Pacemaker2: Fighting Malicious Peers with Pulse Gossip Protocols

September 16, 2009

Abstract

1 Introduction

2 Overview

3 Model

4 Pacemaker2 in Detail

5 Practibility (PlanetLab)

6 Scalability and Resistance to Churn

7 Analysis of Tree Redundancy

8 Applications

8.1 Monitoring of Availability

8.2 Timestamp of Documents

9 Related Work

10 Conclusion

References

Server at round i

```
let phase^i = Seeding;
let seed^i = random_seed();
let S^i_seed = sign( <i, seed^i>, KS_priv);
let M^i_seed = Seed(i, seed^i, S^i_seed);
let map^i = ∅
∀q ∈ NS_server, send(q, M^i_seed);

wait T;

phase^i = Idle;
H^i = Hash(map^i);
let S^i_pulse = sign( <i, seed^i, H^i>, KS_priv );
let M^i_pulse = Pulse(i, seed^i, H^i,
                      0, { 0 → map^i }, S^i_pulse);
∀q ∈ NS_server, send(q, M^i_pulse);
```

Figure 1:

Server or peer receiving SeedReply (i, $\mathbf{X}_q$) from peer q:

```
if phase^i = Seeding then
  map^i = {q →X_q}⊕ map^i;
end if
```

Figure 2:

Every $\Delta << T$ seconds, on peer p:

```
if phase^i = Seeding then
  let H^i = H(map^i);
  let M^i_seedreply = SeedReply(i, H^i );
  nreplies^i = nreplies^i + 1;
  replies^i[ nreplies^i ] = (H^i, map^i);
  ∀q ∈ NS_p, send(q, M^i_seedreply);
end if
```

Figure 3:

Peer p receiving $\mathbf{M}^i_{seed}$ = Seed (i,$\mathbf{seed}^i$,$\mathbf{S}^i_{seed}$):

```
if verify_sig( S^i_seed, <i, seed^i>, KS_priv) then
    ∀q ∈ NS_p, send(q, M^i_seed);

    let S^i_p = sign( <i, seed^i>, K^p_priv);
    let map^i = {p → H(S^i_p)}
    nreplies^i = 0;
    phase^i = Seeding;
end if
```

Figure 4:

Peer p receiving Pulse(i, $\mathbf{seed}^i$, $\mathbf{H}^i$, level, $\mathbf{tree}^i$, $\mathbf{S}^i_{pulse}$):

```
if
    verify_sig(S^i_pulse, <i, seed^i, H^i >, KS_pub )
then
    phase^i = Idle;
    included^i = ⊥;
    if
      ∃ { p → H^i } ∈ tree^i[level] and
      ∃ n | replies[n] = (H^i, oldmap^i) and
      included^i < n and
      verify_tree(H^i, level, tree^i)
    then
      let tree'^i = tree^i ⊕ { level+1 → oldmap^i };
      M^i_pulse = (i, seed^i, H^i, level+1, tree'^i, S^i_pulse)
      History[i] = M^i_pulse;
      included^i = n;
      ∀q ∈ children^p, send(q, Pulse M^i_pulse);
    end if
end if
```

Figure 5:

Peer p sending to q its availability at round i:

```
let bits^i = new bitfield[N_t];
for x in [1..N_t]
  if History[i - N_t + x] = ∅ then
    bits^i[x] := 0
  else
    bits^i[x] := 1
  end if
end for
let S^i_avail = sign( < i, bits^i >, K^p_priv )
let M^i_avail = (i, bits, S^i_avail);
send( q, Availability M^i_avail )
```

Figure 6:

Peer p receiving Challenge(i, nonce) from peer q:

```
let M^i_pulse = History[i]
send( q, Proof M^i_pulse)
```

Figure 7:

Peer q receiving from peer p Proof (i, nonce, $\mathbf{K}^i_{pub}$, $\mathbf{trees}^i$, $\mathbf{S}^i_p$, $\mathbf{S}^i_{pulse}$, $\mathbf{S}_{proof}$):

```
Peer q receiving from peer p
    Proof (i, nonce, K^i_pub, trees^i, S^i_p, S^i_pulse, S_proof):
  if
     (i, nonce, p) ∈ challenges ∧
     verify_sig(S^i_pulse,<i, seed^i,K^i_pub,trees^i[0]>,KS_pub) ∧
     verify_sig(S^i_p, <i, seed^i >, K^p_pub) ∧
     S^i_p ∈ trees^i ∧
     ∀n > 0, H(trees^i[n]) ∈ trees^i[n-1] ∧
     verify_sig(S_proof, < nonce, H^p, H^q >, K^i_pub )
  then
     good_reply(p)
  else
     bad_reply(p)
  end if
```

Figure 8: