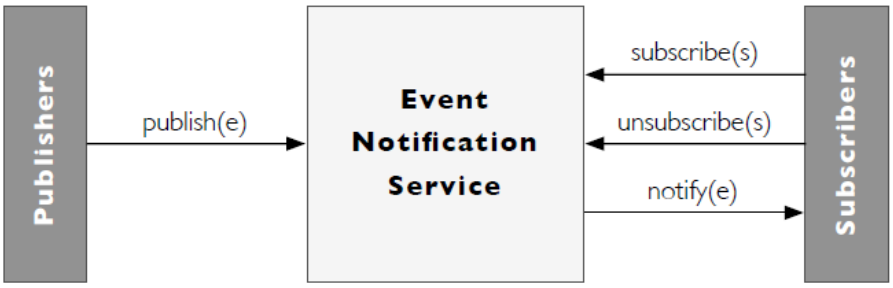
Il paradigma Publish/Subscribe

Introduzione

Fin dagli albori dei sistemi distribuiti, il paradigma client/server ha da sempre fatto da padrone nel mondo delle comunicazioni. Nel mondo di internet, con particolare riferimento all'ultima decade, l'evoluzione delle transazioni sulla rete ha messo in luce limiti ed handicap del datato client/server.
In una realtà fatta di social networks e distribuzione di eventi si evince la necessità di costruire un nuovo modo di diffusione dell'informazione, dove l'attore interessato alla notizia non per forza debba conoscerne il produttore. In altre parole, proponendo un parallelismo con il client/server, si desidera che il client possa esporre le proprie richieste aspettandosi delle risposte senza preoccuparsi a riguardo di *chi* è tenuto ad assolverle.
Nel contesto descritto, il paradigma Publish/Subscribe [4] [pub/sub] si pone come ottima soluzione. Nel citato paradigma il client denuncia il proprio interesse nei confronti di una determinata informazione una ed una sola volta, il server, o più in generale il produttore dell'informazione, rende nota la pubblicazione di una notizia, sarà una poi una parte terza a preoccuparsi di mettere in connessione le denunce di interesse con le informazioni pubblicate.
Nel paradigma introdotto il client è detto *Subscriber*, il server è detto *Publisher*, la denuncia dell'informazione è detta *Sottoscrizione* e l'informazione è detta *Evento*. La parte terza tenuta al managing delle sottoscrizioni/pubblicazioni, nonché alla *notifica* degli eventi ai subscribers interessati è detta *Event Notification Service* [ENS].
Di seguito è proposto uno schema concettuale del paradigma pub/sub:



Per quanto detto il paradigma di interazione proposto consente:

1. disaccoppiamento spaziale: *ne il subscriber e ne il publisher sono tenuti a conoscersi*
2. disaccoppiamento temporale: *la pubblicazione e la notifica degli eventi avviene in maniera asincrona*
3. disaccoppiamento di flusso: *il flusso di produzione e di consumo delle informazioni avvendono in maniera del tutto indipendente, ad esempio il publisher può pubblicare informazioni indipendentemente dalla presenza (o meno) delle sottoscrizioni.*

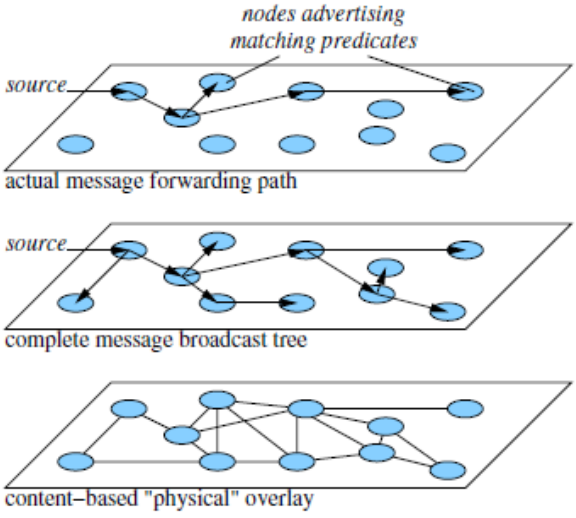
La diffusione degli eventi pubblicati all'interno del sistema è chiaramente la chiave del pub/sub. Tale funzionalità è oggetto dell'implementazione dell'ENS che può essere eseguita in vari modi, dal più scontato flooding ai più complessi meccanismi di alberi di distribuzione [3] ed interest clustering [10][2].
L'ENS, presentato come modulo logico, consiste, in termini implementativi, nella realizzazione del routing degli eventi verso i subscriber pertinenti. Tale modulo si presta chiaramente ad implementazioni sia centralizzate che distribuite, chiaramente l'interesse della letteratura è rivolto alle seconde, che impongono limiti di scalabilità molto più laschi.
La distribuzione delle informazioni è strettamente dipendente dal modo con cui vengono eseguite le sottoscrizioni agli eventi. Tali modalità sono sostanzialmente due: topic-based e content-based.

Nonostante i nomi dei due approcci possano essere in parte auto esplicativi, riteniamo utile spendere due righe per scendere ad un livello di dettaglio migliore.

- Topic-based: *il subscriber comunica all'ENS i topic di interesse, ad ogni evento saranno chiaramente legati dei topic ai quali l'evento è connesso. Questo approccio, particolarmente semplice, non consente chiaramente di eseguire alcun tipo di filtraggio sul contenuto degli eventi, ad esempio, se ci si dichiara interessati al topic "calcio" si riceveranno eventi riguardanti anche le squadre di non interesse. Un tuning sulla granularità dei topic può risolvere molte problematiche, tuttavia, un approccio topic-based è comunque limitativo.*
- Content-based: *in questo caso il subscriber denuncia di essere interessato ad un determinato argomento imponendo limitazioni anche sul suo contenuto, ad esempio dicendo che vuole ricevere eventi sul calcio solamente se la sua squadra del cuore vince. Una analisi pienamente semantica sull'evento è chiaramente un obbiettivo da raggiungere, tuttavia, data la complessità, proporre uno schema generale che tutte le sottoscrizioni devono seguire, nel quale è possibile definire vincoli matematici è già un buon risultato.*

Stato dell'arte

Attualmente esistono varie soluzioni che implementano il paradigma pub/sub. Una delle prime è quella rappresentata da SIENA [3] [12]. Nell'articolo A.Carzaniga et al. presentano un elegante schema di distribuzione degli venti basato su di un approccio CBCB *(combined broadcast and content-based)* capace di implementare un meccanismo di sottoscrizione content-based.
Per semplicità supponiamo che nella rete esista un solo publisher. L'implementazione dell'ENS è la seguente: il publisher, che è a conoscenza della rete in cui orbita, costruisce un proprio albero di broadcast (secondo una determinata euristica il cui dettaglio è omesso) con cui è in grado di raggiungere ogni altro nodo della rete (che è chiaramente un potenziale subscriber). In ogni nodo coninvolto dall'albero di broadcast è stanziata una sorta di routing table che descrive chi sono i *next hops* dei messaggi che riceverà. In questo modo, per innescare la distribuzione è sufficiente spedire l'evento ai nodi a distanza uno nell'albero di broadcast, saranno poi questi e i successivi ad effettuare i rilanci.



Fin qui ogni messaggio è semplicemente distribuito nell'intera rete, lo scopo di SIENA è quello di potare l'albero di broadcast in modo tale da percorrere solamente i rami che collegano il publisher ai sottoscrittori interessati. Implementare tale meccanismo è particolarmente facile: a fronte di una sottoscrizione basta ripercorrere l'albero di broadcast fino alla sorgente aggiornando le tabelle di routing coinvolte. Se ad esempio un nodo Y ha tre figli A,B,C nell'albero di broadcast e la sua tabella di routing è la seguente:

10 > x > 30	Send to	B
Else	Send to	--

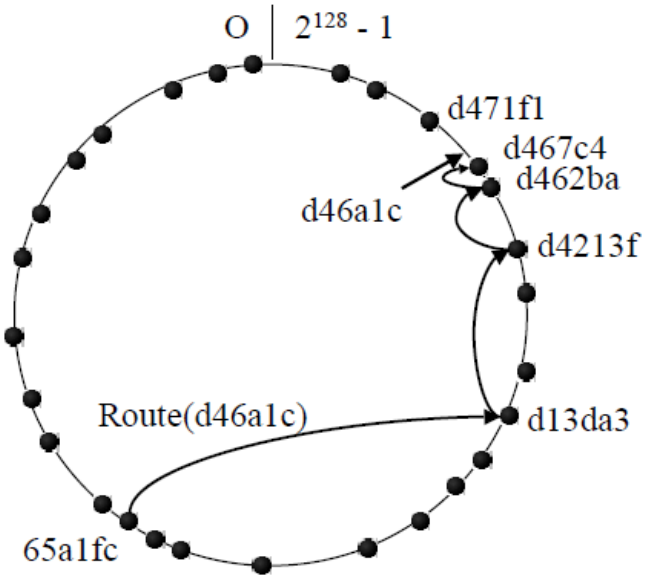
alla ricezione di una sottoscrizione da parte di C che denuncia l'interesse per x>40 e 10>x>30, il nodo Y aggiornerà la tabella di routing (vedi di seguito) ed invierà un avviso al proprio predecessore nell'albero di broadcast avvisando di essere interessato a ricevere anche gli eventi che hanno x maggiore di 40.

10 > x > 30	Send to	B,C
x > 40	Send to	C
Else	Send to	--

Chiaramente, dalle tabelle di routing deduciamo che il nodo A non è sottoscritto per l'evento x, e che i nodi B e C sono sottoscritti ad x ma sono interessati a tale evento se e solo se questo rispetta i vincoli numerici descritti.
L'evoluzione di tale approccio alla presenza di *n* subscribers è semplicissima: ogni nodo dovrà tener conto non più di una sola tabella di routing ma bensì di *n*. Si noti come si possibile implementare politiche di ottimizzazione, al fine di limitare il numero di messaggi scambiati per il pruning degli alberi, anche facendo uso di eventuali politiche di aggregazione tra le tabelle.
Un approccio completamente diverso, capace di implementare la tecnica topic-based, è rappresentato da Scribe [13]. Questo protocollo implementa un ENS tramite la tecnica del randez-vous routing. Tale tecnica si basa sull'utilizzo di due funzionalità di base in grado di mappare i nodi coinvolti a fronte di sottoscrizioni e pubblicazioni. Tali funzioni sono dette EN ed SN e vengono chiaramente sfruttate per decidere quali nodi sono responsabili di quali sottoscrizioni e quali nodi vanno coinvolti per il routing degli eventi, nel dettaglio abbiamo:

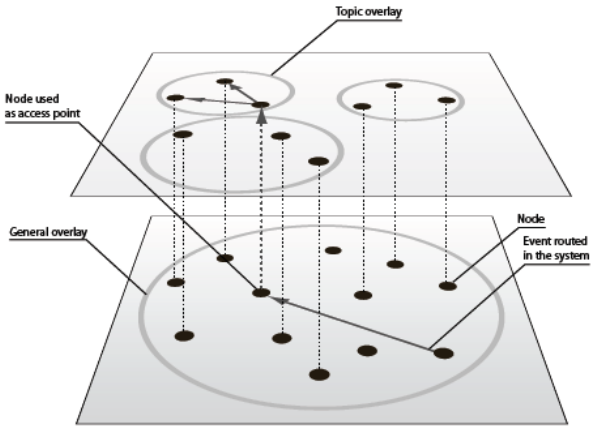
- SN(s): *tale funzione, invocata sulla sottoscrizione s, fornisce in output una serie di nodi che saranno i responsabili dell'archivio e del matching di tale sottoscrizione con gli eventi che verranno pubblicati*
- EN(e): *tale funzione, invocata sull'evento da pubblicare e, fornisce in output una serie di nodi che sono i responsabili delle sottoscrizioni associate a quell'evento*

A fronte di una sottoscrizione, il subscriber invoca SN(s) e quindi invia ai nodi restituiti l'avviso con la propria sottoscrizione. A fronte di un pubblicazione, il publisher invoca EN(e) ed invia la pubblicazione a tutti i nodi restituiti. Si noti come SN ed EN abbiano una stretta dipendenza complementare, e soprattutto, come un sistema basato su queste primitive possa variare il livello di replicazione delle informazioni semplicemente cambiando l'implementazione delle due funzioni.
L'implementazione di Scribe consiste sostanzialmente nell'implementazione delle funzioni SN ed EN.



Per fare ciò, Scribe si appoggia su una DHT su cui è dispiegato uno spazio dei nomi secondo le politiche del sottosistema Pastry [5]. Avendo a disposizione una DHT è possibile creare delle funzioni EN ed SN particolarmente efficaci che producono delle chiavi nello spazio dei nomi, senza preoccuparsi dei nodi che ne sono responsabili; sarà poi il sottosistema DHT a gestirne l'associazione.
Grazie al disacccoppiamento tra output delle funzioni EN ed SN e nodi attualmente presenti nel sistema, Scribe può far leva su tutta la prolifica ricerca nel campo delle DHT, proponendosi così come soluzione dalla alta affidabilità e scalabilità. Inoltre, sempre grazie all'uso di base delle tabelle hash distribuite, il protocollo pub/sub in oggetto garantisce caratteristiche prestazionali di rilievo, che danno modo al progettista di potersi concentrare sul solo sviluppo delle funzionalità SN/EN, proponendo messe a punto ad-hoc per tutte le possibili istanze di problema.
Una soluzione interessante, che per la sua complessità releghiamo alla semplice citazione, è quella proposta ultimamente da sub2sub. Tale soluzione, anch'essa basata su concetti legati all'hash, propone una distribuzione di sottoscrizioni content-based espresse mediante attributi a virgola mobile, su di un

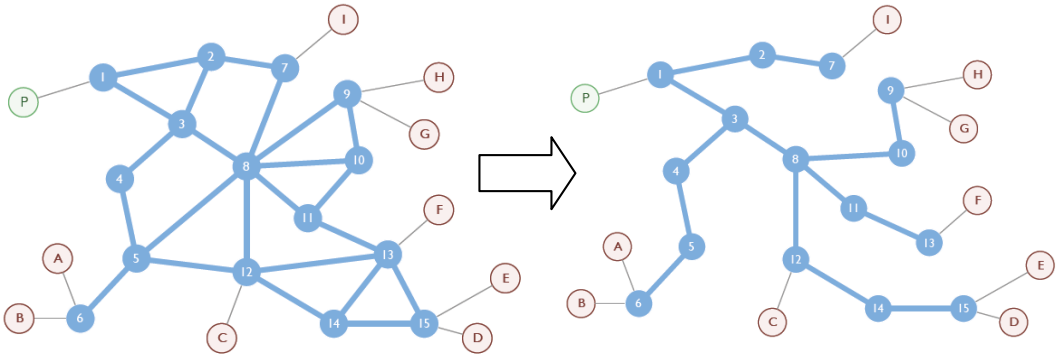
complesso spazio delle chiavi. Anche qui lo storing ed il retrive delle informazioni sono guidati da funzioni comuni a disposizione di tutti i nodi.
Concludendo la panoramica sullo stato dell'arte ci è sembrato importante dare enfasi anche alle soluzioni basate su politiche di interest clustering [2]. La linea guida di tali soluzioni è rappresentata dal raggruppamento dei nodi in sottogruppi logici all'interno dei quali sono implementate politiche di distribuzione in broadcast dell'evento. Tale approccio introduce un forte disaccoppiamento tra routing dell'evento e managing delle sottoscrizioni: mentre nel caso di Siena bisognava continuamente mantenere l'albero di distribuzione a partire da *ogni* publisher, qui abbiamo che il publisher deve solo raggiungere il gruppo dei sottoscrittori, sarà poi questo a provvedere alla distribuzione dell'evento. In questo contesto un esempio particolarmente rappresentativo è quello proposto da R.Baldoni et al. con TERA[10]. Tale protocollo implementa la politica di sottoscrizione topic-based e si basa sul mantenimento di un overlay di base in cui i nodi interessati allo stesso topic si raggruppano in overlay di livello superiore. Gli eventi che raggiungono almeno un nodo degli overlay di livello superiore vengono propagati il prima possibile nell'intero gruppo a mezzo broadcast.



Per quanto detto, a fronte di una pubblicazione, un publisher dovrà solamente cercare il gruppo pertinente al proprio evento da pubblicare. Tale ricerca viene effettuata a mezzo random walk ed è ottimizzata tramite tabelle presenti in ogni nodo, nelle quali vengono uniformemente distribuite delle tuple che mappano topic/gateway. I gateway altro non sono che dei nodi facenti parte degli overlay di livello superiore.
Le prestazioni offerte da TERA, anche in termini di overhead di gestione, sono notevoli e lo pongono come validissima alternativa ai sistemi precedentemente descritti. Inoltre, il crescente interesse per l'interest clustering, altro non fa che confermare la bontà della soluzione proposta e stimolare la ricerca verso nuovi protocolli basati su tale approccio, capaci di offrire un minor overhead e una maggiore resistenza e trasparenza al dinamismo e alla eterogeneità della rete.

Un risultato importante

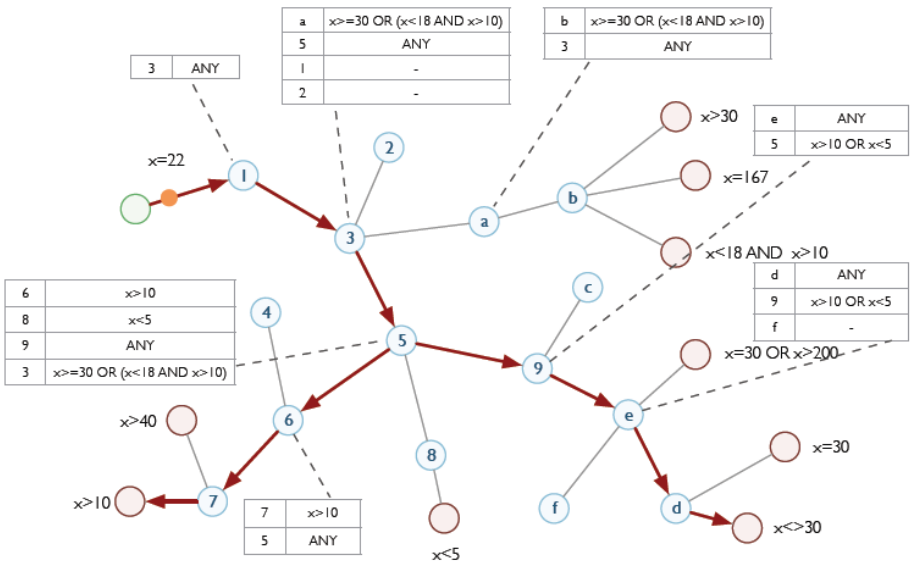
Un risultato importante, seppur non recentissimo, è quello rappresentato da Siena [3] [12]. Tale protocollo, oltre a dare un'ottima interpretazione del paradigma pub/sub con sottoscrizioni content-based, propone soluzioni assolutamente generali per la distribuzione degli eventi sulle reti.
L'articolo pubblicato nel 2004 da Antonio Carzaniga, Matthew J. Rutherford ed Alexander L. Wolf costituisce uno dei primi sistemi pub/sub mai proposti in letteratura, che grazie anche alle sue analisi sperimentali è punto di riferimento anche per i più moderni protocolli.
Come detto più volte l'implementazione di un sistema pub/sub consiste sostanzialmente nella implementazione del modulo ENS, che altro non è che la definizione di uno schema di routing degli eventi all'interno di una rete.
Nell'articolo gli autori definiscono come schema di routing quello che chiamano CBCB *combined broadcast and content-based*. L'approccio, relativamente semplice, si bassa sull'uso combinato di un albero di broadcast e di una politica di pruning.
Ottenere un albero ricoprente a partire da un grafo è una problematica ben nota e risolta. Non ci dilungheremo a riguardo, assumeremo di fatti che esiste un algoritmo in grado di calcolare uno spanning tree che sia aciclico e possibilmente minimo; ovviamente a partire da un nodo.
Dato un grafo, assumiamo per il momento che esista un solo nodo publisher. Tale nodo, prima di poter pubblicare qualunque forma di informazione deve provvedere a creare il proprio albero di broadcast, come mostrato dalla figura seguente:



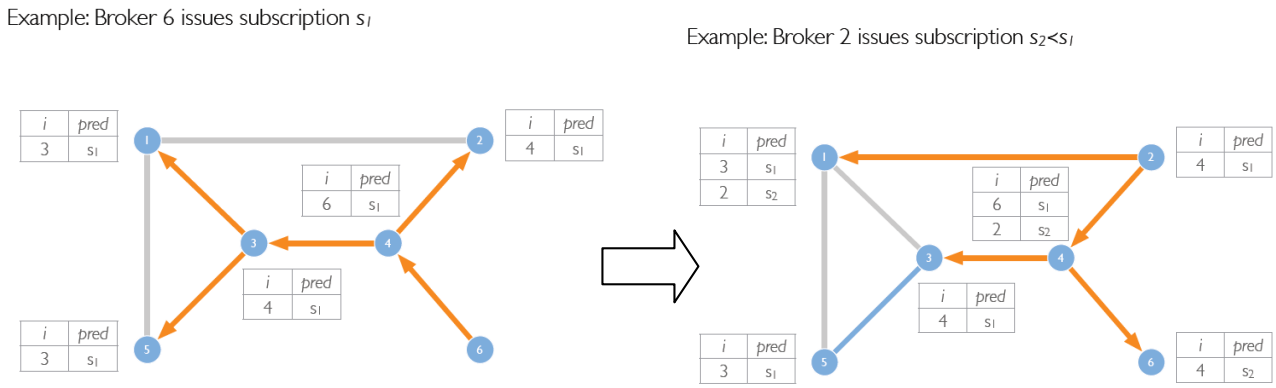
il nodo P (verde) è il publisher del grafo.
Chiaramente lo scopo è far viaggiare l'informazione solamente lungo i rami/nodi che collegano i sottoscrittori dell'evento, per garantire questo, n linea teorica, diremo che ogni nodo è in grado di calcolare la seguente funzione:

$$\text{forwarding formula} = (B(source(e), incom\_if(e)) \cup \{I_0\}) \cap F_C(e)$$

dove $B(source(e), incom\_if(e))$ è l'albero di broadcast costruito a partire dal publisher del messaggio, I_0 è l'interfaccia verso l'applicazione locale e $F_C(e)$ è la funzione che calcola i nodi interessati dall'evento. L'output di tale funzione rappresenta i nodi su cui effettuare il forwarding dell'informazione.
L'implementazione della formula di forwarding è realizzata tramite tabelle di routing stanziate in ogni nodo. Le tabelle sono continuamente aggiornate dalle sottoscrizioni e mappano i nodi che rappresentano i next hop per i messaggi ricevuti. Ad esempio il nodo 8 potrà avere nella sua tabella i nodi 10, 11, 12 come next hops dei messaggi ricevuti dal nodo 3 secondo l'albero di broadcast blu.
In Siena le sottoscrizioni espresse dai nodi sono di tipo content-based e sono rappresentate da un insieme di elementi definiti da *nome, tipo, valore*. I tipi contemplati possono essere i più svariati, ad ogni modo, per semplicità assumeremo che lo spazio degli eventi è rappresentato da un unico attributo numerico x, su cui è possibile definire vincoli utilizzando gli operatori <=>.
Per prima cosa mostreremo la forma delle tabelle di routing nei nodi e successivamente mostreremo come mantenerle aggiornate.



si noti come ogni nodo ha una riga per ogni interfaccia su cui si ha responsabilità (la responsabilità è chiaramente data dall'albero di broadcast.). In ogni riga è archiviata la tupla nodo – condizione, che esprime i vincoli di forwarding funzione delle sottoscrizioni espresse.
Per poter tenere aggiornato l'albero di distribuzione due approcci sono possibili, aggiornamenti *push* ed aggiornamenti *pull*, nel primo caso le tabelle vengono aggiornate tramite un avviso a partire dal sottoscrittore a risalire fino al publisher, nel secondo caso invece, viene periodicamente avviata una procedura di aggiornamento che consente la potatura o l'aggiunta di rami di distribuzione. Ogni approccio ha i propri vantaggi, in un caso abbiamo reattività immediata, nell'altro invece abbiamo possibilità di costruire path aggregate ovviando al mantenimento di una riga per ogni possibile nodo su cui effettuare il forwarding. In ogni caso i due approcci hanno dei criteri di funzionamento molto simili.
Di seguito al fine dare un'idea del funzionamento del protocollo mostriamo l'approccio *push* illustrando cosa accade all'introduzione di una nuova sottoscrizione:



come anticipato, l'articolo assume valore anche per le valutazioni sperimentali condotte. Il piano di test proposto definisce uno schema utile per qualunque protocollo pub/sub, al fine di avere una stima dettagliata e proporre analisi comparative con altre implementazioni. Secondo A. Carzaniga et al. i punti chiave da testare su un protocollo pub/sub sono i seguenti:

1 – il protocollo consegna i messaggi ai nodi interessati?
2 – il protocollo previene il forwarding inutile?
3 – il protocollo produce un overhead ragionevole?

I carico con cui stressare il protocollo è anch'esso un fattore importante, gli autori, indipendentemente dal rate di emissione degli eventi e dal rate di emissione delle sottoscrizioni suggeriscono che realisticamente un evento incontra tra il 5 e il 15% delle sottoscrizioni del sistema, e non supera mai il limite del 25%. Tale caratterizzazione è fondamentale, gli autori infatti tengono a sottolineare che l'uso di un sistema pub/sub ha senso se e solo se le percentuali medie di interesse agli eventi sono quelle sopra citate, in caso contrario, l'overhead imposto dal pub/sub non giustifica il mancato uso di un più semplice protocollo di broadcast.
Un confronto importante è quello condotto tra gli errori di consegna introdotti dall'uso del protocollo *push* e dal protocollo *pull*. I test hanno mostrato come un approccio push-based, a seguito di un breve bootstrap non commetta più errori di consegna, mentre invece, il più generoso pull-based mantenga una percentuale di falsi-positivi (consegna di eventi indesiderati) pari a non oltre il 10%. Il numero di messaggi scambiati per il mantenimento dell'albero di distribuzione è chiaramente funzione degli aggiornamenti che vengono compiuti nel sistema: nel caso di basse frequenze di aggiornamento l'overhead di gestione del push-based è pressoché nullo, tuttavia, per frequenze realistiche il numero di messaggi scambiati è assolutamente paragonabile, il che renderebbe la scelta dell'approccio da usare particolarmente difficile. Dal punto di vista della scalabilità è comunque il pull-based a vincere, e i motivi sono sostanzialmente due:

- gli avvisi in modalità push possono andare persi
- la dimensione delle tabelle di routing cresce in maniera particolarmente sensibile rispetto al pull-based che rimane costante

Il sunto dell'articolo è il seguente: Scribe costituisce il primo protocollo pub/sub content based conosciuto, con ragionevoli caratteristiche di scalabilità. I test sperimentali condotti definiscono le linee guida base per ogni protocollo della stessa famiglia che intenda basarsi su di un modello di sottoscrizione basata sul contenuto e i risultati ottenuti rappresentano un punto di riferimento per tutta la ricerca del settore.

Impatto sulla società
L'event driver è ovunque, portare esempi come il protocollo push della mail, l'RSS, la domotica e altro ancora, chiedi a leonardo la statistica.

Implementazioni commerciali
DDS

Prospettive di ricerca
Notevoli: dato che l'event-driven è ovunque…bisogna migliorarne la scalabilità! Sbilanciamoci a favore dell'interest clustering. QoS.

Conclusioni/progetto di ricerca
Il pub/sub è fico, ma bisogna migliorarne la scalabilità. La mia idea è di combinare le potenzialità mostrate da alcune overlay network al fine di potenziare la dell'interest clustering in termini di scalabilità e resistenza al churn. Proporre un approccio combinato tra topic-based e content based può essere la soluzione?