by the server and the *connection completed* events represent the corresponding *Finished* message being sent by the server. It is even possible to consider a nested model, each level representing a different layer in the protocol stack.

How realistic is this model regarding legitimate arrival and service rates? It is known that user sessions initiations resemble phone calls [23] and thus have a Poisson arrival process with exponential inter-arrival times. We will make the supposition that all incoming connections follow this pattern, assumption that is used in other DoS related research [21, 5]. In most cases, serving rates depend only of network transit times but in some cases user interaction is also a factor. It has been shown that because of the network queueing algorithms, all the IP packet traffic tends toward a Poisson process as the load increases [4]. For modelling purposes, we will make the supposition that the network is heavily used and therefore that legitimate service rate follows a Poisson process model. According to our assumption, *connections completed* events are generated at exponential intervals of time from the *connection arrived* events, if the timeout has not elapsed. Otherwise, *connection expired* events are generated at timeout intervals from the *connection arrived* events.

The rate at which *connection completed* messages are generated by the legitimate clients is μ_c. Only messages that arrive to the server before their timeout elapses will generate *connection completed* events; we thus have that $\mu_l \geq \mu_c$. All the other will be ignored by the server and *connection expired* events are generated when the timeout elapses. Therefore, the Probability Distribution Function (PDF) of the legitimate connection service time $G_l(t)$ will have the form of an exponential distribution for t smaller than the timeout t_{out}, followed by an appropriately weighted delta Dirac function at t_{out}

$$G_l(t) = \begin{cases} \mu_c e^{-t\mu_c} & t < t_{\text{out}} \\ \delta(t - t_{\text{out}}) p_{\text{expire}} & t = t_{\text{out}} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

where

$$p_{\text{expire}} = \int_{t_{\text{out}}}^{\infty} \mu_c e^{-t\mu_c} dt = e^{-t_{\text{out}}\mu_c} \ . \tag{2}$$

The mean service time and the service rate for legitimate connections are

$$t_l \triangleq \int_0^{\infty} t G_l(t) dt = \frac{1 - e^{-t_{\text{out}}\mu_c}}{\mu_c} \ ; \qquad \mu_l \triangleq \frac{1}{t_l} = \frac{\mu_c}{1 - e^{-t_{\text{out}}\mu_c}} \tag{3}$$

While we model legitimate packet arrivals as a Poisson process this is not general for attack traffic as the attacker is free to use whatever strategy he or she wants. Even though there is no proof that this is optimal, the attacker might want to mimic the legitimate arrivals process in order to thwart certain time analysis detection methods. In any case, we will assume that the *residual attack traffic*, unfiltered by upstream defence mechanisms is distributed according to a Poisson distribution, because otherwise it could have been potentially discriminated by such techniques. We make this assumption in order to be able to construct a simple enough mathematical model that we can numerically resolve. However, we will later explore in Sect. 4 attacks for which this is not true.

Concerning the malicious packet service process, the strategy of the attacker is to exhaust the server resources using the smallest effort possible. This is achieved by generating the *connection arrived* events and then abandoning the communication without any notice to the server. Malicious connections will eventually all expire and generate *connection expired* events at t_{out} intervals of time from the *connection arrived* events. The malicious connection service rate is in this case $\mu_m = 1/t_{\text{out}}$.

Although the triangular DoS Markov chain model that we presented describes the states in which the server will be during an attack, these states are not directly visible because individual connections can not be labelled as legitimate or malicious. For this reason, we will analyse the *visible* Markov chain that has $c+1$ states, each state being characterised by the number of connections N used by both legitimate and malicious users. The probability that the visible Markov chain is in a state N is the sum of all probabilities that the hidden Markov chain is in state (N_l, N_m) with $N_l + N_m = N$.

The visible connection arrival process is the sum of two Poisson processes with rates λ_l and λ_m and thus also a Poisson process with rate $\lambda = \lambda_l + \lambda_m$. In a Markov chain model the *load* is defined as the ratio between the arrival and service rates. In our case, we distinguish the load generated by the legitimate users $\rho_l = \lambda_l/\mu_l$, and the load generated by malicious users $\rho_m = \lambda_m/\mu_m$. The overall load ρ cannot be computed directly because the service processes are not memoryless. Our goal is to compute the overall load by approximating the overall mean service time $\tilde{t}$. We consider $\tilde{t}$ to be constant in time and equal to the average of the mean legitimate service time t_l and mean malicious service time t_m weighted by the legitimate load and the malicious load, respectively:

$$\tilde{t} = \frac{\rho_l}{\rho_l + \rho_m} t_l + \frac{\rho_m}{\rho_l + \rho_m} t_{\text{out}} \tag{4}$$

The approximative mean service rate in the visible chain is:

$$\tilde{\mu} \triangleq \frac{1}{\tilde{t}} = \frac{\mu_l \mu_m (\lambda_m \mu_l + \lambda_l \mu_m)}{\lambda_m \mu_l^2 + \lambda_l \mu_m^2} \tag{5}$$

We can now calculate an approximative overall load generated by both legitimate and malicious users as $\tilde{\rho} \triangleq \lambda/\tilde{\mu}$. With this approximation we can compute the steady-state probability that the system is in the state k using Erlang's loss formula:

$$p_k = \frac{\tilde{\rho}^k}{k!} \Big/ \sum_{i=0}^{c} \frac{\tilde{\rho}^i}{i!} \tag{6}$$

2.2 Approximate solutions to the model

Because the connections are served independently, the only significant performance measure is the probability ϕ that a legitimate connection will fail, which is equal to the probability that the connection will be rejected ϕ_r plus the probability the connection will expire ϕ_e, i.e. $\phi = \phi_r + \phi_e$.

The *blocking probability* is by definition the probability that the system is saturated, i.e. that the queue is full. A connection is rejected if the server is saturated when the *connection arrived* event is generated. The probability that a connection is rejected ϕ_r is thus equal to the probability that the server is in state c at that moment. If the system were at equilibrium, this will be exactly the steady-state probability p_c. If we assume that the system will never be far from equilibrium, we can approximate it as such, i.e. $\phi_r \approx p_c$.

A connection expires with the probability p_{expire} if the server is not saturated when the *connection arrived* event is generated. The connection expire probability can also be approximated with the steady-state probabilities as follows:

$$\phi_e \approx \sum_{k=0}^{c-1} p_k p_{\text{expire}} \tag{7}$$

In this model, the resources that the attacker spends to achieve a negative impact on the service level are proportional to the residual malicious connections arrival rate λ_m; the actual malicious traffic arrival rate at the upstream defences might be significantly higher. The resources that the server spends to achieve the required service level is represented by the capacity c of the queue. We are interested in how the tradeoff between the attacker and server resources varies for the same legitimate connection fail probability ϕ, or equivalently for the same *connection complete probability* $1-\phi$. Even though the fully expanded expression of ϕ is quite complex, what lies beneath it is a tradeoff between these quantities that is essentially linear for the same connection complete probability, as we have verified with several numerical calculations. Fig. 1(a) illustrates the contour curves for the connection complete probability, for different values of attack rates and server capacities. Note that they are essentially straight, indicating that an increase in attack rate by the attacker can be efficiently matched by a corresponding linear increase in queue capacity by the defender, while keeping the same quality of service; this confirms previously known intuition by experts in the network security field.

Although the residual traffic rates represented might seem ridiculously small, this traffic would have already been severely filtered by other upstream defences, if such were present. Thus, in order to get this small amount of residual traffic through, the attacker might have had to generate large amounts of traffic at the perimeter, resulting in a high resource cost. See Table 1 in the Appendix for default configuration parameters of different implementations of connection-oriented protocols.

Given a certain attack rate λ_m and server capacity c, the parameter that can be optimised by the defender is the timeout. As Fig. 1(b) shows, the two components of the legitimate connections reject probability ϕ, ϕ_e and ϕ_r, change in opposite directions as we vary the timeout: ϕ_e decreases exponentially with the timeout, while ϕ_r increases. When no attack is present ϕ_r is null for $\lambda_l < \mu_l c$; it has the limit $\lambda_l - \mu_l c$ for infinite timeout when $\lambda_l > \mu_l c$. When an attack is present, ϕ_r has the limit λ_l when timeout is infinite. For a specific attack rate and capacity there is an optimal timeout value that can be calculated numerically.

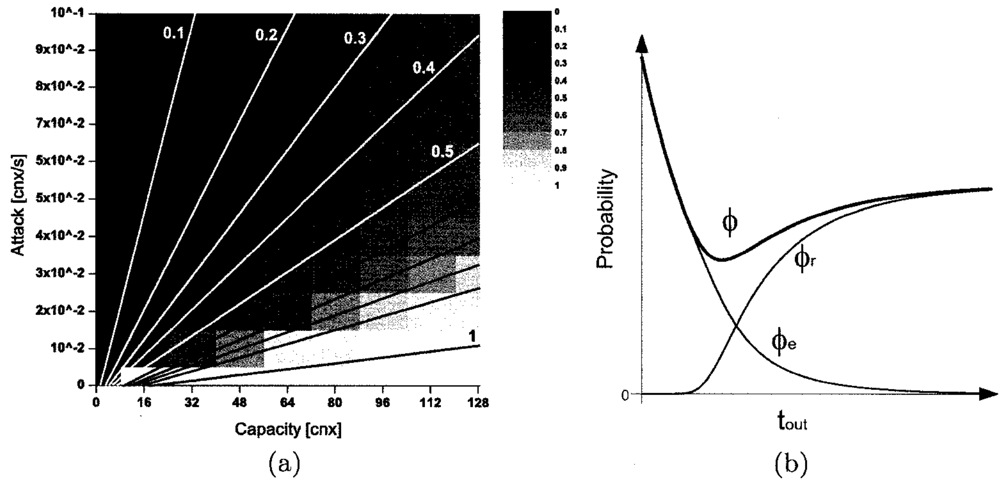

Fig. 1. (a) Steady-state legitimate connection complete probabilities for various queue capacities c (x-axis) and attack rates λ_m (y-axis), at fixed legitimate arrival rate $\lambda_l = 10$ cnx/s, mean service time $t_l = 1$ s, and timeout $t_{\text{out}} = 75$ s. For each pair (x, y), the corresponding connection complete probability is indicated as a gray-scale value for the corresponding rectangular region of the graph. Better quality of service (i.e. higher probability, lighter shades) are achieved with bigger queues and lower attack rates. The contour curves connect points (x, y) with the same connection complete probabilities (same colour), and are approximately represented by straight lines in the figure. (b) Variation of the steady-state reject, expire and fail probabilities, ϕ_r, ϕ_e, and ϕ, respectively, as a function of the timeout value t_{out}.

3 Dynamic Timeout Management Strategies

We will now analyse two queue management strategies that consist in dynamically adjusting the timeout. This is, of course, in contrast with the standard strategy of having a fixed, non-adaptive connection timeout value. Ideally we would want to make this adjustment by looking at the triangular Markov chain and choosing a timeout according to the number of legitimate and malicious connections in the server. Unfortunately, this model is not visible because the server is unable to distinguish if a connection request is legitimate or malicious. Therefore, the only information available to adjust the timeout is the total number of connections used. While the threshold prevention strategy is already implemented in Microsoft Windows Server 2003 and some security appliances, the second strategy, linear timeout prevention, is a concept that we introduce. Fig. 7 in the appendix illustrates how they fit in the taxonomy of DDoS defence of [20].

There are for each of these strategies, two alternate methods for deciding how to flush out timed out connections: *deterministic* and *deferred.* The *deterministic method* consists in tagging each connection with a pre-determined expiry time upon its arrival. The expiry time is simply the arrival time plus the timeout value at the moment of arrival. To take into account the fact that the reality of the system might have changed drastically since the arrival of a connection, another approach seems more suitable: to *defer* the assignment of an expiry time, such that if the timeout decreases after its arrival, the connection is checked against the new timeout value. Thus at any given time, connections are flushed if the time elapsed since their arrival is bigger than the current timeout value. We refer to this method as the *deferred method.* In the rest of this section, we instantiate the general Markov models of Sect. 2 and compute steady-state probabilities for the deterministic method only. We will nonetheless present simulation results for both in Sect. 4.

3.1 Threshold-based timeout adjustment strategy

This consists in using a normal, long timeout t_0 at first. If the number of connections used in the server is greater then a certain threshold S, a shorter, attack timeout t_1 will be used. The timeout used will depend at all times on the state k in which the server is:

$$t_{\text{out}}^{(k)} = \begin{cases} t_0 & k < S \\ t_1 & \textit{otherwise} \end{cases} \tag{8}$$

The probability that an individual connection will expire p_{expire}, the legitimate service rate μ_l and the approximative overall service rate $\tilde{\mu}$ described in (2), (3) and (5) all become state dependent:

$$p_{\text{expire}}^{(k)} = e^{-t_{\text{out}}^{(k)}\mu_c}\ ; \qquad \mu_l^{(k)} = \frac{\mu_c}{1 - e^{-t_{\text{out}}^{(k)}\mu_c}}\ ; \qquad \tilde{\mu}^{(k)} = \frac{\mu_l^{(k)}\mu_m(\lambda_m\mu_l^{(k)} + \lambda_l\mu_m)}{\lambda_m(\mu_l^{(k)})^2 + \lambda_l\mu_m^2} \tag{9}$$

We use the same principle as before to calculate the probability that the server is in a specific state k using Erlang's loss formula:

$$p_k = \frac{1}{k!}\prod_{j=0}^{k-1}\frac{\lambda_l + \lambda_m}{\tilde{\mu}^{(j)}} \Bigg/ \sum_{i=0}^{c}\left(\frac{1}{i!}\prod_{j=0}^{i-1}\frac{\lambda_l + \lambda_m}{\tilde{\mu}^{(j)}}\right) \tag{10}$$

Similar to the case where no timeout adjustment is made, the significant performance measure ϕ representing the legitimate *connection fail* event probability is calculated as:

$$\phi = \phi_r + \phi_e = p_c + \sum_{k=0}^{c-1} p_k p_{\text{expire}}^{(k)} \tag{11}$$

The tradeoff between the attacker and server resources is still linear but more favourable for the server than with a fixed timeout. Fig. 2 illustrates this tradeoff for numerical values of the rates (λ_l and μ_l), timeouts (t_0 and t_1) and threshold S similar to what we can find in Microsoft and McAfee products that use this strategy in a real-life scenario.

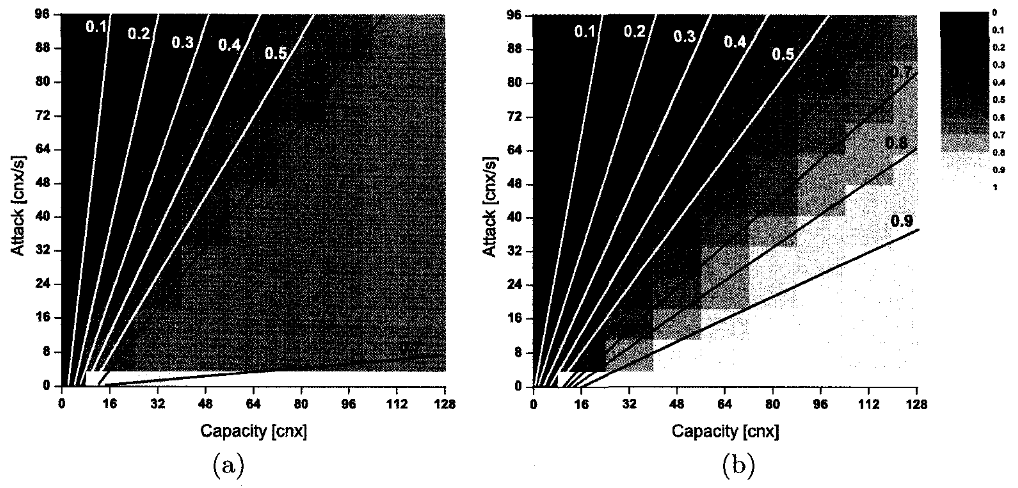

Fig. 2. Steady-state legitimate connection complete probabilities for various queue capacities c (x-axis) and attack rates λ_m (y-axis), at fixed legitimate arrival rate $\lambda_l = 10$ cnx/s, mean service time $t_l = 1$ s, and timeout values $t_0 = 75$ s, $t_1 = 1$ s for (a) a single threshold at $S = c/2$, and (b) linear adjustment. Connection complete probabilities for each combination (x, y) of queue size and attack rate is represented by gray-scaling the corresponding rectangular region.

3.2 Linear timeout adjustment strategy

This strategy differs from the threshold-based one in the way the timeout is decreased. Instead of suddenly decreasing the timeout when the server state reaches a certain threshold, this strategy gradually decreases the timeout as the number of connections in the server increases. When no connection is used (i.e. the server is in the state 0) an empty-queue long timeout t_0 is used; when all connections are used (i.e. the server is in the state c), a full-queue shorter timeout t_1 is used; and otherwise, a linear interpolation of the two values is used in all other server states. Thus, (8) becomes $t_{\text{out}}^{(k)} = t_0 + (t_1 - t_0)k/c$.

The same definitions in (9) that describe the individual *connection expire* event probability $p_{\text{expire}}^{(k)}$, the legitimate service rate $\mu_l^{(k)}$ and the approximative overall service rate $\tilde{\mu}^{(k)}$ can be inserted in the Erlang loss formula (10) to calculate the legitimate *connection fail* event probability:

$$\phi = \phi_r + \phi_e = p_c + \sum_{k=0}^{c-1} p_k p_{\text{expire}}^{(k)} \tag{12}$$

Once again, we are interested in the tradeoff between the attacker and server resources. Analysis of the two protection strategies show that for the same values of systems parameters and traffic (within the range explored), the linear timeout protection strategy could perform better than the threshold timeout protection strategy. These results are illustrated in Fig. 2(b). Finally, it is important to note that while the linear timeout adjustment strategy is slightly more complex than the threshold-based one, the computational overhead for a server implementing it is negligible.

4 Experimental Results and Interpretation

We implemented these two strategies, in both their deterministic and deferred variants, and measured their performance using a home-made traffic simulator. We also implemented and measured the performance of the standard fixed-timeout strategy, for comparison. The legitimate and residual malicious *connection requests* were generated using Poisson processes. The *connection complete* events for legitimate connections were also generated using a Poisson process. The residual attack traffic was generated in two different ways: a) a Poisson process, in order to validate the theoretical model, and b) a deterministic process with bursts of instantaneous traffic at regular time intervals; the volume of each burst adjusted such that the averaged traffic rate would always remain the same.

In the first case, we conducted simulations with parameters equivalent to those of a hardened Web server under attack. The queue capacity was set to a more realistic 8000 cnx and shorter timeout values were used: $t_{\mathrm{out}} = 10$ s, $t_0 = 10$ s, $t_1 = 0.2$ s. The range of attack rates explored went from a modest 128 cnx/s to a very respectable 65536 cnx/s, equivalent to a 26 Mbps (!) residual attack bandwidth. For all strategies, nine different input data sets were used (except for the linear deferred, where only one simulation was run). The averaged results are shown in Fig. 3 and they give a clear picture of the relative performance of the various methods we have discussed here; the maximum standard deviation for performance in all runs was 0.023.

In order to better understand these results, it is useful to define the notion of *relative attack virulence* as the ratio between the rate of attack λ_m and the queue size c. Intuitively, it corresponds to how many queues per second the attack could fill up, if there was no timeout and no legitimate traffic. In fact, our first observation is that virulence is indeed the most important parameter affecting completion probabilities. We have confirmed this by running simulations at various combinations of attack rate and queue size, and have observed the same linearity between them as we have described in Sect. 3 for the theoretical model (see Fig. 4 in the appendix for more details).

As can be seen, at low virulence (< 0.05 s^{-1}) the QoS degradation is negligible, and at very high virulence (> 16 s^{-1}) the degradation is equally unacceptable for all strategies. In between these values, which constitutes the "window of interest" of these results, several conclusions can be drawn with respect to the relative performance of these strategies that confirm the theoretical predictions of Sect. 3. First, both timeout adjustment strategies are much better than those with a fixed timeout. Second, linear adjustment performs slightly better than the threshold-based timeout adjustment. In particular, the differences in performance can be as high as 20%, for virulence around 2 s. This corresponds to a relatively high residual attack rate of 16,000 cnx/s (6.5 Mbps) at which all strategies would notice a significant decrease in QoS (at least 30% legitimate con-

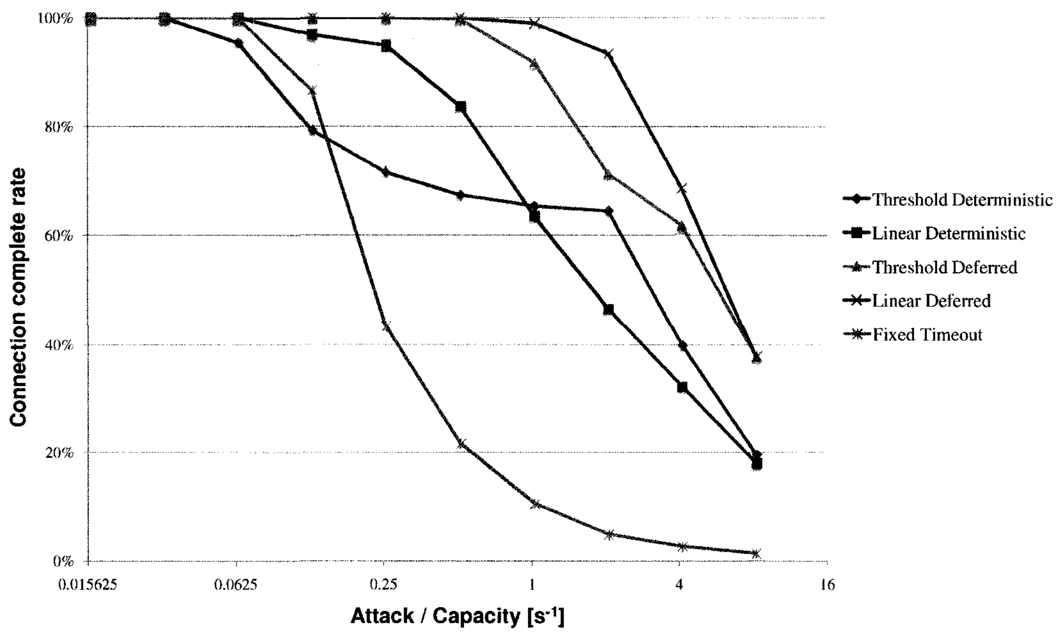

Fig. 3. Legitimate connection complete rate (y-axis) for various strategies, with fixed queue size $c = 8000$, legitimate traffic rate λ_l=100 cnx/s, mean service time $t_l = 0.2$ s, and timeout values t_0=10 s and t_1=0.2 s, for various relative virulence (x-axis).

nections lost), except the linear deferred strategy where QoS degradation would be very small (a few percent). Finally, let us emphasise that these conclusions are quite general. We ran a separate set of simulations with values typical of an unprotected TCP stack in an unhardened OS. For the same relative virulence, the QoS degradation results obtained are very similar, hence re-confirming the relative performance of the various strategies.

In the second case, we explored the performance of these strategies against attack traffic not generated according to a Poisson process, something we could not do with our theoretical model. The results of these simulations are shown in Fig. 4, where we show the performance of the strategies for a fixed attack rate and various burst inter-arrival times. First, we notice that a Poisson attack strategy is not always optimal for the attacker, as a significant degradation of QoS happens at an inter-arrival rate of 2 s (identified with a vertical line in Fig. 4). This value is particularly significant as at this virulence level the queue is completely filled with attack traffic at every burst, and the only time that legitimate traffic can be serviced is after some of these packets have timed out and before the next burst. This is akin to a “resonance effect” where the attack characteristics are matched to those of the queue. This is optimal to the attacker, first because higher inter-arrival times results in bursts that are oversized and waste attack packets, and in addition result in an increased time window in which legitimate packets can be serviced. Consequently, QoS levels re-establish

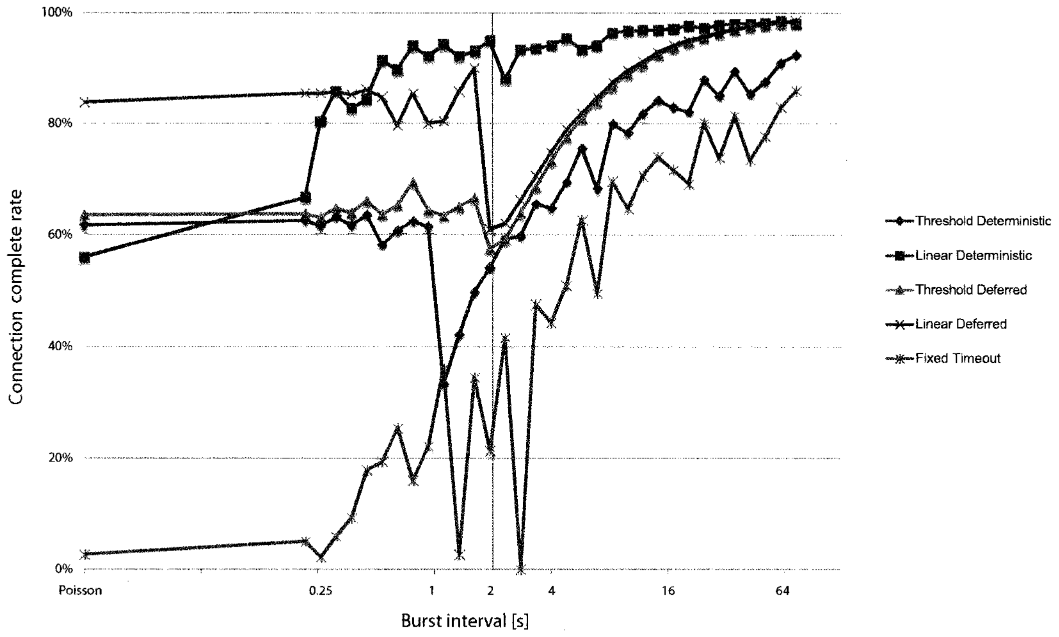

Fig. 4. Legitimate connection complete rate (y-axis) for various strategies, with fixed queue size $c = 128$ cnx, legitimate traffic rate λ_l=10 cnx/s, mean service time $t_l = 1$ s, timeout values t_0=75 s and t_1=1 s, and attack rate λ_m = 64 cnx/s, for Poisson attacks (far left) and various burst inter-arrival times (x-axis).

themselves linearly with respect to inter-arrival times. Second, if inter-arrival time is decreased, burst volume also decreases thus leaving space in the queue for legitimate requests arriving before the next burst to be serviced.

Nonetheless, the relative performance of the queue management strategies is the same as in the Poisson attack case. The only notable deviation is that the linear deterministic adjustment strategy is more robust to the queue resonance effect described above. Its performance is better than the linear deferred method (and all others) at all inter-arrival time settings, except for low-volume, frequent bursts.

5 Conclusions and Future Work

In this paper we made an effort to understand the effectiveness of queue management strategies against DoS attacks. We first constructed a Markov model describing the behaviour of a server under DoS attack that tries to exhaust the available connection slots in the queue. This model has allowed us to gain intuition on the likely tradeoffs between the various parameters that characterise a system under attack (traffic and service rates, queue size, etc.). Of particular interest, but relatively unexplored, is the possibility of optimising queue management parameters such as timeout and queue capacity with the respect to an expected residual attack rate and QoS requirement. There are however a few limitations to this model that should be the object of further research. First, we have used the steady-state approximations, thus assuming equilibrium, which is not accurate in the case of high residual traffic rates. Second, we have not described in this paper the model for analysing the deferred method of policing timeout connections out of the queue.

Nonetheless, from the analysis of the model in combination with the simulation results (which include non-Poisson residual traffic distributions), several interesting conclusions can be drawn that should be of immediate application for those vendors and system administrators that are incorporating or using such types of strategies in OS and applications in host servers or in anti-DoS network appliances:

1. The tradeoff between residual attack rate and queue capacity is indeed linear for almost all strategies and scenarios. This confirms previously known empirical evidence.
2. Dynamically adjusting timeout is always a good idea, except for coarse threshold-based adjustments that are overprotective in the case of light residual attack traffic.
3. Fine-grained linear timeout adjustments always outperforms fixed timeout and threshold-based adjustments, and is *significantly* better for moderate attack traffic rates.
4. The deterministic method of policing connections out of the queue is more robust to attack parameter optimisation (the "resonance effect") and has lower CPU overhead. However, the deferred method performs better against Poisson attacks, at the cost of a CPU overhead linear in the size of the queue.

We hope to further confirm these findings in future work by a) exploring a wider range of attack strategies and queue management algorithms and parameters in simulation, and b) conducting actual experiments in laboratory networks pitting various attacks against implementations of these strategies in different OS and applications. In these experiments we hope to test in conditions beyond some of the modelling assumptions made, such as Poission service rates for legimitate connections. In particular, we are aware that RTT distributions tend to be heavy-tailed [26], and we hope to test our results such conditions which are probably more realistic for normal network conditions.

Finally, while the work shown here is only applicable *as-is* to SYN-flood attacks it has the potential to be applied to other types of connection depletion attacks for TCP or other higher level protocols. One of the immediate difficulties of generalising this work, is that the standards for most relevant protocols (e.g. HTTP v1.1 [9], TLS v1.1 [6] and FTP [24]) do not define connection timeout mechanisms. Nonetheless, several applications that implement these protocols do include such timeout mechanisms (see Table 1 in the Appendix), and as such some of the results obtained might be applied to make them more resilient to the corresponding version of connection depletion attacks. Verifying this intuition for such protocol implementations is the object of ongoing research by our group.

References

1. J. Baras. Modeling and simulation of telecommunication networks for control and management. In *Proc. Winter Simulation Conf.*, 2003.
2. T. Benzel, R. Braden, D. Kim, C. Neuman, A. D. Joseph, and K. Sklower. Experience with DETER: A testbed for security research. In *Proc. Int. Conf. on Testbeds & Research Infrastructures for the DEvelopment of NeTworks & COMmunities (TRIDENTCOM 2006)*, 2006.
3. D. Bernstein. SYN cookies. `http://cr.yp.to/syncookies.html`, 2003.
4. J. Cao, W. Cleveland, D. Lin, and D. Sun. Internet traffic tends toward Poisson and independent as the load increases. In D. Denison, M. Hansen, C. Holmes, B. Mallick, and B. Yu, editors, *Nonlinear estimation and Classification*, volume 171 of *Lecture Notes in Statistics*, pages 83–110. Springer-Verlag, 2003.
5. C.-M. Cheng, H. Kung, and K.-S. Tan. Use of spectral analysis in defense against DoS attacks. In *Proc. IEEE Global Telecommunications Conf. (GLOBECOM)*, volume 3, pages 2143–2148, 2002.
6. T. Dierks and E. Rescorla. The transport layer security (TLS) protocol. Version 1.1. `http://tools.ietf.org/html/rfc4346`, Apr. 2006. RFC 4346.
7. W. Feng, E. Kaiser, and A. Luu. Design and implementation of network puzzles. In *Proc. Annual Joint Conf. of IEEE Computer and Communications Societies (INFOCOM)*, volume 4, pages 2372–2382, 2005.
8. P. Ferguson and D. Senie. Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing. `http://tools.ietf.org/html/rfc2267`, Jan. 1998. RFC 2267.
9. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol – HTTP/1.1. `http://tools.ietf.org/html/rfc2616#section-8`, June 1999. RFC 2616.
10. F. Gong. Deciphering detection techniques: Part III denial of service detection. `http://www.mcafee.com/us/local_content/white_papers/wp_ddt_dos.pdf`, Jan. 2003. McAfee Network Security Technologies Group.
11. X. Hoang and J. Hu. An efficient hidden Markov model training scheme for anomaly intrusion detection of server applications based on system calls. In *Proc. IEEE Int. Conf. on Networks (ICON)*, volume 2, pages 470–474. IEEE Computer Society Press, 2004.
12. A. Juels and J. Brainard. Client puzzles: A cryptographic defense against connection depletion. In *Proc. Network and Distributed System Security Symposium (NDSS)*, 1999.
13. S. Khan and I. Traoré. Queue-based analysis of DoS attacks. In *Proc. IEEE Work. on Information Assurance and Security (WIAS)*, pages 266–273, 2005.
14. J. C. Lui, V. Misra, and D. Rubenstein. On the robustness of soft state protocols. In *Proc. IEEE Int. Conf. on Network Protocols (ICNP)*, pages 50–60, 2004.
15. B. Madan, K. Goseva-Popstojanova, K. Vaidyanathan, and K. Trivedi. Modeling and quantification of security attributes of software systems. In *Proc. Int. Conf. on Dependable Systems and Networks (DSN)*, pages 505–514, 2002.
16. C. Meadows. A formal framework and evaluation method for network denial of service. In *Proc. IEEE Computer Security Foundations Work.*, 1999.
17. C. Meadows. A cost-based framework for analysis of denial of service networks. *Journal of Computer Security*, 9(1/2):143–164, 2001.
18. Microsoft Corporation. Security considerations for network attacks. `http://www.microsoft.com/technet/security/topics/networksecurity/secdeny.mspx`.

19. J. Mirkovic, S. Dietrich, D. Dittrich, and P. Reiher. *Internet Denial of Service: Attack and Defense Mechanisms.* Prentice Hall PTR, Dec. 2004.
20. J. Mirkovic and P. Reiher. A taxonomy of DDoS attack and DDoS defense mechanisms. *SIGCOMM Comput. Commun. Rev.*, 34(2):39–53, 2004.
21. J. Mirkovic, P. Reiher, S. Fahmy, R. Thomas, A. Hussain, S. Schwab, and C. Ko. Measuring denial of service. In *Proc. ACM Work. on Quality of Protection (QoP)*, pages 53–58. ACM Press, 2006.
22. J. Mirkovic, M. Robinson, and P. Reiher. Alliance formation for DDoS defense. In *Proc. New Security Paradigms Work. (NSPW)*, pages 11–18. ACM SIGSAC, 2003.
23. C. Nuzman, I. Saniee, W. Sweldens, and A. Weiss. A compound model for TCP connection arrivals for LAN and WAN applications. *Comput. Networks*, 40(3):319–337, 2002.
24. J. Postel and J. Reynolds. File transfer protocol (FTP). `http://tools.ietf.org/html/rfc959`, Oct. 1985. RFC 959.
25. M. Robinson, J. Mirkovic, S. Michel, M. Schnaider, and P. Reiher. DefCOM: defensive cooperative overlay mesh. In *Proc. DARPA Information Survivability Conf. and Exposition*, volume 2, pages 101–102, 2003.
26. S. Shakkottai, R. Srikant, N. Brownlee, A. Broido, and K. Claffy. The RTT distribution of TCP flows in the Internet and its impact on TCP-based flow control. Technical report, Cooperative Association for Internet Data Analysis (CAIDA), Feb. 2004.
27. R. Varanasi, V. Phoha, and S. Joshi. IP-traceback based attacker tracking: A probabilistic technique for detecting Internet attacks using the concept of hidden markov models. In *Proc. IEEE Information Assurance Work.* IEEE Computer Society Press, 2004.
28. A. Zuquete. Improving the functionality of SYN cookies. In *Proc. IFIP TC6/TC11 Joint Working Conf. on Communications and Multimedia Security*, pages 57–77, 2002.

A Additional Tables & Figures

Protocol	Server	Queue size c [cnx]	Timeout t_{out} [s]	Attack rate λ_m [cnx/s]
TCP	Linux 2.6.20	1024	180	5.7
	Solaris 9	1024	60	17.1
	Windows 2003	1000	21	47.6
HTTP/1.1	Apache 2.0	150	300	0.5
	IIS 6.0	8000	120	66.7

Table 1. Minimal attack rate exhausting all the connections of a server configured by default

Mean results of the legitimate connection completion rates when using the fixed-threshold and the linear timeout protection strategies are presented in Fig. 5. The standard deviation was smaller than 10^{-2} for all scenarios and strategies tested.

Legit. rate λ_l [cnx/s]	No protection	Threshold det.	Threshold def.	Linear det.	Linear def
128	9.08%	66.22%	85.86%	59.63%	92.40%
16384	9.01%	64.08%	62.43%	64.93%	87.25%

Table 2. Simulation results showing connection success rate for all strategies and different legitimate connection request rates, λ_m=10000 cnx/s, mean service time t_l=0.2 s, and timeout values t_0=10 s and t_1=0.2 s

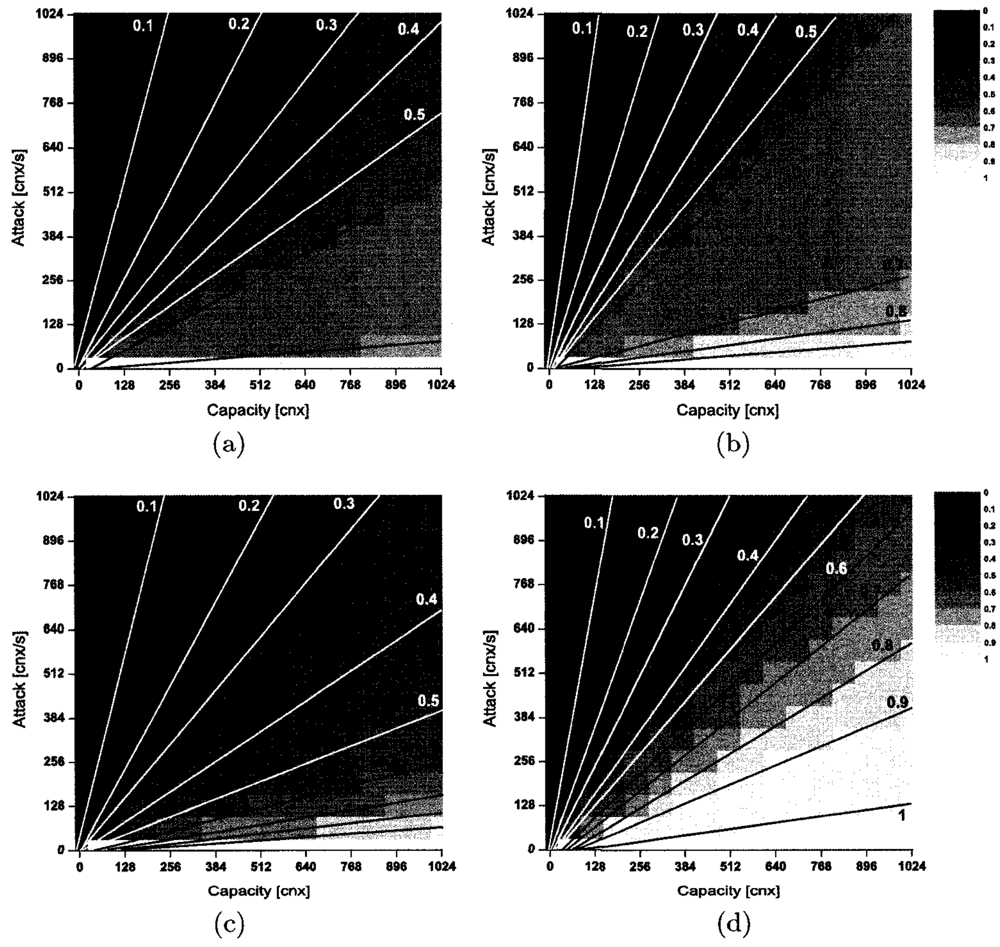

Fig. 5. Simulation results showing legitimate connection complete frequencies for various queue capacities and attack rates, λ_l=10 cnx/s, μ_l=1 cnx/s, t_0=75 s and t_1 = 1 s, for the single threshold, (a) and (b), and linear strategies, (c) and (d), using the deterministic and deferred methods, respectively.

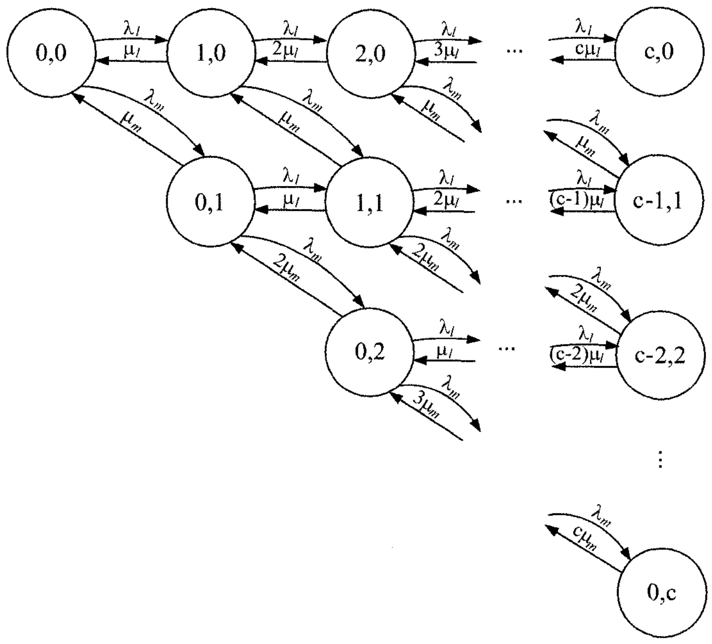

Fig. 6. Triangular DoS Markov chain model

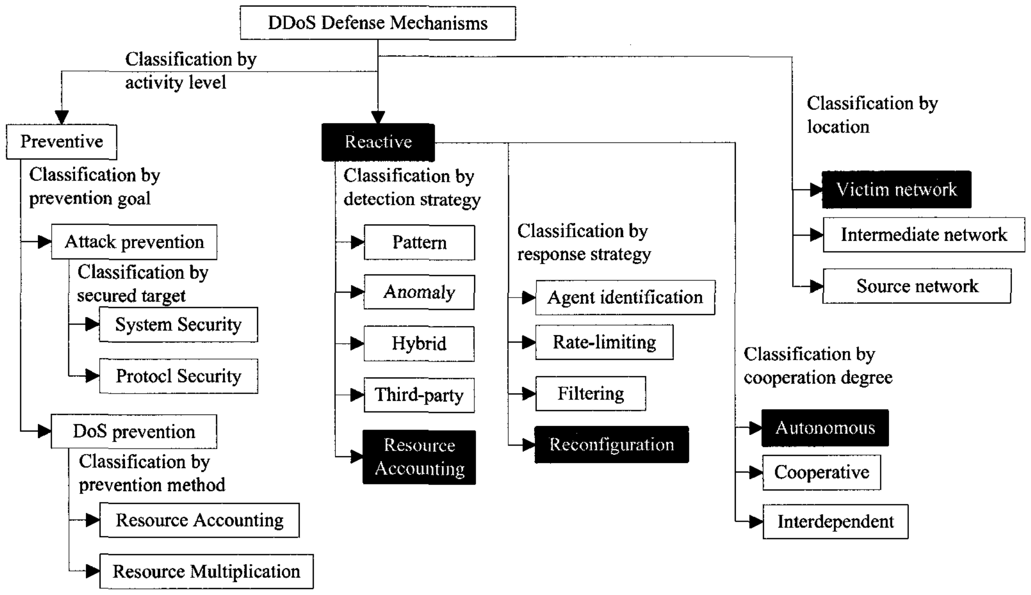

Fig. 7. Taxonomy of distributed denial-of-service defence mechanisms. The properties of the two protection strategies we analyse are highlighted.

ANNEXE B
Implementing and Testing Dynamic Timeout Adjustment as a DoS Counter-measure

[PAGE BLANCHE INTENTIONNELLE]

Implementing and Testing Dynamic Timeout Adjustment as a DoS Counter-measure

Daniel Boteanu Édouard Reich
José M. Fernandez
École Polytechnique de Montréal
{daniel.boteanu|jose.fernandez}@polymtl.ca

John McHugh
Dalhousie University
mchugh@cs.dal.ca

ABSTRACT

In this paper we experimentally analyse various dynamic timeout adjustment strategies in server queues as potential counter-measures against degradation of service attacks. Previous theoretical work studied the relative performance of both coarse-grained threshold-based timeout and fine-grained adjusment strategies where the timeout value is adjusted as the number of connections in the queue varies. In addition, two methods for removing timed-out connections were explored: the *deterministic* method where the expiry time is determined at connection arrival depending on the timeout value at that moment, and the *deferred* method where connections are continuously polled and flushed when the time-in-queue is larger than the current timeout value.

We report on experiments performed on a lab network where these strategies were tested against various configuration and attack parameters. The experimental results confirm the conclusions previously obtained from mathematical modelling and simulation, i.e. that a) finer-grained dynamic adjustment performs better than coarse-grained or no adjustment, and b) that the deferred method performs better than the deterministic one. Furthermore, our implementation of these counter-measures is very efficient and transparent with respect to the servers and applications it tries to protect. It could therefore be easily integrated into existing OS and applications or implemented in separate network devices, either on dedicated machines or network appliances.

Categories and Subject Descriptors

G.2.0 [**Computer-Communication Networks**]: General—*Security and protection*

General Terms

Experimentation, Measurement, Performance, Security

Keywords

Denial of Service, Degradation of Service, SYN flood

1. INTRODUCTION

Denial-of-service attacks have been plaguing the Internet for more than a decade. They have been a topic of much research for almost as long. Much has been done and written about modelling them and about potential counter-measures against them (see [1, 2] for complete surveys on the topic).

The amount of effort or resources expended by the attacker (whether bandwidth, expendable source IP addresses or even individual botnet machines) to attack a single target, is in most cases negligible compared to the amount of resources the defender would have to spend to maintain an equivalent availability of service. Previous work [3, 4] has tried to discuss and formalise such tradeoffs, and several counter-measures based on protocol modifications have been proposed to try to tip these tradeoffs in favour of the defenders. However, it is commonly assumed amongst security experts that with the current availability of botnets from which to launch these attacks, there is little one can do to prevent a single target from becoming completely flooded and hence unavailable: the tradeoffs are just hopeless. Nonetheless, in the context of large-scale orchestrated DoS campaigns such as the recent one in Estonia (see [5] for a very informative and quantitative technical summary), potentially involving hundreds or even thousands of targets, such tradeoffs might not be so advantageous to an attacker with finite resources. In addition, it is not only necessary to understand the tradeoffs in the context of *crippling* DoS attacks, where the target is reduced to 0% availability, but also the tradeoff between resources expended to degrade to or maintain an equivalent quality of service (QoS). In other words, these resource tradeoffs must also be understood in the context of *Degradation of Service* attacks [6, 1], where the objective is not necessarily to make a target completely unavailable but rather substantially decrease its QoS.

In previous work [7], we have explored these tradeoffs between timeout adjustment counter-measures and flooding attacks on connection-oriented protocols, the quintessencial example of which is the SYN-flood attack on TCP. The main interest of such counter-measures is that they can in principle be implemented in a transparent fashion by simple configuration adjustments, without extra hardware or software, and are complementary to the many other SYN-flood protection measures that have been developed and commercialised.

We describe in the next section the various counter-measures we studied and the attack models against which we evaluated their QoS maintenance performance. We report on the lab experiments we have performed in Sect. 3, describing laboratory setup, implementation details and testing and measurement methodologies. We describe the experimental results and compare them with the theoretical results previously obtained in Sect. 4. We discuss the limitations and practical applicability of our work in Sect. 5, and summarise our findings and conclude in Sect. 6.

2. PREVIOUS WORK

In these attacks, the defender resources being expended are available slots in a pending connection queue, while the attacker resources are numbers of connection attempts (e.g. SYN packets sent, in the case of SYN flood). The measure of QoS is the percentage of *legitimate* connection attempt requests that get serviced by the target. In [7] we considered the relative performance of various queue management strategies with respect to maintaining this QoS measure. In particular, since illegitimate connection that make it to the queue never get completed (e.g. because the corresponding ACK packets are never sent by the attacker), it would seem intuitive that lowering the timeout values when the queue is under attack would result in more illegitimate request being flushed out than legitimate ones. On the other hand, increasing it again when the queue empties out would prevent unadvertently flushing out legitimate connections when no longer under attack. With this in mind, we considered three types of strategies for adjusting timeout values:

A. The traditional *fixed* timeout strategy, where the timeout is always the same, regardless of queue occupation.

B. The *threshold* strategy where the timeout changes between two fixed values, as the number of connections in the queue crosses a pre-defined threshold. This coarse-grain adjusment method is not new and is already implemented in the TCP stack of some operating systems (OS), e.g. Microsoft Windows Server 2003 [8].

C. The *linear* method, a straightforward generalisation of the former, where the timeout value is determined according to a linear function depending on the number of connections in the queue, with two pre-defined empty- and full-queue timeout values.

Furthermore, we considered two timeout enforcement methods for flushing connections from the queue when the timeout is dynamically adjusted:

1. The *deterministic* method, where the expiry time for each connection is deterministically set when the connection arrives in the queue.

2. The *deferred* method, where connections in the queue are continuously polled, and flushed if they have been in the queue longer that the current timeout value.

Finally, two attack models were considered:

I. the *Poisson* attack model, a simpler albeit not very realistic model, where the interarrival times of illegitimate connection attemps follow an exponential (i.e. a Poisson model) distribution.

II. the *burst* attack model, where illegitimate connection requests arrive in (almost) instantaneous bursts of a fixed number of attempts, with burst spaced at a fixed burst interrarival time (BIT).

Using Markov chain-based queue models (in the case of Poisson attacks) and a custom-built event-driven simulator (for both types of attacks), we were able to verify that:

i. For all strategies and methods, the tradeoff between attack rate (connection attempts per second) and server queue size is essentially linear. Because of this tradeoff, the only parameter that significantly influences QoS degradation, for a fixed strategy and method, is the ratio between attack rate and queue size, which we called the *relative attack virulence.*[1]

ii. Fine-grained timeout adjustment (linear) always outperforms coarse-grained (threshold) adjustment, and the latter outperforms the fixed timeout strategy.

iii. The deferred method generally performs better than the deterministic one, except for the case of the linear deferred which performs worse than the linear deterministic method for burst attacks.

iv. In the case of burst attacks, some strategies and methods are quite sensitive to attack parameter optimisations. In particular, the lowest QoS for each defensive strategy is achieved when the BIT is set such that the queue is filled with a single burst. We called this phenomenon the *queue resonance effect.*

These results, if confirmed in real-life settings, would be of high practical interest. They would indicate how simple choices in queue management algorithms could result in dramatic improvements in resilience against QoS degradation attacks. Of course, this is only true for attack virulences that are not too low or not too high, i.e. attacks whose virulence is within the *window of interest,* since for attacks outside this window the QoS degradation is equally negligible or overwhelming, for all strategies. Therefore, we enunciate the following hypothesis based on the theoretical evidence of [7]:

Main Hypothesis. Within the window of interest (attack virulences between $1/8$ and $8\ s^{-1}$), finer-grained timeout adjustment strategies using the deferred method will always perform better against SYN-flood attacks than those using coarser-grained adjustment or the deterministic method.

Until we can verify this hypothesis for real-life networks, we have to content ourselves with gathering supporting evidence from testing on laboratory networks. The setup and methodology we used to do so in the case of SYN-flood attacks is described in the next section.

3. EXPERIMENTAL SETUP AND TESTING METHODOLOGY

The five components of our experimental setup are the following:

1. The attack traffic generator, generating illegitimate SYN packets on the network.

2. The legitimate traffic generator, attempting to establish fully fledged TCP connections.

[1]Intutively, the attack virulence indicates how many times per s the attack could fill the queue, if there were no legitimate requests and a very low timeout value.

3. The server, whose TCP stack half-open connection queue is being flooded.

4. The Queue Guardian (QG), a separate application whose role is to protect the server queue.

5. The network, on which both kinds of traffic travel.

3.1 Attack Traffic Generator

For this component, we used the IXIA 400T, a special purpose traffic generator chassis, built for performance and conformance testing of network applications. The model we used has four separate Ethernet ports, capable of generating traffic up to 1 Gbps each.

In order to generate the two types of malicious traffic we wanted to test (Poisson and burst), we used the IxExplorer application that runs on the IXIA hardware. Since neither the hardware nor the software can natively generate Poisson traffic, this type of attack was synthesised by cyclically sequencing 255 different *modes*, each mode consisting in sending one single SYN packet. For each attack rate, pauses between modes were statically set to random values following an exponential distribution. We performed a Kolmogorov-Smirnov test on the inter-arrival times of the IxExplorer-generated traffic measured on the server. The maximum difference between the theoretical exponential and the observed cumulative distribution functions (CDF) was as low as 0.12 for an attack of 1000 packets/s, which confirms that the traffic follows the Poisson process model closely.

For burst traffic, we ran experiments with different BIT values, where the number of packets in a burst was chosen so that overall attack rate remained the same for all experiments. IxExplorer allowed us to generate burst attack traffic using only one mode, the burst mode, for BIT $< 8s$. For burst attacks with BIT $\geq 8s$, several modes were sequenced, each mode sending an entire burst followed by one or several "pause" modes. In the first case, we were able to script several experiments at various BIT values, one after the other. A pause at least as long as the server's largest timeout value was inserted between attacks in order to prevent the experiments results from being contaminated by previous ones.

3.2 Legitimate Traffic Generator

We used a home-made C++ application to generate the legitimate traffic necessary for successful TCP handshake. Both the SYN and ACK messages were sent with exponentially distributed inter-arrival times. Contrary to TCP stack implementations in standard OS, this test application will *not* send a SYN retry message if there is no response from the server. This was a deliberate choice meant to keep the connection attempt rate constant and independent of the connections complete rate. For performance measuring purposes, all the legitimate SYN messages came from the same IP address. This address is discriminated only when counting the total number of legitimate connection attempts. After a TCP handshake is completed, the application will send a RST message in order to free the connection on the server side. We deployed the legitimate traffic generator on a dedicated machine running Gentoo Linux, with 2 GB of memory.

The server whose TCP stack is flooded is also a Gentoo Linux, with 2 GB of memory, which allowed us to experiment with queue sizes up to 16384.

3.3 Queue Guardian (QG)

Rather than modifying the TCP stack kernel code, which is neither easy nor practical in real-life deployments, we chose to implement the dynamic timeout strategies on a separate application, in a manner transparent to the server and the legitimate clients. The QG has four different roles:

1. It maintains an up-to-date mirror of the server queue. This is achieved by sniffing the network connection and interpreting packets being send and received by the server. We used the `libpcap` library to sniff all IP packets on the network.

2. It drops connections from the mirror queue, according to the chosen timeout adjustment strategy and connection expulsion method.

3. It forces the server queue to drop the same connections that were dropped from the mirror queue. This is achieved by sending RST packets to the server. The IP and TCP headers are spoofed so that the message appears to come from the original client. In order to send the spoofed RST packets at high speeds, this role was implemented using raw sockets.

4. It regularly logs the state of the queue, as well as the number of different types of packets sniffed on the network. This log is used later for evaluating the performance of the timeout strategy under test.

For the deterministic method, we used a priority queue implemented as a red-black tree to store the connections, ordered by their expiration time. When all legitimate connections get served, the complexity of the algorithm is $O(\log cN_m + cN_l)$, where c is the size of the server queue and N_m and N_l are the number of SYN-ACK responses sent to malicious and legitimate SYN packets, respectively. In the deferred method, only the oldest connection in the queue needs to be analysed: if it is present in the queue for longer than the current timeout, it will be dropped from the queue. Hence, a single FIFO ring-buffer can be used to implement this method. When all legitimate connections get served the complexity of the algorithm is $O(N_m + cN_l)$. In practice, however, the legitimate connections are almost always at the end of the queue so only $N_m + N_l$ atomic operations need to be performed. Finally, and for performance reasons, we chose to implement each of these four roles in a separate thread in the QG application. The QG is run on a separate machine, based on a Intel Core 2 Duo processor at 2.16 GHz.

3.4 Network Setup

A 16-port gigabit switch (Linksys SRV-2016) was used to connected all these components together. The legitimate traffic generator machine, the server and the IXIA traffic generator were each connected to a separate port on the switch. For sniffing purposes, the QG machine was connected on a switch port setup to mirror the server port. For sending RST packets, a separate card on the QG machine was connected to another network port on the switch. Other deployment schemes are possible as well and we will discuss them in Sect. 5. Fig. 1 illustrates the network connections between the components we have used.

3.5 Testing Methodology

In all the experiments we ran, the following steps were followed in sequence:

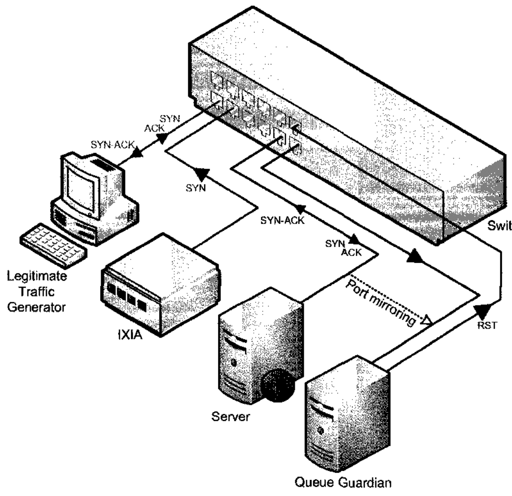

Figure 1: Experimental lab network setup

1. The server queue size was configured with the value required for testing.
2. The server timeout was configured to be at least as long as the longest timeout on the QG. This way, all the connections drops were triggered by the QG.
3. The legitimate connection traffic generator was started with the connection arrival and connection completion rates required for testing.
4. The QG was configured with the required parameters and started.
5. The attack traffic parameters were configured in IxExplorer.
6. The attack was started and the experiment was run during a period of time ten times longer than the longest timeout on the QG.
7. The connection success rate was computed based on the QG's log.

Connection completions correspond to ACK messages being sent to the server. Legitimate connection attempts correspond to SYN messages being sent from the legitimate IP address. The connection success rate was computed as the ratio between the connections completed and the legitimate connection attempts during the attack.

4. RESULTS AND ANALYSIS

We measured the performance of the two dynamic timeout strategies, threshold and linear, along with the fixed timeout strategy for comparison purposes. For the dynamic strategies, we tested both the deterministic and deferred methods of assigning timeouts to connections. We compare these results with those obtained in a previously built home-made traffic simulator (described in [7]) that implements these strategies. The attack traffic was generated using both models described above. In both cases we tested the attacks

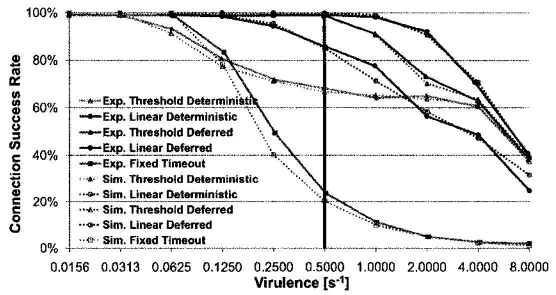

Figure 2: Legitimate connection complete rate (y-axis) for various strategies, with fixed queue size $c = 128$, legitimate traffic rate 100 packets/s, mean service time 0.2 s, initial and final timeout values 10 s and 0.2 s, respectively, for various Poisson attack virulences (x-axis).

against a small queue size of 128 and a more reasonable queue size of 1024.

4.1 Poisson attacks

In the case of Poisson attacks, we explored virulences from 0.015 to 8 s^{-1}. The corresponding attack speeds varied from 2 to 1024 packets/s when testing against a queue size of 128, and from 32 to 8192 packets/s when testing against a queue size of 1024. The legitimate connection attempt rate was 10 packets/s and the mean RTT time for the legitimate traffic was 200 ms (as observed experimentallly in [9]). The fixed timeout strategy used a timeout value of 10 s and the dynamic timeout strategies used empty- and full-queue timeout values of 10 s and 200 ms, respectively. Results for the tests against a queue size of 128 are shown in Fig. 2.

Overall, the experimental results are very similar to the simulation results. The average difference between the simulation and experimental results is 2%. The greatest discrepancy (17%) was measured for the linear deterministic strategy faced with an attack of virulence 8 s^{-1} against a queue size of 1024.

As anticipated from the simulations, results for low and high virulences are not interesting. For low virulence values ($< 0.05s^{-1}$) the attack is not strong enough to degrade QoS at the the server, even when using the fixed timeout strategy. For very high virulence values ($> 8s^{-1}$) the attack is so strong that none of the dynamic timeout strategies can maintain a connection success rate greater than 50%. In between these values, the window of interest, several conclusions can be drawn that confirm previous theoretical results.

First, the dynamic timeout strategies perform better or equivalent than the fixed timeout strategy. We measured differences of up to 85% between the linear deferred strategy and the fixed timeout strategy, and up to 50% between the threshold deterministic and the fixed timeout strategy around virulences of 1 s^{-1}. Second, the deferred technique always performs better than the deterministic technique. Differences up to 30% can be observed between the deferred and the deterministic techniques around virulences of 2 s^{-1}. This is due to the fact that the deferred technique is more reactive, deciding whether a connection should expire or not based on the current status of the queue, as opposed to the

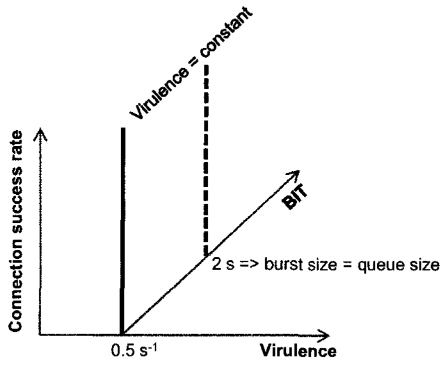

Figure 4: Relationship between Poisson attack parameters of Fig. 2 represented here on the xz-plane, and burst attack parameters from Fig. 3, yz-plane.

status of the queue at the time of the connection arrival in the case of the deterministic technique. Third, the linear timeout strategy performs better than the threshold timeout strategy with the exception of the deterministic technique for virulence values greater than 1 s^{-1}. The threshold strategy has an overprotective behaviour when faced to an attack, and this seems to correct some of the delayed reactivity of the deterministic technique for medium and high virulence values.

4.2 Burst attacks

In order to study the generality of these results with respect to different attack types we also used a deterministic process to generate bursts attacks. A virulence of 0.5 s^{-1} was chosen, which corresponds to attack rates of 64 and 512 packets/s when testing against queue sizes of 128 and 1024, respectively. The average connection success rates over 9 experimental runs and their corresponding standard deviation for the queue of size 128 are illustrated in Fig. 3. The vertical black line at BIT = 0.015625 s in Fig. 3 represents that the packet inter-arrival time is the same as the mean packet inter-arrival time in the Poisson experiments at virulence 0.5 s^{-1}, marked by the vertical black line in Fig. 2. The dashed black vertical line at BIT = $2s$ in Fig. 3 marks the point where one single burst would fill up an empty queue entirely. Fig. 4 offers a three-dimensional illustration of the correspondence between the Poisson and the burst figures.

Two "phases" can be observed when analysing the burst attack results. The "liquid phase", at the leftmost part of the figures, with BIT $< 2s$, corresponds to attack traffic bursts smaller than the queue size. The "solid", rightmost phase, for BIT $> 2s$, corresponds to attack traffic bursts greater than the queue size. The resonance effect is created at BIT = $2s$, corresponding to bursts of the same size as the server queue. In simulations, the fixed timeout strategy performance is practically null at this value. This is due to the fact that the simulated attack and legitimate traffic start at the same time and the attack burst instantly fills up the entire queue. During a period of 10 s, equal to the timeout value, the queue is full and no legitimate connection attempts can be processed. After this period, exactly after the malicious connections are dropped from the queue, the following burst arrives and fills up all the queue once again. This happens when the burst traffic is perfectly synchronized with the queue timeout, as is the case with the simulator. In experiments, however, we do not observe the same behaviour.

First of all, the legitimate traffic and the malicious traffic are not synchronised. By the time the first attack burst arrives, around three slots in the queue are already used by legitimate connection attempts, so three of the attack packets are discarded by the server. During a period of 10 s, only the number of slots used by legitimate connection attempts at the time the first burst arrived will be available. However, because there are only 10 legitimate connection attempts per second, and because the legitimate connections complete rather quickly (5 every second), the few free slots in the server queue are enough for a large percentage of legitimate connection attempts to complete. Furthermore, in experiments, the burst are never instantaneous due to packet transmission times and eventual collisions in the Ethernet network. This allows for legitimate connection attempts to infiltrate the burst and thus reduce the burst efficiency for the attacker. Due to the above mentioned factors, we can say that the network acts as a "low-pass filter" thus greatly diminishing the resonance effect. In simulations, the fixed timeout strategy is influenced by the resonance effect with "harmonics" at BIT = $2^{-k}s$, for $k = \{0..5\}$. In experiments, however, the resonance effect is absorbed by the network. The only two strategies that seem to be slightly affected by the resonance effect in experiments, are the linear deterministic and the threshold deferred timeout strategies, and this only for the harmonic at BIT = $1s$.

Is it important to note that the deferred method, which performs better than the deterministic one, is also more robust and consistent, having lower standard deviation values. The fixed timeout strategy, on the other hand, is the most unstable, both in simulation and in experiments, with maximum standard deviation values of over 10%.

5. LIMITATIONS AND FUTURE WORK

Although the dynamic timeout strategy implementations have a low CPU overhead, there are two limitations that prevented us from testing higher attack rates and queue sizes.

The first limitation is due to the network architecture we implemented. The malicious SYN packets and the QG-generated RST packets are sent through the same switch and thus, at high attack rates, some RST packets are dropped by the switch due to Ethernet collisions. This creates a cumulative difference in the size and content of the mirror queue compared to the server queue. For example, an average of 0.3% of the resets sent by the QG are dropped by the switch for an attack rate of 8192 packets/s, which is equivalent to 6.65 Mbps. For attack rates higher than 8 Mbps the RST packet loss is so significant that the QG's mirror queue is completely corrupt in a matter of seconds. This limitation can be overcome by using a different network architecture, where the QG would be deployed either directly on the server or on a network appliance directly connected to the server, while still preserving the transparency to both the server application and OS and the legitimate clients.

The second limitation is due to CPU consumption on the QG machine. When using the deterministic technique to protect a queue size of 1024 with 10 legitimate connection attempts per second against an attack of 6.65 Mbps, the

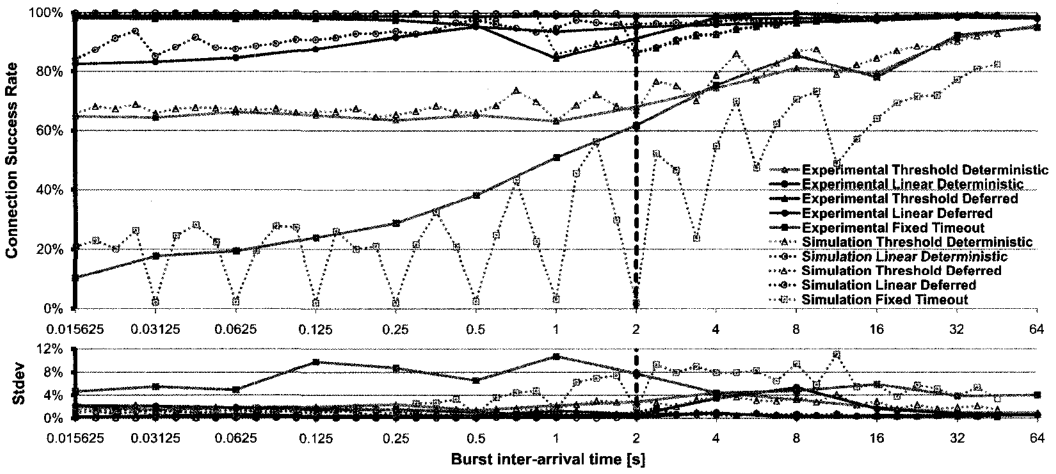

Figure 3: Legitimate connection complete rate and standard deviation (y-axis) for various strategies, with fixed queue size c = 128, legitimate traffic rate 100 packets/s, mean service time 0.2 s, initial and final timeout values 10 s and 0.2 s, respectively, virulence 0.5 s^{-1} for various burst inter-arrival times (x-axis), over 9 runs.

average CPU usage was of 9%. The deferred technique consumed in average 8% of the CPU, for the same attack parameters. Most of this CPU time is spent in kernel mode, handling the sending and sniffing of IP packets. We ignored the previous architectural limitation of RST packets loss and managed to reach attack rates as high as 300 Mbps for the deterministic method and 62 Mbps for the deferred method, compiled in debug mode, before the approaching 100% CPU usage, and hence starting to miss some of the packets that traverse the network. It is our belief that a more powerful machine should be able to handle up to gigabit attack traffic.

6. CONCLUSIONS

In this paper, we tested the hypothesis that within the window of interest (attack virulences within 1/8 and 8 s^{-1}), finer-grained timeout adjustments strategies using the deferred method perform better than ones using coarser-grained adjustment or the deterministic method. We implemented both a fine-grained, linear and a coarse-grained, threshold dynamic timeout adjustment strategy in their deterministic and deferred variants. The performance measures obtained in these laboratory experiments for the different strategies against Poisson attack traffic was consistent with the performance measures against Burst traffic. First, that using a dynamic timeout strategy is always a good idea. Second, that the deferred method performs better than the deterministic technique, and has slightly lower CPU usage, due to having a lower algorithmic complexity. Third, the linear, fine-grained timeout adjustment strategy performs better than the threshold, coarse-grained timeout adjustment strategy when in their deferred implementation. Finally, the resonance effect that we expected when testing against burst attack traffic is very limited in experiments, due to network delays created by network equipment buffers, Ethernet collisions and non-instantaneous packet send times.

7. REFERENCES

[1] J. Mirkovic, S. Dietrich, D. Dittrich, and P. Reiher. *Internet Denial of Service: Attack and Defense Mechanisms.* Prentice Hall PTR, December 2004.

[2] J. Mirkovic and P. Reiher. A taxonomy of DDoS attack and DDoS defense mechanisms. *SIGCOMM Comput. Commun. Rev.*, 34(2):39–53, 2004.

[3] C. Meadows. A formal framework and evaluation method for network denial of service. In *Proc. Computer Security Foundations Workshop (CSFW)*, pages 4–13, 1999.

[4] C. Meadows. A cost-based framework for analysis of denial of service networks. *J. Comp. Security*, 9(1/2):143–164, 2001.

[5] José Nazario. Estonian DDoS attacks - a summary to date. `http://asert.arbornetworks.com/2007/05/estonian-ddos-attacks-a-summary-to-date`, February 2007.

[6] J. Mirkovic, P. Reiher, S. Fahmy, R. Thomas, A. Hussain, S. Schwab, and C. Ko. Measuring denial of service. In *Proc.ACM Workshop on Quality of Protection (QoP)*, pages 53–58, 2006.

[7] Daniel Boteanu, José M. Fernandez, John McHugh, and John Mullins. Queue management as a DoS counter-measure? In *Proc. Information Security Conference (ISC)*, 2007. *To appear.*

[8] Microsoft Corporation. Security considerations for network attacks. `http://www.microsoft.com/technet/security/topics/networksecurity/secdeny.mspx`.

[9] Srinivas Shakkottai, R. Srikant, Nevil Brownlee, Andre Broido, and K.C. Claffy. The RTT distribution of TCP flows in the internet and its impact on TCP-based flow control. Technical report, Cooperative Association for Internet Data Analysis (CAIDA), February 2004.